View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Institut National Polytechnique de Toulouse (Theses)

N° d'ordre : 20227

THÈSE

présentée
pour obtenir le titre de

DOCTEUR DE L'INSTITUT NATIONAL POLYTECHNIQUE DE TOULOUSE
Spécialité : Systèmes Informatiques

par

Ali KALAKECH

Étalonnage de la sûreté de fonctionnement des systèmes d'exploitation – Spécifications et mise en œuvre

Soutenue le 8 juin 2005 devant le jury composé de :

Guy	JUANOLE	Président
Isabelle	PUAUT	Rapporteur
Carol	SMIDTS	Rapporteur
Karama	KANOUN	Directeur de thèse
Yves	CROUZET	Examinateur
Jean-Paul	BLANQUART	Examinateur
Christel	SEGUIN	Examinateur

N° d’ordre : 20227

À la plus tendre des princesses,

À Amira,

Je ne t’oublierai jamais Maman

N° d'ordre : 20227

À toute ma famille,

Baba, Sammoura et Hammoudi,

Pour leur amour et leur soutien

Avant-propos

Les travaux présentés dans ce mémoire ont été effectués au Laboratoire d'Analyse et d'Architecture des Systèmes du Centre National de la Recherche Scientifique (LAAS-CNRS). Je tiens à remercier Monsieur Jean-Claude Laprie et Monsieur Malik Ghallab, Directeurs de Recherche CNRS, qui ont assuré la direction du LAAS-CNRS depuis mon entrée, pour m'avoir accueilli au sein de ce laboratoire.

Je remercie également Messieurs David Powell et Jean Arlat, Directeurs de Recherche CNRS, responsables successifs du groupe de recherche Tolérance aux fautes et Sûreté de Fonctionnement informatique (TSF), pour m'avoir permis de réaliser ces travaux dans ce groupe.

J'exprime ma très sincère reconnaissance à Madame Karama Kanoun, Directeur de Recherche CNRS, pour m'avoir encadré et encouragé tout au long de cette thèse. Je la remercie très profondément pour la confiance qu'elle m'a témoignée, et le soutien qu'elle m'a accordé dans les difficiles moments que j'ai rencontrés durant ces travaux. Je remercie également Monsieur Yves Crouzet, Chargé de Recherche CNRS, pour m'avoir co-encadré avec Madame Kanoun et pour m'avoir calmé lors de mes nombreux moments d'énervement. Nos discussions nocturnes au laboratoire et son support technique m'ont été de grand secours. Leurs lectures attentives des différentes versions du manuscrit, leurs conseils très constructifs et leur soutien dans les moments de doute ont fortement contribué au bon déroulement des travaux présentés dans ce mémoire. Je remercie également Jean Arlat, Directeur de Recherche CNRS, pour ses conseils et sa disponibilité. Je suis très honoré d'avoir la chance de travailler avec vous.

Je remercie Monsieur Guy Juanole, Professeur à l'Université Paul Sabatier, pour l'honneur qu'il m'a fait en présidant mon jury de thèse, ainsi que :

- Monsieur Jean-Paul Blanquart, Ingénieur d'études à Astrium SAS,
- Monsieur Yves Crouzet, Chargé de Recherche CNRS,
- Madame Karama Kanoun, Directeur de Recherche CNRS,
- Madame Isabelle Puaut, Professeur à l'Univversité de Rennes,
- Madame Christel Seguin, Ingénieur de Recherche à l'ONERA-CERT,
- Madame Carol Smidts, Professeur à l'Université de Maryland (USA),

pour l'honneur qu'ils m'ont en participant à mon jury. Je remercie tout particulièrement Madame Isabelle Puaut et Madame Carol Smidts qui ont accepté la charge d'être rapporteurs.

Mes sincères remerciements vont également à Monsieur Jean-Paul Blanquart qui, par ses remarques pertinentes, a contribué à l'amélioration de la qualité de ce mémoire.

Cette thèse n'aurait pas pu avoir cette forme actuelle sans l'aide précieuse de Sandrine Car, Olivier Guitton, Ana-Elena Rugina et Philippe Rumeau qui ont directement contribué à l'aboutissement de ces travaux. Il m'est particulièrement agréable de remercier Tahar Jarboui pour ses précieux conseils et support au début de ces travaux, et pour son amitié.

Je tiens à remercier Benjamin Lusier, Ana-Elena Rugina, Ludovic courthès, Eric Alata et Ayda Saidane pour leurs relectures attentives qui ont permis d'améliorer la qualité de ce manuscrit.

Les travaux présentés dans ce mémoire s'inscrivent dans le cadre du projet européen DBench (*Dependability Benchmarking* IST-2000-25425). J'ai énormément apprécié le travail en groupe avec les partenaires du projet aux niveaux scientifique et humain. Ce fut une expérience extrêmement enrichissante.

Mes remerciements vont naturellement à tous les membres du groupe TSF, permanents doctorants et stagiaires : Ana (Este Ana-Elena, zeita frumuseti, regina bunatatii…nu te voi uita niciodata), Christina, Céline, Eric Alata (je peux t'en piquer une ????), Benji, Guillaume, Ludo, Marco, Agnan De Bonneval (chef), Mohammed Kaâniche, Vincent (mon lapin), Jérémie (heavy metal), Mouslem, Christophe (McGyver), Arno, Noreddine (noura), Magnos (amigo), François, Nicolas, Carlos Aguilar-Melchor, Mathieu, Boris et tous les autres. Je leur témoigne toute ma gratitude pour l'ambiance de travail décontractée que nous avons instaurée au sein du groupe. Je remercie vivement Joëlle Penavayre et Gina Briand, les deux secrétaires successives de groupe TSF, pour leur aide dans les innombrables tâches administratives.

Je remercie très sincèrement The Abdellatif Family (Elyes, Olfa et Slouma) et The Bennani Family (Meryem, Afef et Taha). Je vous remercie pour la deuxième famille que vous m'avez offerte et au sein de laquelle j'ai passé mes plus agréables moments en France. Je remercie également Houda Ben Attia (setna el doctora), Ayda Saidane, Anis Youssef et Houda Yaccoub pour leur grande amitié.

Je tiens également à remercier mes amis libanais à Toulouse : Momo, najla, Rony et Elie, ainsi que mes amis au Liban : Bilal (abou Raad), Oukba, Anwar, Nassim (moudir), Chadi (abou mezrabbeh), Maroun (sidna), houdd, Haj-Shhadeh brothers, Khaled Fayed (moutreb), Abed et Walid, pour leur soutien inconditionnel

Enfin, bien sûr, je remercie ma famille pour son soutien permanent, malgré la grande distance qui nous a séparé et les grandes difficultés que nous avons rencontrées. Qu'ils trouvent ici un témoignage de mon affection et de ma reconnaissance.

Sommaire

Sommaire 1

Introduction générale...5

Chapitre 1 Contexte des travaux...9

1.1 Introduction 9

1.2 Notions de sûreté de fonctionnement informatique 10

1.2.1 Concepts de sûreté de fonctionnement informatique 10

1.2.2 La prévision de fautes 11

1.2.3 L'injection de fautes 13

1.3 Étalons de performance...17

1.4 Les systèmes d'exploitation..19

1.4.1 Fonctions des systèmes d'exploitation 19

1.4.2 Classification des systèmes d'exploitation..21

1.4.3 Exemples de systèmes d'exploitation...22

1.5 Caractérisation des systèmes d'exploitation 24

1.5.1 Etalons de performance des systèmes d'exploitation 24

1.5.2 Caractérisation de la sûreté de fonctionnement des systèmes d'exploitation par injection de fautes...26

1.6 Conclusion 32

Chapitre 2 Étalonnage de la sûreté de fonctionnement 33

2.1 Introduction 33

2.2 Etalons de sûreté de fonctionnement existants..33

2.3 Cadre conceptuel pour l'étalonnage de la sûreté de fonctionnement 37

2.3.1 Dimensions de catégorisation *38*
2.3.2 Dimension de mesures *39*
2.3.3 Dimensions d'expérimentation *41*
2.4 Propriétés d'un étalon de sûreté de fonctionnement 45
2.4.1 La répétitivité *45*
2.4.2 La reproductibilité *45*
2.4.3 La représentativité *46*
2.4.4 La portabilité *48*
2.4.5 La non-intrusivité *48*
2.4.6 L'interférence *49*
2.4.7 La mise à l'échelle *49*
2.4.8 L'automatisation *49*
2.4.9 Le coût de l'étalonnage *49*
2.5 Conclusion 50

Chapitre 3 Spécification d'étalons de systèmes d'exploitation 51

3.1 Introduction 51
3.2 Spécification des étalons 52
3.2.1 Système cible et contexte d'étalonnage *52*
3.2.2 Mesures fournies par l'étalon *53*
3.2.3 L'expérimentation *59*
3.2.4 Environnement d'étalonnage et conduite d'expériences *62*
3.3 Validation des propriétés par construction 66
3.3.1 La reproductibilité *66*
3.3.2 La représentativité *66*
3.3.3 La portabilité *67*
3.3.4 La non-intrusivité *68*
3.3.5 La mise à l'échelle *68*
3.4 Conclusion 68

Chapitre 4 Étalonnage de la sûreté de fonctionnement des systèmes Windows......71

4.1 Introduction 71

4.2 Mise en œuvre 72

4.3 Mesures de l'étalon 73

4.3.1 Robustesse de l'OS...73

4.3.2 Temps de réaction de l'OS...74

4.3.3 Temps de redémarrage du système...75

4.3.4 Récapitulatif 75

4.4 Mesures complémentaires et affinement..75

4.4.1 Robustesse de l'OS...75

4.4.2 Temps de réaction de l'OS...77

4.4.3 Temps de redémarrage 82

4.4.4 Temps d'exécution de l'activité 83

4.4.5 Récapitulatif 84

4.5 Validation des propriétés par expérimentation 84

4.5.1 La reproductibilité 84

4.5.2 La répétitivité 85

4.5.3 La représentativité 85

4.5.4 La portabilité..90

4.5.5 L'interférence..90

4.5.6 L'automatisation 90

4.5.7 Le coût de l'étalonnage...91

4.6 Conclusion..92

Chapitre 5 Étalonnage de Windows et de Linux...93

5.1 Introduction 93

5.2 Mise en œuvre 93

5.3 Résultats de DBench-OS-Postmark...96

5.3.1 Mesures de l'étalon...96

5.3.2 Raffinement des mesures de l'étalon 100

5.4 Résultats de DBench-OS-JVM 104

5.4.1 Mesures de l'étalon 104

5.4.2 Raffinement des mesures 107

5.5 Comparaison 110

5.6 Conlcusion 112

Conclusion générale 115

Annexe 1 Exemple d'implémentation des mécanismes de substitution de paramètres et d'observation 119

Annexe 2 Raffinement des mesures de robustesse par rapport à l'état final de *Postmark* 121

Annexe 3 Validation des résultats de DBench-OS-Postmark et DBench-OS-JVM par rapport à l'ensemble de fautes 123

Annexe 4 Temps de redémarrage de Windows et de Linux avec DBench-OS-Postmark 127

Références bibliographiques 129

Table des matières 137

Introduction générale

Les systèmes informatiques sont aujourd'hui devenus incontournables dans les différents secteurs de notre vie quotidienne. Ils permettent aussi bien de réaliser des tâches triviales (courrier électronique, bureautique…) que de résoudre des problèmes les plus complexes (systèmes de commande avioniques ou nucléaires…). Souvent confronté à des contraintes économiques et temporelles, le développement des systèmes informatiques s'appuie depuis longtemps sur l'utilisation des composants commerciaux sur étagère. Des composants matériels comme les microprocesseurs et les mémoires ont tout d'abord été principalement utilisés, mais suite aux évolutions technologiques, cette tendance d'utiliser des composants commerciaux s'est également portée sur les composants logiciels. En particulier, l'utilisation de composants matériels et logiciels sur étagère est devenue aujourd'hui une pratique usuelle dans le développement des systèmes informatiques. Ces composants sont dénommés composants commerciaux sur étagère ou COTS (acronyme du terme anglais équivalent : « *Commercial-Off-The-Shelf* »). Par la suite, l'appelation de COTS s'est étendue pour rassembler tous les composants sur étagère, d'origine commerciale ou non, en devenant l'acronyme de « *Component-Off-The-Shelf* ».

L'utilisation des composants logiciels *COTS* dans le processus de développement de la plupart des systèmes informatiques étant de plus en plus courante, la justification du choix des composants *COTS* sélectionnés pour être intégrés dans un système est devenue une question importante. Ce choix peut être basé sur plusieurs critères tels que le prix, la performance, la durée de vie. La performance a longuement été considérée comme le critère déterminant, parce qu'elle permet d'assurer un fonctionnement du système plus rapide et donc plus efficace. Cependant, l'utilisation de *COTS* dans des systèmes de plus en plus critiques nécessite l'inclusion du critère de la *sûreté de fonctionnement*.

Dans le passé, le terme « système critique » a été utilisé pour désigner les systèmes avioniques, nucléaires, et tout autre système dont la défaillance peut entraîner des catastrophes humaines. De nos jours, un satellite de communication, un serveur de commerce électronique et bien d'autres systèmes informatiques qui passent inaperçus dans notre vie quotidienne deviennent de plus en plus critiques, dans la mesure où leur défaillance peut induire des catastrophes économiques.

Actuellement, les systèmes d'exploitation s'imposent comme une solution particulièrement attractive dans le processus de développement des systèmes informatiques, et en conséquence, sont considérés comme composants critiques parce que leur défaillance peut entraîner la défaillance du système entier. En effet, un système d'exploitation est un

élément majeur de la conception des systèmes informatiques vu le rôle fondamental qu'il joue en offrant les services nécessaires à l'exécution de l'ensemble des applications.

Pour sélectionner le système d'exploitation le mieux adapté pour faire partie du système global, le développeur d'un système critique intégrant un système d'exploitation a besoin d'informations caractérisant la sûreté de fonctionnement des différents systèmes candidats. Ces informations peuvent être obtenues de différentes sources, comme les développeurs du système d'exploitation, les données statistiques de sa vie opérationnelle ou les informations obtenues par expérimentation. Les travaux effectués dans notre mémoire s'inscrivent dans le contexte d'évaluation de la sûreté de fonctionnement par expérimentation. Nous nous intéressons plus particulièrement à l'étalonnage par expérimentation de la sûreté de fonctionnement des systèmes d'exploitation.

Étalonner la sûreté de fonctionnement d'un système informatique consiste à évaluer ses mesures de sûreté de fonctionnement, ou de performance en présence de faute, de manière expérimentale ou basée sur la modélisation et l'expérimentation. Cette évaluation doit permettre de comparer, de façon non ambiguë, la sûreté de fonctionnement des solutions alternatives. Cette non ambiguïté et la confiance dans les résultats de l'étalon sont assurées par un ensemble de propriétés que l'étalon doit satisfaire. Ces propriétés concernent la représentativité, la reproductibilité, la répétitivité, la portabilité, etc.

Sur le plan conceptuel, un étalon de sûreté de fonctionnement des systèmes informatiques est constitué d'un ensemble de méthodes qui permettent d'évaluer la sûreté de fonctionnement de ces systèmes. En pratique, un étalon de sûreté de fonctionnement peut être disponible sous forme d'un exécutable prêt à être installé sur le système à étalonner, ou bien mis en œuvre à partir d'une spécification. Il est essentiel qu'une personne qui souhaite étalonner la sûreté de fonctionnement d'un système informatique soit capable, à partir de cette spécification, de réaliser toutes les étapes nécessaires pour obtenir les différentes mesures souhaitées. Cette spécification doit inclure, entre autres, la description des frontières du système cible à étalonner, la définition bien détaillée des mesures à évaluer, la description des différentes étapes de la mise en œuvre expérimentale, etc. Elle peut également inclure des exemples de code source ou même des spécifications d'outils qui permettent de faciliter la mise en œuvre de l'étalonnage. En d'autres termes, cette spécification doit être suffisamment claire pour permettre de :

- Mettre en œuvre l'étalon de sûreté de fonctionnement du système cible,
- Bien comprendre et interpréter les résultats fournis.

Les travaux présentés dans ce mémoire portent en particulier sur l'étalonnage de la sûreté de fonctionnement des systèmes d'exploitation à usage général. La sûreté de fonctionnement d'un système d'exploitation peut être caractérisée par plusieurs attributs qui n'ont pas forcément le même niveau d'importance suivant le domaine d'application dans lequel l'OS est utilisé. Ceci est lié au fait que suivant le domaine d'application, des attributs de la sûreté de fonctionnement peuvent être mis en avant plus que d'autres. Par exemple, la fiabilité et la disponibilité sont des attributs demandés dans tous domaines d'applications, alors que la sécurité-confidentialité est principalement requise dans le cas des applications

Internet. L'étalon de sûreté de fonctionnement que nous proposons est avant tout un étalon de robustesse de l'OS. Le glossaire standard de terminologie d'ingénierie logicielle de IEEE (IEEE Std 610.12-1990) définit la robustesse comme la quantification de la capacité d'un système ou d'un composant à se comporter correctement en présence d'entrées invalides ou de conditions environnementales stressantes.

Pendant leur vie opérationnelle, les systèmes d'exploitation sont confrontés à différentes sortes de requêtes erronées malgré lesquelles ils doivent continuer à exercer leurs fonctions. Ces requêtes erronées peuvent être originaires de plusieurs sources, telles que la plate-forme matérielle, les pilotes de périphériques ou les applications qui s'exécutent dessus. Les résultats présentés dans [Gray 1990] montrent que les défaillances des systèmes informatiques sont majoritairement provoquées par des fautes logicielles, alors qu'elles étaient longtemps considérées comme la conséquence des fautes matérielles. D'autres études comme [Chou 2001] ou [Sullivan 1991] mettent l'accent sur l'ampleur des fautes résiduelles dans les composants logiciels des systèmes informatiques. Nos travaux portent ainsi plus particulièrement sur l'étalonnage de la sûreté de fonctionnement des systèmes d'exploitation par rapport aux requêtes erronées du logiciel applicatif. Pour simuler ce comportement erroné, nous utilisons les techniques d'injection de fautes.

Notre travail s'appuie donc, d'une part, sur l'étalonnage de performance, et d'autre part, sur l'évaluation de la sûreté de fonctionnement. Les études des états de l'art de ces deux domaines nous servent par la suite pour la définition d'un cadre conceptuel permettant de définir un étalon de sûreté de fonctionnement d'un système informatique, ainsi que les principales propriétés qu'il doit satisfaire. Ainsi, les étalons de sûreté de fonctionnement des systèmes informatiques peuvent être spécifiés à partir de ce cadre conceptuel, ce qui permet par la suite de comparer la sûreté de fonctionnement de ces systèmes.

Ce mémoire est structuré en cinq chapitres :

Dans le chapitre 1, nous explorons les axes de recherche sur lesquels s'appuient nos travaux : l'évaluation de la sûreté de fonctionnement informatique par injection de fautes et l'étalonnage de performance. Tout d'abord, nous décrivons les concepts de base de la sûreté de fonctionnement avant de nous intéresser plus particulièrement aux techniques d'injection de fautes. Ensuite, nous présentons l'état de l'art dans le domaine d'étalonnage de performance des systèmes informatiques. Après, nous présentons les notions fondamentales des systèmes d'exploitation, avant de passer en revue, dans la dernière partie, des exemples de leur caractérisation par des étalons de performance ou par injection de fautes.

Dans le chapitre 2, nous présentons les travaux menés sur l'étalonnage de la sûreté de fonctionnement des systèmes informatiques. Nous présentons tout d'abord les principaux étalons existants de sûreté de fonctionnement des systèmes informatiques. Nous décrivons ensuite un cadre conceptuel pour le développement des étalons de sûreté de fonctionnement

tel que nous l'avons défini dans le projet DBench[1] à partir de l'état de l'art réalisé au premier chapitre. Nous donnons également les propriétés que des étalons de sûreté de fonctionnement doivent satisfaire.

Dans le chapitre 3, nous spécifions trois étalons que nous proposons pour étalonner la sûreté de fonctionnement des systèmes d'exploitation à usage général. Ces étalons seront utilisés dans les chapitres suivants pour étalonner la sûreté de fonctionnement des systèmes d'exploitation des familles Windows et Linux. La première partie de ce chapitre est destinée à présenter la spécification des mesures à fournir et des conditions d'expérimentation nécessaires pour la réalisation de l'étalonnage. À partir de cette spécification, certaines des propriétés définies dans le deuxième chapitre peuvent être directement vérifiées sans avoir besoin d'expérimentation.

Dans le chapitre 4, nous présentons la mise en œuvre et les résultats de l'étalonnage de la sûreté de fonctionnement de trois systèmes d'exploitation de la famille Windows. Il s'agit de Windows NT4, Windows 2000 et Windows XP. Nous vérifions ensuite, à l'aide du prototype développé, que les étalons définis satisfont bien les propriétés qui nécessitent une expérimentation.

Dans le chapitre 5, la sûreté de fonctionnement des systèmes d'exploitation appartenant à deux familles différentes (Windows et Linux) est étalonnée et comparée. Pour chacun d'eux, deux étalons de sûreté de fonctionnement sont utilisés pour atteindre cet objectif. Ils diffèrent par l'activité utilisée pour solliciter le système. Après avoir présenté la mise en œuvre de ces deux étalons, nous consacrons le reste du chapitre à présenter et commenter les résultats obtenus.

---

[1] Projet européen DBench (Dependability Benchmarking), IST-2000-25425
http://www.laas.fr/dbench/

Chapitre 1 Contexte des travaux

1.1 Introduction

Nos travaux portent sur l'étalonnage de la sûreté de fonctionnement des systèmes d'exploitation. Cette thématique s'inspire de deux domaines de recherche distincts, qui ont jusqu'à présent évolué de façon indépendante : la sûreté de fonctionnement informatique et l'étalonnage.

De nombreux travaux de recherche se sont déjà intéressés à la sûreté de fonctionnement des systèmes informatiques et à différentes méthodes permettant son évaluation. Parmi celles-ci, nous distinguons les méthodes analytiques et les méthodes expérimentales. Dans le cadre de nos travaux, nous avons adopté une approche d'évaluation expérimentale basée sur l'injection de fautes.

Les travaux actuels portant sur l'étalonnage servent principalement à caractériser la performance des systèmes informatiques. Nous avons étudié ces étalons de performance afin de nous en inspirer pour développer des méthodes d'étalonnage de la sûreté de fonctionnement de systèmes informatiques.

Dans ce chapitre, nous introduisons les concepts de base de ces deux domaines : la sûreté de fonctionnement et l'étalonnage de performance. Nous passons en revue les attributs, entraves et moyens de la sûreté de fonctionnement, avant de nous intéresser en particulier à un de ses moyens : la prévision des fautes. Nous mettons l'accent sur les techniques d'*injection de fautes* qui permettent de caractériser la sûreté de fonctionnement des systèmes informatiques avant de présenter les notions de base de l'étalonnage de performance des systèmes informatiques.

Nous présentons ensuite les systèmes d'exploitation (qui font l'objet de notre étude) en termes de fonctions et classification avant de consacrer la dernière section à la caractérisation des systèmes d'exploitation. Les systèmes d'exploitation peuvent être caractérisés par leur performance et par leur sûreté de fonctionnement. Nous mettons l'accent sur des exemples d'étalon de performance et sur les travaux en sûreté de fonctionnement dédiés à la caractérisation des systèmes d'exploitation basés essentiellement sur l'injection de fautes.

1.2 Notions de sûreté de fonctionnement informatique

Les travaux présentés dans ce mémoire s'inscrivent dans le cadre de l'évaluation expérimentale de la sûreté de fonctionnement informatique. Dans cette section, nous passons en revue les principaux éléments de cette notion de sûreté de fonctionnement. Nous présentons tout d'abord les concepts relatifs à la sûreté de fonctionnement (attributs, entraves et moyens) avant de nous intéresser en particulier à l'un de ses moyens : la prévision des fautes. Nous conclurons cette section en présentant l'injection de fautes, une approche au service de la prévision et de l'élimination des fautes.

La terminologie utilisée tout au long de cette section est extraite du « Guide de la Sûreté de Fonctionnement » [Laprie 1995].

1.2.1 Concepts de sûreté de fonctionnement informatique

La *sûreté de fonctionnement* d'un système informatique est la propriété qui permet à ses utilisateurs de placer une confiance justifiée dans le *service* qu'il leur délivre. Le *service* délivré par un système est son comportement tel que perçu par son, ou ses *utilisateurs* ; un *utilisateur* est un autre système (humain ou physique) qui interagit avec le système considéré. La sûreté de fonctionnement informatique s'articule en trois principaux axes : les *attributs* qui la caractérisent, les *entraves* qui empêchent sa réalisation et enfin les *moyens* de l'atteindre (la prévention, la tolérance, l'élimination et la prévision des fautes).

Attributs de la sûreté de fonctionnement

Les attributs de sûreté de fonctionnement sont définis pour exprimer les propriétés de sûreté de fonctionnement du système. L'importance de chacun de ces attributs est relative aux applications auxquelles le système est destiné. On peut distinguer :

- La *disponibilité*, le fait que le système soit prêt à l'utilisation,
- La *fiabilité*, la continuité du service,
- La *sécurité-innocuité*, la non-occurrence de conséquences catastrophiques pour l'environnement,
- La *confidentialité*, la non-occurrence de divulgations non-autorisées de l'information,
- L'*intégrité*, la non-occurrence d'altérations inappropriées de l'information,
- La *maintenabilité*, l'aptitude aux réparations et aux évolutions.

Entraves à la sûreté de fonctionnement

Les entraves à la sûreté de fonctionnement sont les circonstances indésirables – mais non inattendues – causes ou résultats de la non-sûreté de fonctionnement. Dans l'ensemble des entraves, on distingue la *défaillance* du système qui survient lorsque le service délivré dévie de l'accomplissement de la fonction du système, l'*erreur* qui est la partie de l'état de système qui est susceptible d'entraîner une *défaillance*, et enfin la *faute* considérée comme la cause adjugée ou supposée de l'*erreur*.

Moyens pour la sûreté de fonctionnement

Il s'agit des méthodes et techniques permettant de fournir au système l'aptitude à délivrer un service conforme à l'accomplissement de sa fonction, et de donner confiance dans cette aptitude. Le développement d'un système sûr de fonctionnement passe par l'utilisation combinée de ces méthodes qui peuvent êtres classées en :

- *Prévention de fautes* : comment empêcher l'occurrence ou l'introduction de fautes ;
- *Tolérance aux fautes* : comment fournir un service à même de remplir la fonction du système en dépit des fautes ;
- *Élimination des fautes* : comment réduire le nombre et la sévérité des fautes ;
- *Prévision des fautes* : comment estimer la présence, la création et les conséquences des fautes.

La *prévention* des fautes et la *tolérance* aux fautes peuvent être vues comme des moyens d'obtention de la sûreté de fonctionnement, c'est-à-dire comment procurer au système l'aptitude à fournir des services conformes aux fonctions attendues.

L'*élimination* des fautes et la *prévision* des fautes peuvent être vues comme constituant les moyens de la **validation** de la sûreté de fonctionnement, c'est-à-dire comment avoir confiance dans l'aptitude du système à fournir des services conformes à l'accomplissement de sa fonction.

Nos travaux s'inscrivent dans le cadre de la validation de la sûreté de fonctionnement, et plus spécifiquement dans les techniques utilisées pour la prévision des fautes, dont nous présentons les concepts de base dans le paragraphe suivant.

1.2.2 La prévision de fautes

La prévision des fautes est réalisée en effectuant des évaluations du comportement du système par rapport à l'occurrence des fautes et à leur activation. Ces évaluations peuvent être conduites de deux façons, selon le but à atteindre :

1) L'*évaluation ordinale*, ou *qualitative*, destinée à identifier, classer et ordonner les défaillances, ou les méthodes et techniques mises en œuvre pour les éviter ;
2) L'*évaluation probabiliste*, ou *quantitative*, destinée à évaluer en termes de probabilités le degré de satisfaction de certains des attributs de la sûreté de fonctionnement ; ces attributs sont alors vus comme des mesures de la sûreté de fonctionnement du système.

La vie d'un système est perçue par son, ou ses utilisateurs comme une alternance entre deux états du service par rapport à l'accomplissement de la fonction du système :

1) Service correct, où le service délivré accomplit la fonction du système ;
2) Service incorrect, où le service délivré n'accomplit pas la fonction du système.

Une défaillance est donc une transition de service correct à service incorrect ; la transition de service incorrect à service correct est une *restauration*. La quantification de l'alternance

entre *service correct* et *service incorrect* du système est utilisée pour quantifier de différentes mesures de sûreté de fonctionnement:

- *Fiabilité* : mesure de la délivrance continue d'un service correct, ou, de façon équivalente, du temps jusqu'à défaillance ;
- *Disponibilité* : mesure de la délivrance d'un service correct par rapport à l'alternance « service correct – service incorrect » ;
- *Maintenabilité* : mesure du temps jusqu'à restauration depuis la dernière défaillance survenue, ou, ce qui est équivalent, de la délivrance continue d'un service incorrect.

D'autres mesures peuvent être considérées en combinant des mesures de performance et des mesures de sûreté de fonctionnement. C'est le cas de systèmes multi-performants, où l'on peut distinguer plusieurs modes de délivrance de service depuis la pleine capacité jusqu'à l'interruption totale. Ces mesures sont habituellement dénommées par le terme de *performabilité* [Meyer 1993].

Dans le domaine de l'évaluation quantitative, nous distinguons plusieurs techniques pour évaluer la sûreté de fonctionnement des systèmes informatiques. En fonction des niveaux d'abstraction considérés pour le système évalué (modèles axiomatiques, de simulation ou physiques), deux ensembles des techniques peuvent être considérés :

1) Les techniques à base de modèles, ou ***modélisation analytique***, qui consistent à construire des modèles mathématiques permettant ainsi de quantifier les mesures de sûreté de fonctionnement,
2) Les techniques à base de mesures expérimentales, ou ***évaluation expérimentale***, qui consistent à observer le comportement du système à évaluer, collecter les informations nécessaires, les analyser et les traiter pour obtenir enfin les mesures quantitatives désirées. Nous distinguons deux grandes familles de ces techniques. Il s'agit de la collecte des données statistiques provenant de l'observation de systèmes opérationnels et de l'injection de fautes.

La modélisation analytique s'appuie sur la description analytique ou graphique du comportement du système. Les mesures de sûreté de fonctionnement sont évaluées par l'affectation des probabilités aux paramètres des modèles. Parmi les principaux types de modèles largement utilisés, on peut citer notamment les diagrammes de fiabilité, les arbres de fautes et les graphes d'états (chaînes de Markov, réseaux de Petri stochastiques, etc.). Ces méthodes d'évaluation ont été largement utilisées pour prendre des décisions lors de la phase de conception des systèmes.

Le problème de l'évaluation analytique peut être divisé en trois phases étroitement liées :

- Le choix des mesures de la sûreté de fonctionnement à évaluer,
- La construction d'un, ou des modèles pour décrire le comportement du système,
- Le traitement du, ou des modèles pour calculer les mesures désirées.

La technique d'observation de systèmes informatiques pendant leur vie opérationnelle consiste à collecter toutes les informations caractérisant leur sûreté de fonctionnement pour évaluer les mesures de sûreté de fonctionnement à partir de ces informations. En effet, ces mesures peuvent être utilisées pour élaborer des modèles analytiques à partir desquels les mesures de sûreté de fonctionnement sont évaluées. Il faut noter que l'analyse des relevés statistiques offre des renseignements significatifs sur la vie opérationnelle des systèmes. Pour avoir des mesures représentatives, il est essentiel d'avoir un volume de données recueillies assez important, collecté pendant une longue période d'observation. Cette technique nécessite généralement trois étapes :

1) La collecte des données, où il faut définir quoi et comment collecter ;
2) La validation des données, ce qui revient à analyser les données recueillies pour vérifier si elles sont correctes et cohérentes ;
3) Le traitement des données, la dernière étape où l'analyse statistique des données validées aura lieu pour évaluer des mesures de sûreté de fonctionnement.

L'injection de fautes est une technique bien répandue dans le domaine de l'évaluation expérimentale de la sûreté de fonctionnement. Elle consiste à tester la réaction d'un système informatique face à des conditions anormales, en lui communiquant délibérément des fautes. L'objectif d'une telle méthode est d'étudier les effets des fautes réelles qui peuvent affecter le système évalué pendant son exploitation opérationnelle.

Dans le cadre des travaux présentés dans ce mémoire, nous nous intéressons à l'évaluation expérimentale des systèmes informatiques, et plus spécifiquement, à la méthode d'injection des fautes dont nous présentons les principaux concepts dans le paragraphe suivant.

1.2.3 L'injection de fautes

Dans le contexte de la sûreté de fonctionnement, l'injection de fautes est une technique utilisée au service de l'élimination des fautes et de la prévision des fautes [Arlat 1989]. Concernant l'élimination des fautes, l'injection de fautes est considérée parmi les moyens les plus adoptés, à côté des techniques de test, pour la *vérification* du système ; c'est la première étape de l'*élimination de fautes* qui consiste à vérifier si le système satisfait des propriétés appelées des *conditions de vérification*, et qui précède deux étapes nommées *diagnostic* et *correction.* Sur le plan de la *prévision des fautes*, l'injection de fautes constitue un moyen privilégié de l'évaluation expérimentale permettant de caractériser des mécanismes de tolérance aux fautes du système évalué (par exemple, le taux de couverture des mécanismes de tolérance aux fautes ou la latence de détection d'erreur).

Parmi les objectifs que les expériences d'injection de fautes visent à atteindre, nous distinguons :

- La correction et l'amélioration des mécanismes de tolérance aux fautes. L'injection de fautes permet d'élaborer des solutions architecturales comme les mécanismes de confinement d'erreurs afin de mieux détecter et tolérer les erreurs [Salles 1999],

- La prévision de l'impact de fautes résidant dans le système sur son comportement, en évaluant des mesures caractérisant l'efficacité des mécanismes de tolérance aux fautes.

Plusieurs techniques d'injection de fautes peuvent être utilisées pour valider un système tolérant aux fautes. Deux principaux critères permettent de classer ces différentes techniques : le niveau d'abstraction du système cible et la forme d'application de l'injection. Concernant le système cible, il peut être représenté sous une forme physique (c'est-à-dire sous sa version finale ou sous la forme d'un prototype), ou par un modèle de simulation représentant sa structure ou son comportement. Quant à la forme d'application de l'injection, les fautes peuvent être directement injectées sur les composants matériels au moyen d'altérations physiques ou électriques, ou elles peuvent être injectées au niveau informationnel (par exemple altération de variables booléennes ou du contenu de mémoires).

Il est évident que sur les modèles de simulation, seules les fautes de type informationnel peuvent être appliquées, ce qui donne lieu à une technique d'injection de fautes appelée *simulation*, comme dans le cas de MEFISTO-L [Arlat 1999]. En revanche, les deux types de fautes peuvent être appliqués aux modèles physiques, ce qui donne lieu à la technique d'injection des fautes par des moyens physiques (ou HWIFI pour HardWare Implemented Fault Injection), et la technique d'injection de fautes par logiciel (SWIFI pour SoftWare Implemented Fault Injection) quand il s'agit des fautes de type informationnel appliquées sur des modèles physiques, détaillée dans le paragraphe 1.2.3.2 .

Après la description des attributs de l'injection de fautes dans le paragraphe suivant, nous focalisons sur l'injection de fautes par logiciel.

1.2.3.1 Les attributs de l'injection de fautes

L'injection de fautes correspond à une séquence de tests, où des fautes sont délibérément introduites dans le système évalué. Cette séquence de tests est censée évaluer certains mécanismes de sûreté de fonctionnement du système, et elle est caractérisée par un *domaine d'entrée* et un *domaine de sortie* [Arlat 1990].

Le *domaine d'entrée* correspond à l'ensemble des fautes à injecter, nommé ***F***, et à l'activité ***A*** qui va activer le système selon le domaine d'application visé.

Le *domaine de sortie* correspond à l'ensemble des observations ***R*** (*relevés*) et à l'ensemble des mesures ***M*** dérivées à partir des ensembles ***F***, ***A*** et ***R***.

Les ensembles ***F***, ***A***, ***R*** et ***M*** forment les attributs de la séquence des expériences d'injection de fautes. En pratique, le choix de la mesure ***M*** à évaluer influence le choix des autres ensembles. Chaque expérience correspond à un point de l'ensemble **{*F*x *A*x *R*}** : une faute de l'ensemble ***F*** est injectée dans le système cible activé par ***A*** ; l'ensemble des observations ***R*** est choisi d'une façon qui, en le combinant avec la faute injectée et l'activité, permet de calculer la mesure désirée.

Pour caractériser les attributs de l'injection de fautes, il faut considérer deux points de vue principaux : 1) le niveau d'abstraction du système cible dans lequel les fautes sont injectées, et 2) les buts recherchés par l'injection des fautes. Les travaux de ce mémoire s'inscrivent

dans le cadre de l'évaluation expérimentale de la sûreté de fonctionnement des systèmes informatiques en utilisant les techniques d'injection de fautes sur des systèmes cibles physiques. Ainsi, les attributs de l'injection de fautes peuvent être caractérisés de la manière suivante :

Le domaine d'entrée

Dans le domaine de la prévision de fautes, le principal souci réside dans les faits que l'ensemble ***F*** des fautes à injecter doit être statistiquement représentatif de l'ensemble des fautes qui peuvent affecter le système, et qu'un grand nombre d'erreurs doit être injecté pour améliorer le niveau de confiance dans les résultats fournis.

L'ensemble des fautes à injecter comprend deux grandes catégories de fautes : les fautes physiques, et les fautes de type logiciel (du niveau informationnel). Le choix entre ces deux types de fautes dépend de la nature du système cible à évaluer. Par exemple, seules des fautes physiques peuvent être utilisées pour évaluer la sûreté de fonctionnement des prototypes matériels et des fautes logicielles pour des prototypes uniquement logiciels.

Ces fautes peuvent être de nature permanente, c'est-à-dire qu'elles peuvent se manifester pendant toute la durée de l'expérience, ou bien des fautes transitoires, c'est-à-dire qu'elles sont présentes pendant un certain intervalle de temps d'exécution de l'expérience. Une autre dimension à considérer est l'endroit d'injection de la faute. La faute peut être injectée à l'interface d'entrée, à l'interface de sortie ou à l'intérieur du système cible à évaluer.

La sélection de l'ensemble ***A*** des activités pour activer le système dépend largement de l'objectif de l'évaluation. Dans le domaine de la prévision des fautes, cet ensemble d'activités doit absolument représenter le domaine d'application dans lequel le système cible sera exploité. Par conséquent, l'ensemble ***A*** est constitué des programmes d'application représentatifs et de leurs entrées. La synchronisation entre l'activité du système et la faute injectée dépend de la technique d'injection de fautes mise en œuvre.

Le domaine de sortie

Le domaine de sortie inclut deux grands ensembles : l'ensemble ***R*** correspondant aux observations à collecter lors des expériences d'injection de fautes et l'ensemble ***M*** incluant toutes les mesures à évaluer. L'obtention des mesures de l'ensemble ***M*** se fait à partir de la combinaison des ensembles ***F***, ***A*** et ***R***.

Dans le contexte de prévision de fautes, l'ensemble ***M*** correspond à des mesures probabilistes ou statistiques, par exemple l'occurrence des événements spécifiques (le nombre de détections observées lors de la réalisation d'un certain nombre d'injections) ou la durée entre ou pour atteindre ces événements (la durée de dormance d'une faute ou latence d'une erreur). Ces événements spécifiques correspondent aux combinaisons de différents états du système ; ces états sont prédéfinis a priori par la prévision du comportement du système.

Comme c'est déjà expliqué, l'objectif des expériences d'injection de fautes est la caractérisation du comportement du système en présence de fautes. Une des principales caractéristiques quantitatives de la tolérance aux fautes est la notion de *couverture*. La

couverture est définie comme la probabilité conditionnelle que, étant donné une faute dans le système, le système parvient à la traiter correctement. En pratique, il est important de souligner que la notion de *couverture* est appliquée vis-à-vis des erreurs, en particulier pour caractériser les différentes étapes nécessaires au système pour traiter l'erreur : détection, diagnostic et recouvrement d'erreur.

1.2.3.2 Injection de fautes par logiciel

Les fautes physiques ont été longuement utilisées pour caractériser la sûreté de fonctionnement des systèmes informatiques comme avec *Messaline* [Arlat 1989], *Rifle* [Madeira 1994] ou *FIST* [Gunneflo 1989]. Des études comme celle de [Lee 1995] ont montré que la principale cause de défaillance des systèmes informatiques est due aux fautes d'origine logicielle. En plus, plusieurs études ont montré que les erreurs provoquées par des simples inversions de bit étaient similaires à celles qui sont provoquées par les méthodes d'injection de fautes physiques [Kanawati 1995] [Fuchs 1996].

Comme nous l'avons précédemment présentée, cette technique consiste à injecter des fautes au niveau informationnel dans des systèmes cibles physiques. De nos jours, la tendance est d'utiliser cette technique (FERRARI [Kanawati 1995], Xception [Carreira 1998] et MAFALDA [Rodríguez 1999], [Chevochot 2001]), vu la facilité relative de sa mise en œuvre par rapport à l'injection physique des fautes, où il faut faire face à la complexité croissante des matériels utilisés dans le développement des systèmes informatiques.

Cette technique d'injection de fautes, de plus en plus améliorée et développée, met en avant un nombre important d'avantages par rapport aux autres méthodes. Comparée aux approches matérielles, le coût de sa mise en œuvre est assez réduit. D'autre part, cette technique est plus adaptée à la validation des systèmes et des mécanismes de haut niveau d'abstraction. Enfin, cette technique est beaucoup plus souple que les autres, et plus facile à porter sur d'autres systèmes.

En dépit de ces avantages, les techniques d'injection de fautes par logiciel présentent un certain nombre d'inconvénients qui sont résumés ci-dessous :

- Il est impossible d'injecter des fautes dans des endroits inaccessibles pour le logiciel,
- L'instrumentation du logiciel perturbe l'exécution de l'activité sur le système cible et peut même changer sa structure originelle, ce qu'on appelle l'intrusivité,
- La faible résolution du temps des logiciels engendre des problèmes de fidélité par rapport aux résultats fournis. Ce problème n'est pas perceptible dans le cas de fautes de grande latence comme les fautes dans la mémoire. Cependant, dans le cas des fautes de faible latence comme les fautes dans les processeurs, l'approche peut être incapable d'observer certains états erronés.

Les techniques d'injection de fautes par logiciel sont classées selon le moment de l'injection des fautes :

- À la compilation : les instructions du programme sont modifiées avant l'exécution de son image binaire. Cette méthode ne nécessite pas de logiciel d'injection

supplémentaire pendant l'exécution et ne cause aucune perturbation dans le système cible pendant l'exécution,

- À l'exécution : pendant l'exécution, un mécanisme est nécessaire pour le déclenchement de l'injection de fautes. Ce déclencheur est basé sur un temporisateur ou une exception,
- Hybride : cette approche est utilisée pour injecter à la compilation des fautes se comportant comme des fautes transitoires à l'exécution. L'idée est d'ajouter un injecteur minimal pendant l'exécution dont le rôle est d'autoriser ou non les fautes.

1.3 Étalons de performance

L'étalon de performance d'un système informatique est défini comme étant un test, ou un ensemble de tests, conçu pour comparer la performance de systèmes informatiques. Ces étalons sont utilisés dans le but de sélection (« quel est le meilleur système pour moi ? »), de contrôle de performance (« comment puis-je régler mon système pour améliorer sa performance ? »), ou de prévision de performance (« combien cette idée va améliorer la performance de mon système ? ») [Lucas 1971].

Il existe plusieurs types d'étalonnage qu'il est possible de regrouper dans trois grandes catégories [UQTR 2004]. D'abord, on trouve l'étalonnage *interne* dont l'objectif est de comparer les différents composants d'un système afin d'améliorer ses points faibles. Un autre type d'étalonnage est celui de type *compétitif* qui permet de comparer les informations d'un système à celles d'autres systèmes semblables. Enfin, nous citons l'étalonnage *générique* dont le but est de comparer les données d'un système à des « normes » génériques, obtenues par l'accord des différentes communautés sous forme de spécifications et d'exigences à satisfaire, ou bien à partir des systèmes considérés comme des références au niveau mondial.

Les étalons de performance font généralement face à deux critiques. D'abord, les utilisateurs des résultats de ces étalons n'ont pas confiance dans la manière avec laquelle la performance d'un système est déterminée, s'ils estiment par exemple que l'étalon utilisé ne couvre pas le profil d'exécution souhaité. La deuxième critique concerne l'inexactitude des mesures fournies par l'étalon, pour des raisons de passage à l'échelle par exemple.

Pour répondre à ces exigences, un étalon de performance doit satisfaire certaines propriétés [Gray 1993] :

- La pertinence des mesures qu'il fournit ;
- La portabilité : un étalon de performance doit être facile à installer sur différents systèmes et architectures informatiques ;
- La simplicité de ses mesures qui doivent être compréhensibles ;
- La mise à l'échelle: un étalon de performance de systèmes informatiques doit pouvoir être appliqué à différents systèmes et architectures informatiques de tailles différentes.

L'activité soumise au système par l'étalon de performance constitue un élément important de l'étalon. L'analyse des étalons existants montre qu'il existe plusieurs types d'activités, allant d'activités purement synthétiques à des activités correspondantes à des applications réelles (activités réalistes). Les activités synthétiques correspondent à des activités artificielles développées de manière à ce que leurs caractéristiques soient similaires à des activités réalistes. Les activités réalistes sont importantes dans la mesure où elles permettent d'augmenter la confiance dans les mesures fournies par l'étalon.

La plupart des étalons de performance actuels fournissent un ensemble réduit de mesures (certains d'entre eux fournissent une seule mesure), pour des raisons de simplicité et d'acceptabilité de l'étalon. Cependant, il est évident qu'un ensemble de mesures exprimerait mieux les différents aspects relatifs à la performance d'un système informatique. Essentiellement, deux types de mesures sont fournis : le temps d'exécution et la quantité du travail effectué, exprimée en MIPS (Million Instructions Per Second). Il existe encore d'autres mesures qui caractérisent le coût, et qui se trouvent principalement dans les étalons de performance des systèmes transactionnels.

Les étalons de performance peuvent être classés selon d'autres critères comme l'étendue des résultats, ou la nature de l'étalon. Pour l'étendue des mesures, nous distinguons les macro-étalons et les micro-étalons. Les macro-étalons permettent de caractériser globalement la performance du système, tandis que les micro-étalons fournissent des mesures caractérisant des mécanismes internes bien spécifiques de ce système. En ce qui concerne la nature de l'étalon, ce dernier peut être fourni sous forme d'un logiciel complet prêt à s'exécuter, ou bien sous forme d'une spécification dans une documentation qui explique les étapes à suivre pour implémenter l'étalon et l'activer correctement pour obtenir les mesures souhaitées.

De nos jours, plusieurs centaines d'étalons de performance de systèmes informatiques existent sur le marché. Néanmoins, un étalon de performance doit représenter un standard accepté par la communauté, et par conséquent, son importance est directement liée au nombre d'entités qui l'utilisent. Les constructeurs d'étalons se sont rassemblés en organisations internationales à but non lucratif pour définir des standards d'étalons de performance. Les deux organisations les plus connues sont SPEC (Standard Performance Evaluation Corporation) [SPEC] et TPC (Transaction Processing Performance Council) [TPC]. Les grandes compagnies de l'industrie informatique sont membres de ces deux organisations.

Les mesures fournies par les étalons produits par ces deux organisations sont différentes. Les étalons de SPEC produisent des mesures de temps qui correspondent au temps d'exécution d'un programme bien déterminé par l'étalon. Alors que les étalons de TPC fournissent d'une part une mesure de performance et d'autre part une mesure du rapport prix/performance. La mesure de performance correspond à une mesure de débit. De plus, les étalons produits par SPEC sont sous la forme d'un code source standard, extrait d'applications réelles, et modifié par SPEC pour, entre autres, améliorer sa portabilité. Les étalons de TPC sont définis par des spécifications bien détaillées. Dans ce qui suit, nous mettons l'accent sur les étalons de performance de l'organisation TPC, et plus

particulièrement sur un étalon spécifique (TPC-C) largement adopté dans ce mémoire et dans le cadre du projet DBench.

TPC définit des étalons de performance des systèmes transactionnels. Etant défini par des spécifications bien détaillées, pour exécuter un étalon fourni par TPC, il faut mettre en œuvre sa spécification sur le système cible. Dans la pratique, des exemples sont fournis avec la spécification et il suffit de les adapter à l'architecture du système cible. Parmi les étalons que propose TPC, nous trouvons l'étalon TPC-C conçu pour mesurer la capacité d'un système à fonctionner comme serveur de base de données pour le traitement de transactions en ligne (OLTP : On-Line Transaction Processing). Il simule un environnement informatique où un groupe d'utilisateurs, appelés clients TPC-C, exécute des transactions de saisie de commande sur une base de données. Cinq types de transactions sont pris en compte : passage d'une nouvelle commande, règlement ou encaissement du paiement, contrôle de l'état d'une commande, suivi de la livraison et suivi des stocks. TPC-C spécifie l'ensemble des tableaux de la base de données, les caractéristiques des données de ces tableaux et l'ensemble de transactions à effectuer. TPC-C fournit deux mesures : le taux de transaction exprimé en nombre de transactions par minute et le prix associé à chaque transaction.

1.4 Les systèmes d'exploitation

Un système d'exploitation est un composant fondamental et critique dans les systèmes informatiques. Nous présentons tout d'abord les objectifs et les fonctionnalités des systèmes d'exploitation, avant de décrire les différents critères utilisés pour les classer. Pour conclure, nous passons en revue quelques exemples des systèmes d'exploitation.

1.4.1 Fonctions des systèmes d'exploitation

Le système d'exploitation (ou *OS* pour *Operating System*), encore appelé système opératoire, est une couche logicielle qui assure l'interface entre les composantes matérielles du système informatique d'une part et les usagers du système informatique d'autre part. Il fait partie des logiciels de base les plus importants d'un système informatique.

Ainsi, un système d'exploitation est généralement conçu pour mettre à la disposition de l'utilisateur (un utilisateur final ou une application) un ensemble de services qui s'appuient sur les ressources de la machine physique. Ces services peuvent être basiques, comme la reconnaissance de l'entrée du clavier ou l'affichage sur l'écran des différentes données, ou très avancés, comme s'assurer que seuls les utilisateurs autorisés ont le droit d'utilisation.

L'architecture des systèmes d'exploitation est généralement découpée en deux couches logicielles distinctes : 1) l'espace *utilisateur* où les applications et les programmes fonctionnent, et 2) l'espace *noyau* ou *superviseur* où s'exécute la partie la plus critique du système d'exploitation. Le noyau est la seule partie du système d'exploitation qui a le droit d'accéder et de contrôler les différentes ressources matérielles. Le rôle du noyau est de fournir aux applications une abstraction de la couche matérielle sous forme de *services*. Chaque programme s'exécutant dans un système informatique possède son propre espace

d'adressage. L'intérêt principal de cette séparation est la sécurité, puisqu'un programme s'exécutant en mode utilisateur ne peut perturber ni l'exécution des autres programmes du mode utilisateur ni l'exécution du noyau. L'inconvénient majeur est le surcoût temporel engendré par les changements fréquents d'espace entre les applications et le noyau.

Les services fournis par différents systèmes d'exploitation ne sont pas totalement identiques, cependant des classes de fonctions se retrouvent généralement dans chacun de ces systèmes [Nutt 2000] :

- *La gestion des processus et des ressources* correspond à la création, l'ordonnancement et la terminaison des processus d'un programme, ainsi qu'à la gestion des ressources du système nécessaires pour l'exécution de ces processus. Un processus correspond à une série d'instructions d'un programme.
- *La gestion de la mémoire* est responsable de l'allocation de la mémoire disponible pour les différents processus.
- *La gestion des fichiers* offre aux utilisateurs une vision homogène et structurée des données et des ressources : disques, mémoires, périphériques. Elle est aussi responsable de la création, la destruction, l'organisation des fichiers et la gestion de droit d'accès.
- *La gestion des périphériques* permet de communiquer avec les périphériques extérieurs (clavier, écran, imprimante…).

Les systèmes modernes intègrent par ailleurs d'autres fonctionnalités, comme le système de fenêtrage graphique et l'interconnexion des différentes machines et des différents systèmes par des réseaux locaux ou étendus.

L'ensemble de services d'un noyau d'un système d'exploitation est fourni aux applications par l'intermédiaire des *appels système*. La terminologie utilisée pour dénommer l'ensemble de ces appels système est l'*API* (*Application Programming Interface*). Les appels système traversent la frontière entre l'espace noyau et l'espace utilisateur, depuis l'espace d'adressage de l'application vers l'espace d'adressage du noyau. D'autre part, le noyau du système d'exploitation dispose de *l'ABI* (*Application Binary Interface*) pour gérer l'accès aux ressources matérielles du système, et utilise le *DPI* (*Driver Programming Interface*) pour dialoguer avec les pilotes de périphériques. Ces différentes interfaces sont représentées dans la Figure 1-1.

Pour résumer, les systèmes d'exploitation ont pour principales fonctions de détecter et d'initialiser les composants matériels, d'offrir aux pilotes de périphériques une interface d'accès (*DPI*), de partager les ressources du système, via l'*ABI*, entre les différents programmes qui sont actifs, et de permettre aux applications et aux utilisateurs de dialoguer avec les matériels et les périphériques disponibles à travers l'*API*.

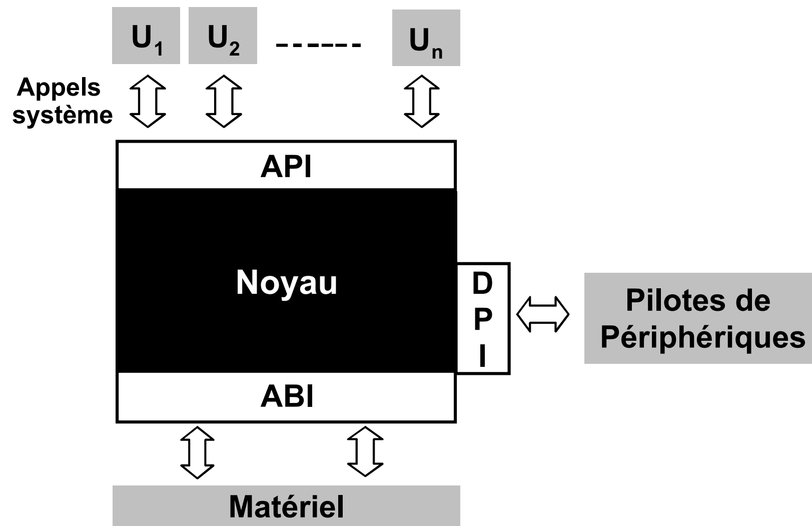


Figure 1-1 Schéma d'un système d'exploitation

1.4.2 Classification des systèmes d'exploitation

Dans ce paragraphe, nous présentons la classification des différents types de systèmes d'exploitation, en se basant dans un premier temps sur les critères *temporels*.

- **Traitement par lots** (*batch processing*) : le système d'exploitation traite les applications les unes après les autres. Le temps de réponse dans ce type de systèmes n'est pas un critère prioritaire.

- **Temps réel** (*real time*) : où les contraintes de temps sont très importantes. Une application qui s'exécute dans un tel système doit être en mesure de répondre à des événements (asynchrones ou périodiques) en provenance de l'environnement qu'elle contrôle (par exemple la température dans un réacteur nucléaire, l'altitude ou la vitesse d'un avion, la trajectoire d'une fusée, la distance d'un robot par rapport à un obstacle...) et en assurer le traitement en un temps déterminé. Le temps de réaction face à l'événement est très important, d'où leur qualification de *« systèmes réactifs »*. Les systèmes temps-réel se caractérisent d'ailleurs par les conséquences dramatiques de retards éventuels dans l'obtention de résultats d'applications qui s'y exécutent (contrôles dans une centrale nucléaire ou un dans avion par exemple).

- **Temps partagé** (*time sharing*) : le système d'exploitation prend en compte plusieurs applications de manière concurrente, et permet l'entrelacement de leur exécution. Ce type de systèmes donne l'illusion d'une exécution concurrente des différentes applications : chaque utilisateur se croit seul derrière son poste, grâce à des changements fréquents du travail courant. Les délais de réponse doivent être relativement courts à l'échelle humaine. D'autres aspects peuvent aussi être ajoutés, comme les **systèmes multiprocesseurs** qui supportent l'exécution d'un ou plusieurs programmes sur plus d'un processeur, et les **systèmes multibrins** (*multi thread*) qui

permettent l'exécution parallèle des différentes parties du même programme. Dans nos travaux, nous nous intéressons en particulier à cette classe de systèmes d'exploitation.

D'autres critères peuvent également êtres pris en considération pour classer les différents systèmes d'exploitation, notamment le modèle d'*architecture du noyau* de ces systèmes : on distingue les **noyaux monolythiques** et les **micro-noyaux**. Dans le premier cas, les différents services sont regroupés dans le noyau (*gestion des processus*, *gestion de mémoire*, *système de fichiers*…), alors que dans le cas des micro-noyaux la plupart de ces services se trouvent sous forme de processus qui s'exécutent à côté du noyau. Généralement, les noyaux monolythiques ne font pas la différence entre les différents services qu'ils fournissent : l'ensemble de ces services s'exécutent en mode *noyau*. En revanche, les micro-noyaux fournissent des services sous forme de processus qui peuvent être lancés ou arrêtés pendant l'exécution du noyau ; l'avantage dans ce dernier cas est la petite taille du noyau, ce qui les favorise par rapport aux autres types de noyaux en termes de performance. Suivant ce principe, une autre approche a été proposée, l'**exonoyau,** qui consiste à utiliser un noyau n'offrant que la protection et le multiplexage des ressources matérielles. La gestion des ressources matérielles est à la charge des applications.

Enfin, des travaux actuels de recherche portent sur une autre famille de systèmes d'exploitation, appelés **flexibles**. Ces systèmes doivent leur dénomination à la possibilité de pouvoir modifier leurs services ou architectures après leur construction. Cette modification peut avoir lieu au cours de l'exécution du système, de manière *dynamique*, ou avant cette exécution, de manière *statique* par compilation ou édition de liens.

Dans nos travaux, nous nous sommes intéressés à l'étude de la sûreté de fonctionnement des systèmes d'exploitation d'après leur domaine d'utilisation, en particulier les systèmes d'exploitation à usage général (ou *GPOS* pour *General Purpose Operating System*). Un tel système est capable d'exécuter une grande variété de programmes, comme des applications commerciales ou des programmes de bureautique. En revanche, d'autres systèmes d'exploitation, dits *dédiés*, sont destinés à accomplir un objectif bien spécifique, comme le contrôle de l'ouverture et de la fermeture des portes, le contrôle d'une pompe, etc. Les systèmes dédiés ne sont pas capables de supporter l'exécution d'autres applications comme des outils de développement ou des jeux.

1.4.3 Exemples de systèmes d'exploitation

Il existe de nombreux systèmes d'exploitation basés sur un micro-noyau. L'objectif de tels systèmes est de fournir les services sous forme de processus. Ces services peuvent ainsi être démarrés et arrêtés dynamiquement. Les noyaux résultant de ce modèle d'architecture bénéficient généralement d'une petite taille, ce qui facilite leur maintien et améliore leur performance, mais requiert une communication efficace entre le noyau et les différents processus. En effet, chaque demande de service passe par le noyau qui appelle à son tour le processus, provoquant ainsi un double changement d'espace d'adressage. La première génération des micro-noyaux était basée sur la communication entre processus (*IPC* : *Inter Process Communication*) implémentée par l'échange des messages. *Mach* fait partie de cette première génération des micro-noyaux, développé initialement à l'Université de Carnegie Mellon [Accetta 1986] [Loepere 1991], il a été construit comme une base au

dessus de laquelle peuvent être émulés simultanément d'autres systèmes d'exploitations de type Unix ou autre. Comme tous les autres micro-noyaux, *Mach* fournit des services de base comme la gestion de processus, la gestion de la mémoire, les communications et les services d'entrée/sortie. Les techniques d'IPC basées sur l'échange de messages se sont avérées inefficaces au niveau de performance, à cause de très coûteuses opérations de recopie des messages. Par conséquent, une deuxième génération de micro-noyaux est apparue en gardant les mêmes concepts des micro-noyaux d'origine, mais avec l'utilisation de techniques plus performantes pour l'IPC (support pour le changement direct de contexte, utilisation des registres pour des messages courts, …). *QNX* [Hildebrand 1992] est un exemple de cette nouvelle génération des micro-noyaux : développé par la société *QNX Software Systems*, ce système d'exploitation est dédié aux applications temps réel et aux applications embarquées. De très petite taille (12 KO), il met seulement en œuvre le multithreading, la gestion des interruptions, les mécanismes IPC et la gestion de la mémoire. Il est très extensible et peut être exécuté sur des machines mono-processeur ou sur des serveurs multi-processeurs. Les domaines d'utilisation de *QNX* comportent l'automatisation industrielle, l'informatique de poche, l'électronique, etc. Il est fréquemment cité comme une référence en matière de fiabilité et de performance de système.

Les systèmes d'exploitation de la famille *Unix* font partie des systèmes d'exploitation à usage général et utilisent un noyau monolithique, mettant en œuvre tous les services système dans le noyau. Parmi l'ensemble des systèmes d'exploitation de la famille Unix, nous pouvons citer *Linux* [Torvalds 1994] développé initialement par Linus Torvalds, *Aix* [IBM 1990] développé par IBM et *Solaris* [Microsystems 1991] développé par Sun Microsystems. Certaines améliorations introduites dans Linux, comme l'utilisation de modules insérés dynamiquement et s'exécutant en mode noyau, offrent une extensibilité similaire à celle offerte par les micro-noyaux. Cependant, ces extensions forment des sources potentielles d'instabilité puisque la seule protection offerte par les systèmes de type *Unix* est l'isolation des applications et des modules de mode *noyau* dans des espaces d'adressage différents. Aucune protection n'a été prévue à l'intérieur du même espace d'adressage. Par conséquent, les extensions forment des sources potentielles d'instabilité pour le noyau.

Les systèmes d'exploitation de la famille Windows NT utilisent une couche HAL (Hardware Abstraction Layer) pour accéder au matériel [Solomon 2000]. La couche HAL correspond à un nano-noyau générique, dans le sens où son objectif est d'offrir de façon homogène et portable les abstractions de la couche matérielle. L'implémentation d'une couche HAL dépend du matériel sous-jacent, mais cette dépendance est masquée pour l'utilisateur. Au-dessus de cette couche HAL, mais toujours dans le mode noyau, s'exécutent un micro-noyau basé sur le noyau *Mach*, puis des serveurs offrant les services de base du système d'exploitation. Les pilotes des périphériques sont gérés par le gestionnaire d'entrées/sorties, et s'exécutent dans des espaces d'adressage séparés des serveurs, contrairement au système d'exploitation Linux.

1.5 Caractérisation des systèmes d'exploitation

Les systèmes d'exploitation ont été la cible de nombreux travaux de caractérisation en termes de performance ou de sûreté de fonctionnement. Pendant longtemps, la performance des systèmes d'exploitation a été considérée comme le principal critère de leur sélection, auquel s'est rajouté récemment le critère de sûreté de fonctionnement. Parmi les principales méthodes d'évaluation expérimentale de sûreté de fonctionnement, les techniques d'injection de fautes ont pris une place importante.

Dans un premier temps, nous présentons les travaux destinés à étalonner la performance des systèmes d'exploitation. Ensuite, nous présentons quelques travaux effectués dans le domaine de la caractérisation de la sûreté de fonctionnement des systèmes d'exploitation par injection de fautes.

1.5.1 Etalons de performance des systèmes d'exploitation

Dans ce paragraphe, nous passons en revue quelques exemples d'étalons de performance des systèmes d'exploitation. Nous utilisons le critère d'étendue des mesures qu'ils fournissent pour les classer en micro et macro-étalons.

Micro-étalons

L'objectif des travaux publiés dans [Ousterhout 1990] était d'analyser pourquoi les systèmes d'exploitation ne sont pas aussi rapides que le matériel. Pour répondre à cette question, plusieurs micro-étalons de performance ont été utilisés pour caractériser certains services des systèmes d'exploitation tels que la gestion de la mémoire et la gestion des fichiers. Les mesures fournies par ces étalons sont soit des mesures temporelles soit le nombre d'instructions exécutées par seconde. Les applications utilisées sont synthétiques, comme des programmes qui invoquent l'appel système *getpid* (retourne l'identificateur du processus qui l'appelle), ou d'autres activités qui permettent d'ouvrir un fichier, lire son contenu et le fermer etc. Parmi les suggestions proposées pour améliorer la performance des systèmes d'exploitation, nous citons l'amélioration de la bande passante et l'utilisation de méthodes asynchrones pour les opérations lecture-écriture des fichiers sur le disque dur.

PenguinoMeter est un étalon de performance dédié à mesurer le taux de transfert des données avec un serveur de fichiers Linux. Les concepteurs de cet étalon ont développé une application similaire à celle de *IoMeter* [IoMeter] pour activer le système à tester. *IoMeter*, développé par Intel, est un étalon de performance pour les serveurs de fichiers, assez célèbre dans le monde des systèmes d'exploitation Windows. Dans [Bryant 2001], et après avoir montré l'équivalence entre l'activité de leur étalon et celle de *IoMeter*, les auteurs ont comparé les deux systèmes Linux 2.4.9 et Microsoft Windows 2000. Leurs résultats montrent que le système d'exploitation Linux est plus performant que Windows.

Lmbench [McVoy 1996] est une suite de micro-étalons simples et portables, ayant pour objectifs de caractériser les systèmes UNIX/POSIX et Linux. Dernièrement, ces étalons ont été portés sur Windows NT. En général, ces étalons fournissent deux types de mesures caractérisant la latence et la bande passante de certains mécanismes internes des systèmes d'exploitation. Plus spécifiquement, ils permettent de calculer la latence de lecture de la

mémoire, la latence de changement de contexte, la latence de création d'un processus, le temps d'exécution d'un appel système, la bande passante de lecture/écriture de la mémoire, la bande passante de la communication inter-processus et la bande passante des opérations d'entrées/sorties du disque dur.

Vu la grande variété de mesures qu'ils fournissent, les étalons de *Lmbench* peuvent être utilisés par les concepteurs ainsi que par les utilisateurs des systèmes d'exploitation.

Hbench [Brown 1997] est également une suite des micro-étalons permettant de caractériser des mécanismes de base des systèmes d'exploitation. Bien que cet étalon soit dérivé de *Lmbench*, les deux étalons divergent dans la philosophie et le code source. *Hbench* a amélioré le traitement des résultats et a rendu les étalons existants plus reproductibles.

Postmark [Katcher 1997] est un micro-étalon de performance de systèmes de fichiers. Développée en langage C pour modéliser une charge de type « courrier électronique » et « nouvelles sur le réseau », l'activité de Postmark effectue des opérations sur le système de fichiers, telles que la création et l'effacement de petits fichiers, l'effacement et l'insertion des données dans ces fichiers. Cet étalon sera revisité plus tard dans ce manuscrit (cf. chapitres 3 et5).

Macro-étalons

Contest est un macro-étalon de performance dédié à la comparaison de noyaux Linux. Il permet de mesurer le temps nécessaire à la compilation d'un noyau Linux en présence d'activités courantes de la machine, telles que la gestion de la mémoire et la gestion des processus. Pour chacune des exécutions de l'étalon, quatre mesures sont évaluées : 1) le temps nécessaire pour compiler le noyau, 2) le pourcentage d'usage du processeur pour réaliser la compilation du noyau, 3) le temps moyen pour exécuter l'ensemble des activités, et enfin 4) le pourcentage d'usage du processeur par ces activités. À la fin de l'exécution de l'étalon, une autre mesure est calculée comme étant le rapport du temps de compilation obtenu en présence d'activités sur le temps obtenu en absence d'activités. Selon *Contest*, le noyau ayant le rapport de temps le plus petit, et le pourcentage d'usage du processeur pour la compilation le plus grand est le plus efficace.

Dans [Microsoft 2000], le département de recherche de Microsoft propose des conseils pour configurer et régler le système d'exploitation Windows 2000, ceci afin d'assurer aux utilisateurs une meilleure performance quand le système d'exploitation est utilisé comme serveur des fichiers, serveur web ou installé sur un ordinateur personnel. Pour cela, ils ont utilisé un ensemble des macro-étalons pour mesurer sa performance. Dans la suite de ce paragraphe, nous présenterons quelques uns de ces étalons.

Pour mieux configurer Windows 2000 destiné pour des ordinateurs personnels, deux étalons de performance ont été utilisés : *Sysmark* [Sysmark] et *Business Winstone* [BWinstone]. Ces étalons permettent de mesurer la performance globale des systèmes d'exploitation de la famille Windows, utilisés pour exécuter les tâches les plus courantes d'un utilisateur final. Pour activer le système d'exploitation, ils exécutent des applications utilisées quotidiennement sur les machines personnelles, comme Microsoft Word, Microsoft Power Point, Winzip, etc.

L'étalon *Sysmark* fournit le « temps de réponse » comme mesure fondamentale de performance. Ce temps est défini comme la moyenne des temps nécessaires à l'ordinateur pour réaliser différentes activités. *Business Winstone* utilise un système d'unités pour caractériser les systèmes d'exploitation. Plus le nombre d'unités est levé, plus le système d'exploitation est performant. Le nombre d'unités résultant de l'étalonnage est comparé à celui d'une *machine de base*, définie par le concepteur de l'étalon comme une machine qui prend 10 unités de performance. Ainsi, un système avec un résultat d'étalonnage de 20 unités est deux fois plus performant que la *machine de base*.

Les entreprises qui ont développé ces deux étalons ne cessent de les améliorer, surtout pour rendre l'activité soumise à l'ordinateur le plus proche possible de la réalité. Ces entreprises lancent chaque année une nouvelle version améliorée de leurs produits. Par exemple, *Sysmark* a intégré, dans sa version 2002, le temps nécessaire à un utilisateur final pour réfléchir.

WebBench [Webbench], développé également par VeriTest, est un étalon dédié à l'étalonnage de la performance des serveurs Web. WebBench utilise des machines clients qui envoient des requêtes au serveur pour des fichiers statiques (des fichiers HTML, des images .GIF) placés sur le serveur lors de l'installation de l'étalon, ou pour une combinaison de fichiers statiques et d'exécutables dynamiques qui permettent de produire des données qu'un serveur retourne aux clients. Quand le serveur répond à la requête d'un client, ce dernier sauvegarde des informations telles que le temps nécessaire au serveur pour répondre à la requête et le taux de transfert des données en bits par seconde. À la fin des tests, WebBench calcule les résultats finaux de performance du serveur.

1.5.2 Caractérisation de la sûreté de fonctionnement des systèmes d'exploitation par injection de fautes

Plusieurs travaux utilisent l'injection de fautes pour la caractérisation de la sûreté de fonctionnement de composants logiciels, tels que les micro-noyaux temps réel [Chevochot 2001] [Arlat 2002b], les systèmes d'exploitation à usage général [Tsai 1996] [Koopman 1999] et les intergiciels [Marsden 2001] [Pan 2001]. Dans cette section, nous passons en revue quelques exemples de techniques d'injection de fautes dédiées à la caractérisation de la sûreté de fonctionnement des systèmes d'exploitation. En particulier, nous présentons deux outils dédiés à la caractérisation de la robustesse des systèmes d'exploitation, dénommés MAFALDA et Ballista, suivis par les travaux effectués dans le cadre du projet DBench (à l'exception de ceux présentés dans cette mémoire). Enfin, nous présentons l'outil FINE dont l'objectif est d'étudier les canaux de propagation d'erreurs dans les systèmes d'exploitation.

1.5.2.1 MAFALDA

MAFALDA (*Microkernel Assessment by Fault injection Analysis and Design Aid*), développé au LAAS-CNRS, est un outil d'injection de fautes par logiciel dont les objectifs sont d'aider à caractériser les modes de défaillance d'un exécutif à base de micro-noyau et de faciliter la mise en œuvre de mécanismes d'empaquetage du noyau pour améliorer sa

sûreté de fonctionnement [Fabre 1999] [Salles 1999]. Le premier micro-noyau considéré a été Chorus. Dans l'outil MAFALDA, deux modèles de fautes sont considérés :

- Des fautes d'interaction par sollicitation intensive et erronée des services du micro-noyau via son interface,
- Des fautes simulant l'effet des fautes physiques ; elles visent les segments de code et de données du micro-noyau,

L'injection de fautes d'interaction consiste à intercepter les appels au micro-noyau et à inverser un bit de façon aléatoire dans la chaîne des paramètres avant de poursuivre l'exécution de l'appel dans le micro-noyau. Une activité synthétique est associée à chaque composant fonctionnel du micro-noyau cible (par exemple : synchronisation, ordonnancement, communication…). Deux modules logiciels sont chargés dans chaque micro-noyau cible : le module d'interception et le module d'injection. Les paramètres caractérisant une campagne d'injection de fautes sont stockés sur la machine hôte dans deux fichiers d'entrée : le descripteur de campagne et le descripteur de l'activité.

Le banc de test MAFALDA est composé de deux entités :

1) Un ensemble de dix machines cibles, exécutant des micro-noyaux cibles en parallèle ;
2) Une machine hôte dont le rôle est de définir, d'exécuter, de contrôler les expériences sur les machines cibles et d'analyser les résultats.

Après corruption des paramètres des appels système du micro-noyau, 22% des erreurs dans le composant fonctionnel de communication (COM) ont été bien détectées par les mécanismes de détection d'erreurs du micro-noyau. Le blocage du micro-noyau et des applications a été observé dans moins de 1 % des cas. Cependant, 40% des erreurs ont abouti à un service incorrect, c'est-à-dire à une défaillance de l'application et 36% des cas ont abouti à des conséquences non observées malgré l'activation des fautes. Par ailleurs, les modes de défaillance du composant fonctionnel de synchronisation (SYN) sont différents : seules des défaillances d'applications (87%) ont été observées lors de la corruption de ses primitives. Les autres 13% correspondent à des conséquences non observées. En effet, les mécanismes de synchronisation mis en œuvre par Chorus ne vérifient pas les paramètres d'entrée. C'est pourquoi des mécanismes de confinement d'erreurs ont été développés spécifiquement pour le composant fonctionnel de synchronisation. Des expériences d'injection de fautes ont été menées après l'insertion de ces mécanismes de confinement dans le noyau et ont montré l'efficacité de ces mécanismes de confinement. En effet, aucune défaillance d'application n'a été observée en présence de ces mécanismes lors de l'injection de fautes dans les paramètres des primitives de synchronisation.

MAFALDA permet également d'injecter des fautes dans les segments de mémoire du micro-noyau. Le but est d'évaluer l'efficacité des mécanismes internes de détection d'erreur et de tolérance aux fautes. MAFALDA procède à une distribution aléatoire et uniforme des fautes injectées sur l'espace d'adressage du module cible, mais ne réalise que les expériences où les fautes sont effectivement activées. L'étude expérimentale a montré la différence des distributions de modes de défaillance, d'une part entre les différents

composants fonctionnels, et d'autre part suite à l'injection de fautes dans les segments de texte et de données.

En plus de la caractérisation des modes de défaillance du noyau, MAFALDA permet de déduire les canaux de propagation existant entre différents modules fonctionnels et de calculer la latence de détection des exceptions levées par le processeur. L'évaluation des canaux de propagation d'erreurs à l'intérieur du micro-noyau Chorus a révélé des relations de dépendance fonctionnelle entre les différents modules.

Les premiers résultats ont ciblé la caractérisation du micronoyau Chorus. Par la suite, l'outil a également été utilisé pour analyser LynxOS. Les derniers travaux concernant MAFALDA ont donné lieu à une version améliorée (MAFALDA-RT) incluant des aspects temps-réel (temps de réponse, dépassement de limites temporelles…) [Rodríguez 1999] [Rodríguez 2002].

1.5.2.2 BALLISTA

BALLISTA, développé à l'université de Carnegie Mellon, est un outil d'évaluation de la robustesse des composants logiciels sur étagère [Koopman 1997] [Koopman 1999]. Il combine des approches de test de logiciel et des approches d'injection de fautes. Il est basé sur la combinaison de cas de tests prenant en compte les paramètres valides et invalides des appels système et des fonctions d'un composant logiciel. Bien que l'outil ait été conçu pour tester la robustesse de n'importe quel logiciel exécutif, sa cible privilégiée est constituée par les systèmes d'exploitation.

Pour caractériser la robustesse du système cible, Ballista définit l'échelle *CRASH* composée de cinq modes de défaillance :

- Catastrophique : une défaillance majeure du système qui nécessite un redémarrage,
- Redémarrage : une tâche est bloquée et nécessite une relance,
- Arrêt d'exécution : interruption ou arrêt anormal de la tâche,
- Silencieux : aucun code d'erreur n'est retourné,
- Hindering : le code d'erreur retourné est incorrect.

Les trois premiers modes de défaillance sont observés automatiquement tandis que les deux derniers nécessitent un traitement manuel.

Pour chaque cas de test, un appel système est appelé une seule fois avec un ensemble de paramètres bien particulier. À chaque paramètre est associé un nombre de valeurs prédéfinies. Ces valeurs sont issues d'une base de données valides et exceptionnelles associées à chaque type de donnée de chaque argument de l'appel système. La détermination des valeurs valides et invalides ne dépend pas de l'aspect fonctionnel de l'appel système mais plutôt du type du paramètre. L'avantage majeur de cette approche est la génération automatique du code source des cas de test avant exécution.

Dans un premier temps, cette approche de test de robustesse a été appliquée aux systèmes d'exploitation supportant l'interface POSIX. Les résultats obtenus en utilisant BALLISTA

pour 15 systèmes d'exploitation POSIX ont montré que le mode de défaillance *Catastrophique* est présent dans six systèmes d'exploitation. Le *redémarrage* d'application n'a été observé que dans deux cas. Le mode de défaillance *Arrêt d'exécution* quant à lui a été révélé par tous les systèmes d'exploitation. Par ailleurs, le mode de défaillance *Silencieux* ne garantit pas forcément un comportement sûr de fonctionnement. Il convient de noter qu'une défaillance de type *Arrêt d'exécution* est un comportement attendu de la part du système après l'introduction de fautes. En revanche, les défaillances *Catastrophique* et *Redémarrage* reflètent des comportements anormaux et graves.

Par la suite, l'approche BALLISTA a été portée aux systèmes de la famille Windows (2000, NT, 95, 98, 98 SE et CE) ciblant ainsi l'interface standard WIN32 de Microsoft pour permettre la comparaison des familles Windows et Linux [Shelton 2000].

À l'origine, BALLISTA était basé sur une approche de test en boîte noire. Cependant, les résultats présentés dans [Shelton 2000] ont été détaillés suivant les principales fonctionnalités du système. Globalement, les résultats sont similaires pour Linux et les dernières versions de Windows (NT, 2000). Cependant, une analyse fine montre que Linux possède une mise en œuvre des appels système plus robuste que celle de Windows, mais que Linux est plus sensible au mode Arrêt d'exécution dans le cas des fonctions de la bibliothèque C.

Le travail le plus récent relatif à BALLISTA concerne son application à l'étude de la robustesse de plusieurs ORB CORBA [Pan 2001]. Ces travaux montrent la portabilité de l'outil et sa facilité d'utilisation à différents niveaux du système.

1.5.2.3 DBench

Les travaux de DBench sur les systèmes d'exploitation concernent deux aspects complémentaires : 1) la représentativité des fautes, et 2) l'étalonnage de la sûreté de fonctionnement de systèmes d'exploitation à base de micro-noyaux et à usage général vis-à-vis de pilotes de périphériques ou de l'application.

Représentativité des fautes

Pour étudier la représentativité des fautes dans les systèmes d'exploitation, les travaux présentés dans [Jarboui 2002a] ont visé à analyser le degré de similitude des comportements erronés des noyaux, suite à l'injection des différents modèles de fautes. Ces études ont ciblé le noyau Linux 2.4.0. Un modèle de fautes est défini par rapport au type de faute et au lieu d'injection. Les types de fautes peuvent être soit des bit-flips (comme dans MAFALDA), soit des corruptions sélectives des paramètres (comme dans BALLISTA). Quant au lieu, l'injection peut avoir lieu au niveau des appels système, c'est-à-dire au niveau de l'API, ou sur les paramètres des fonctions internes du noyau.

Les fautes au niveau de l'API simulent les fautes des applications et sont utilisées pour tester la robustesse du noyau en présence d'applications erronées. Les substitutions sélective et systématique des paramètres ont été utilisées pour corrompre les paramètres des appels système. Des activités modulaires ont été développées pour activer le système à

caractériser, et en particulier, deux modules fonctionnels de base : l'ordonnancement et la gestion de mémoire.

La comparaison des deux modèles de substitution est accomplie à partir de la comparaison de leurs effets respectifs sur le noyau. Le niveau d'observation le plus élevé est l'observation des défaillances. Si les défaillances observées après l'injection de deux modèles de fautes sont différentes alors elles ne sont pas considérées comme équivalentes. Cependant, si les défaillances observées sont similaires, alors un raffinement de ces observations peut être utile pour analyser plus finement le comportement.

Les auteurs ont groupé les défaillances des systèmes d'exploitation en deux classes : « défaillances rapportées » et « défaillances non rapportées ». Les défaillances rapportées englobent les cas où une exception est levée ou un code d'erreur est retourné. Dans le cas des défaillances non rapportées, on distingue le gel de l'application ou du noyau. Si aucun de ces cas n'a été observé, le mode de défaillance est appelé « non-signalement ».

Les résultats obtenus montrent que les deux types de fautes provoquent les mêmes modes de défaillance en nature et en quantité. En effet, le mode dominant de défaillance était le code d'erreur retourné (57% pour la substitution sélective et 45% pour les bit-flips). Les modes de défaillance non rapportée étaient pratiquement équivalents. La substitution sélective a provoqué moins de cas de non-signalement (28% des cas pour la substitution sélective contre 42% pour les bit-flips).

Ces résultats ont été raffinés en considérant d'autres critères comme la nature du code d'erreur retourné, la propagation des erreurs… Les résultats obtenus montrent que les bit-flips sont meilleurs en termes de la facilité d'application et de la diversification des codes d'erreur retournés tandis que la substitution sélective est meilleure en termes de durée d'expérimentation et de propagation d'erreurs.

En plus de l'injection des fautes au niveau de l'interface noyau-application, [Jarboui 2002a] présente les résultats de la substitution systématique des valeurs des paramètres des fonctions internes du noyau *Linux*. Les résultats obtenus montrent qu'un code d'erreur est retourné dans seulement 2% des expériences, alors que dans 28% des cas, une exception est levée, ce qui veut dire que 30% des fautes injectées ont donné lieu à des détections des erreurs. 32% des expériences ont provoqué des gels de l'application ou du noyau.

Etalonnage vis-à-vis des pilotes de périphériques

De part leur proximité du noyau et leurs origines diversifiées, les pilotes de périphérique ont un impact significatif sur la sûreté de fonctionnement du système global. Les travaux de [Albinet 2004] présente une méthodologie pour évaluer la robustesse des interfaces des systèmes d'exploitation vis-à-vis de pilotes défaillants. Ils proposent de simuler des erreurs résiduelles dans le code correspondant aux pilotes par injection de fautes par logiciel pendant l'exécution. Les observations consistent à relever les modes de défaillance du système d'exploitation en présence de ces erreurs. Pour réaliser les expériences, une interface de programmation pilote-noyau *DPI* (*Device Programming Interface*) a été détaillée. Elle regroupe les fonctions qu'utilisent les pilotes pour communiquer avec le noyau et pour effectuer des opérations d'entrée-sortie. Lors du traitement d'une tâche par

un pilote, une faute de type paramètre invalide comme celle définie dans BALLISTA est injectée dans les paramètres d'appel à une fonction du noyau.

L'étude expérimentale a porté sur trois pilotes de périphériques (*SMC-ultra* et *Ne2000* comme pilotes de réseau connus pour leur grande taille de code source et *SoundBlaster*, un pilote de son, comme pilote de taille moyenne) exécutés sur un noyau Linux. Différentes combinaisons de pilotes et de versions du noyau ont été considérées. Pour analyser les résultats expérimentaux, ils proposent un cadre global pour interpréter les résultats. Pour cela, ils caractérisent le noyau selon trois attributs de sûreté de fonctionnement : disponibilité (minimiser les cas du gel de noyau), sécurité-innocuité (minimiser les cas de service incorrect) et le *retour des expériences* (maximiser la notification des erreurs). Les résultats obtenus montrent que *SoundBlaster* a le meilleur comportement vis-à-vis du nombre d'erreurs notifiées, même si les résultats concernant la sécurité-innocuité et la disponibilité sont plus médiocres. Les résultats obtenus avec le test du pilote *SMC* sur deux versions différentes du noyau (2.2 et 2.4 respectivement) montrent le progrès réalisé en termes de notification d'erreurs, même si aucune amélioration n'a été constatée sur les deux autres attributs. Enfin, en comparant les effets de fautes provenant des deux pilotes de réseau *SMC* et NE2000, ce dernier est un peu meilleur en termes de disponibilité alors que SMC est meilleur en termes de nombre d'erreurs notifiées et de sécurité-innocuité.

Etalonnage des micro-noyaux pour systèmes enfouis

Dans le domaine des systèmes informatiques temps-réel, l'attention se porte non seulement sur les résultats générés par ces systèmes, mais également sur la production de ces résultats dans les contraintes temporelles imposées lors de la spécification de ces systèmes. À *Critical Software* [Moreira 2004], des travaux ont été menés pour étalonner la sûreté de fonctionnement des noyaux temps réel des systèmes spatiaux. L'objectif est de permettre aux développeurs/intégrateurs de ces systèmes d'évaluer et de comparer le déterminisme du temps de réponse des appels système de ces noyaux. Les observations collectées sont combinées dans une seule mesure finale appelée la « prédictibilité du temps de réponse ».

1.5.2.4 FINE

FINE, « Fault INjection and monitoring Environment » développé à l'université d'Illinois [Kao 1993], est un outil d'injection de fautes dont la cible est le noyau Unix. Son objectif est d'étudier les canaux de propagation d'erreurs dans le noyau Unix et d'évaluer l'impact de divers types de fautes sur son comportement. DEFINE (DistributEd Fault INjection and monitoring Environment) [Kao 1995] est un outil d'injection de fautes dans un environnement réparti basé sur FINE. Il reprend les fautes et les techniques d'injection de FINE localement sur chaque site du système distribué. Les fautes injectées, logicielles ou matérielles, sont activées par des activités synthétiques. Les logiciels de type applicatif n'ayant pas accès aux ressources internes du système d'exploitation, la partie de FINE responsable de l'injection de fautes a été implantée directement dans le noyau.

Les fautes logicielles considérées dans ces outils sont les fautes d'initialisation, d'affectation, de branchement ou affectant une fonction. Quant aux fautes matérielles, elles

sont classées par la localisation de manifestation : la mémoire, le bus, le processeur et les entrées/sorties.

Les résultats obtenus montrent que les fautes affectant la mémoire et les fautes logicielles ont généralement une latence de détection élevée tandis que les fautes affectant le bus et le processeur causent un arrêt immédiat du système. À peu près 90% des fautes détectées ont été détectées par le matériel. En ce qui concerne la propagation des erreurs, les expériences ont révélé que seules 8% des fautes injectées dans un composant fonctionnel ont été propagées vers d'autres composants. En ce qui concerne la propagation des erreurs entre les machines, les expériences montrent que la propagation des fautes du serveur vers les clients est plus importante que dans le sens contraire.

1.6 Conclusion

Dans ce chapitre, nous avons présenté les états de l'art concernant les deux domaines principaux sur lesquelles reposent nos travaux: l'évaluation de la sûreté de fonctionnement informatique et l'étalonnage de performance. L'objectif majeur est d'identifier des pratiques et des techniques qui nous guident vers la définition d'un cadre conceptuel d'étalonnage de la sûreté de fonctionnement des systèmes informatiques.

De nos jours, deux méthodes peuvent être identifiées pour évaluer la sûreté de fonctionnement ; il s'agit des méthodes analytiques et expérimentales. La méthode expérimentale semble mieux adaptée pour l'évaluation de la sûreté de fonctionnement des systèmes COTS. En particulier, la technique d'injection de fautes par logiciel constitue un choix attrayant parce que sa mise en œuvre est aisée et elle donne de bons résultats. En plus, elle permet de simuler des fautes matérielles et des fautes logicielles.

Nous avons également présenté dans ce chapitre les grands principes d'étalonnage de performance des systèmes d'exploitation, ainsi que des exemples d'étalons de performance de systèmes d'exploitation. Une caractéristique commune à tous ces étalons est de fournir une ou un ensemble réduit de mesures de performance. Cependant, un nombre important de ces mesures permet de mieux caractériser la performance des systèmes, même si les utilisateurs de ces résultats ne sont pas des experts. Les étalons de performance présentés dans ce chapitre peuvent servir dans le contexte d'étalonnage de sûreté de fonctionnement pour analyser, par exemple, la performance du système en présence de fautes, ce qui permet d'enrichir l'ensemble de mesures de sûreté de fonctionnement. De plus, ils constituent une source d'activités pour les étalons de sûreté de fonctionnement.

Les leçons retenues de ce chapitre nous permettent de définir, dans le chapitre suivant, un cadre conceptuel d'étalonnage de sûreté de fonctionnement des systèmes informatiques, ainsi que les principales propriétés qui doivent être respectées lors de la spécification de ces étalons.

Chapitre 2 Étalonnage de la sûreté de fonctionnement

2.1 Introduction

Bien qu'il existe plusieurs techniques d'évaluation et de validation de la sûreté de fonctionnement, ciblant différents niveaux d'un système informatique, il n'existe aucune approche globale et standard permettant la comparaison de systèmes hormis l'étalonnage. Dans la littérature, il existe peu de travaux relatifs à l'étalonnage de la sûreté de fonctionnement. L'objectif de ce chapitre est de présenter l'état de l'art de l'étalonnage de la sûreté de fonctionnement des systèmes informatiques.

Tout d'abord, nous passons en revue quelques étalons de sûreté de fonctionnement existants. Dans un deuxième temps, nous présentons un cadre conceptuel que nous avons élaboré dans le cadre du projet DBench. Ce cadre permet de spécifier clairement un étalon dont les résultats sont interprétables sans ambiguïté. Il identifie les principales dimensions nécessaires à la spécification d'un étalon de sûreté de fonctionnement. Ces dimensions décrivent 1) le système cible et le contexte d'étalonnage, 2) les mesures souhaitées de l'étalon, et enfin, 3) les expérimentations à réaliser.

La différence entre un étalon de sûreté de fonctionnement et les autres techniques d'évaluation de la sûreté de fonctionnement réside dans le fait que l'étalon doit être le fruit d'un consensus établi entre les différentes communautés d'industriels, d'universitaires et d'utilisateurs. En effet, un tel étalon doit satisfaire un certain nombre de propriétés pour être accepté par ceux qui souhaitent caractériser leur système. Par exemple, un étalon doit être portable, représentatif, etc. Dans la dernière section, nous définissons l'ensemble de propriétés que nous jugeons nécessaires pour qu'un étalon de sûreté de fonctionnement soit acceptable à la fois par les concepteurs et les utilisateurs de systèmes.

2.2 Etalons de sûreté de fonctionnement existants

Plusieurs étalons de sûreté de fonctionnement ont été développés pour permettre d'effectuer une analyse comparative de différents systèmes informatiques existants. Nous passons en revue dans ce qui suit les principaux étalons de sûreté de fonctionnement.

Parmi les premiers travaux effectués sur la comparaison de la sûreté de fonctionnement des systèmes informatiques, nous citons l'outil *FUZZ* [Miller 1990]. C'est un outil automatisé

conçu pour tester la robustesse des utilitaires des systèmes d'exploitation *Unix*. Il permet de générer des chaînes de caractères aléatoires qui sont fournies comme des commandes aux utilitaires à tester. Les expériences menées sur à peu près 90 utilitaires de chacune des 7 versions de *Unix* ont provoqué plus de 24 % d'arrêts inopinés. L'outil a été utilisé ensuite pour tester des applications avec une interface graphique (GUI) sur des systèmes *Unix* [Miller 1995] puis les utilitaires de *Windows NT* [Forrester 2000].

Crashme est un autre programme très simple qui permet de tester la robustesse des systèmes *Unix*. Puis, il s'est étendu pour tester d'autres systèmes tels que *Windows* et *FreeBSD*. Ce programme alloue une zone de mémoire remplie de données aléatoires. Ensuite, le programme crée des processus qui essayent d'exécuter ce tableau de données comme s'il était le code d'un programme.

En 1993, [Siewiorek 1993] présente des étalons de robustesse modulaires qui permettent de tester la robustesse des quatre composants fonctionnels du système *Unix* : le système de gestion des fichiers, la gestion de mémoire, les applications et les fonctions de la librairie C. Pour comparer différents systèmes (matériel + OS), une mesure unique *RBM* (Robustness Benchmark Measure) a été calculée à partir des mesures fournies par les quatre étalons.

[Tsai 1996] compare les mécanismes de tolérance aux fautes existant dans les systèmes commerciaux tolérants aux fautes Tandem. L'étalon est basé sur l'outil FTAPE [Tsai 1995] qui permet d'une part de générer une activité synthétique pour activer, selon les spécifications de l'utilisateur, le processeur, la mémoire ou les entrées/sorties, et d'autre part l'injection de fautes dans le processeur, la mémoire ou les entrées/sorties selon des stratégies spécifiques d'injection. L'injection de fautes peut être réalisée selon deux critères : 1) injection sur le composant système le plus stressé, et 2) injection sur le composant système le plus longuement stressé. La technique idéale consiste à combiner les deux. Deux mesures sont définies par l'étalon : 1) le nombre d'incidents catastrophiques, et 2) la dégradation de la performance. La première mesure représente la capacité de recouvrement du système, alors que la dégradation de la performance donne une indication sur la performance du système en présence de fautes.

[Mukherjee 1997] a examiné les efforts dédiés au développement des étalons cités ci-dessus et les a évalués par rapport à certains critères qu'il considère indispensables pour un étalon de sûreté de fonctionnement comme la portabilité, l'extensibilité… Il propose une approche hiérarchique pour mettre en place des étalons de robustesse. Cette approche a été utilisée pour étalonner la robustesse de deux composants fonctionnels du système Unix : les systèmes de fichiers et de mémoire.

[Brown 2000] propose une méthodologie générale pour étalonner la disponibilité des systèmes informatiques. Cette méthodologie utilise l'injection de fautes pour provoquer des situations où la disponibilité peut être compromise. Ainsi, les étalons de disponibilité correspondants doivent mesurer l'impact des fautes logicielles et matérielles sur la performance et l'adéquation du service fourni par un système soumis à une activité réaliste. La méthodologie introduite propose des étalons constitués de quatre parties : 1) un ensemble de métriques qui reflètent la dégradation de la disponibilité du système, 2) un générateur qui produit une activité réaliste et qui fournit le moyen de mesurer la qualité du

service, 3) un environnement d'injection de fautes et 4) une méthode graphique pour rapporter les mesures de disponibilité du système. Ces étalons ont servi à comparer la disponibilité des logiciels *RAID* sur les systèmes serveur Solaris 7, Linux et serveur Windows 2000 et récemment, ils ont été utilisés pour évaluer la disponibilité du système de gestion de base de données Microsoft SQL Server 2000 [Brown 2002].

Récemment, des travaux réalisés au sein de *Sun MicroSystems* ont porté sur l'étalonnage de la sûreté de fonctionnement. [Zhu 2002] définit un étalon de la sûreté de fonctionnement dédié à la disponibilité. Cette disponibilité des systèmes informatiques est déterminée au travers de la caractérisation expérimentale des principaux attributs suivants :

- Taux défaillance/maintenance : le nombre de défaillances et d'événements de maintenance pendant une période de temps donnée ;
- Robustesse : la capacité du système à détecter et à gérer les événements extérieurs au système et à rester disponible face à ces événements ;
- Recouvrement : la vitesse à laquelle un système redevient opérationnel après une défaillance.

Basés sur ce cadre conceptuel, deux étalons spécifiques de sûreté de fonctionnement ont été développés : un étalon de robustesse pour les événements de maintenance du matériel [Zhu 2003] et un étalon de recouvrement des systèmes [Mauro 2004].

Les événements de maintenance des systèmes informatiques, comme la mise à jour d'un logiciel ou le remplacement d'un composant matériel, peuvent entraîner des défaillances de ces systèmes. [Zhu 2003] propose un étalon pour mesurer un aspect spécifique de robustesse des systèmes informatiques concernant la manipulation des événements de maintenance des composants matériels. L'approche adoptée se résume en quatre étapes : 1) la définition de différentes classes d'événements de maintenance, selon la disponibilité du système pendant l'activité de maintenance, 2) la distribution des composants matériels du système sur ces différentes classes, 3) le calcul des pourcentages des événements de maintenance de chacune des classes (ces pourcentages sont déterminés comme étant les rapports entre la somme des taux de défaillance des composants matériels de chaque classe et la somme des taux de défaillance de tous les composants matériels du système), et enfin 4) le calcul de la mesure *MRB-A* (*Maintenance Robustness Benchmark-fault induced HW maintenance events*) correspondant à chacune des classes définies dans l'étape 1. Cette mesure dépend du pourcentage calculé dans la troisième étape. Le meilleur scénario de robustesse correspond à un système dont tous les composants matériels peuvent être remplacés sans avoir besoin de l'éteindre.

Motivés par le fait que le temps de recouvrement d'un système est un élément important pour la disponibilité, les auteurs de [Mauro 2004] définissent le SRB (System Recovery Benchmark), un étalon pour le recouvrement des systèmes. D'abord, ils définissent un cadre conceptuel générique qui permet de définir un étalon pour le recouvrement des systèmes ; dans ce contexte, le recouvrement est défini comment étant le temps nécessaire au système pour reprendre son service normal après l'occurrence d'une faute. Ensuite, ils définissent le SRB-A, une variante de ces étalons destinée à caractériser le recouvrement

des systèmes après une défaillance matérielle ou une défaillance du système d'exploitation. La mesure consiste à calculer le temps entre l'occurrence de la faute fatale et la reprise de service par le système après son redémarrage. Ainsi, les mesures obtenues permettent de comparer la dégradation de performance entre les différents systèmes ciblés.

À IBM, les chercheurs ont développé des étalons permettant de quantifier la capacité des systèmes à être autonomes. Ces étalons s'intéressent aux quatre principaux domaines de l'auto-gestion définis par IBM : l'auto-configuration, l'auto-réparation, l'auto-optimisation et l'auto-protection [Lightstone 2003].

Les auteurs de [Vieira 2003b] présentent un nouvel étalon de sûreté de fonctionnement dédié aux systèmes transactionnels en ligne. Cet étalon adopte l'activité utilisée par l'étalon de performance TPC-C pour activer le système à étalonner, et auquel ils ajoutent deux nouveaux éléments : l'ensemble de fautes à injecter et les mesures relatives à la sûreté de fonctionnement. Ainsi, cet étalon est fourni sous formes d'une spécification qui étend celle de l'étalon TPC-C, dans le but d'évaluer la performance et les mesures de sûreté de fonctionnement des systèmes transactionnels en ligne. L'ensemble considéré de fautes est basé sur trois classes de fautes : les fautes des opérateurs, les fautes logicielles et les fautes matérielles. Quant à l'ensemble des mesures fournies, nous pouvons distinguer : 1) les *mesures de performance de base* comme le nombre de transactions exécutées par minute, et le prix par transaction, 2) les *mesures de performance en présence de fautes*, et 3) les *mesures de sûreté de fonctionnement* telles que le nombre d'erreurs détectées, la disponibilité du point de vue du système à étalonner et la disponibilité du point de vue des utilisateurs finaux. [Durães 2004] propose le WEB-DB, un étalon de sûreté de fonctionnement pour les serveurs Web. Cet étalon utilise la même plate-forme expérimentale, activité et ensemble de mesures de performance de l'étalon de performance SPECWeb99 [SPECweb99]. Les développeurs de cet étalon ont ajouté des mesures de performance en présence de fautes et des mesures de sûreté de fonctionnement à l'ensemble initial de mesures. Cet étalon a été utilisé pour étalonner la sûreté de fonctionnement des serveurs Web Apache et Abyss.

[Ruiz-Garcia 2004] propose un étalon de sûreté de fonctionnement pour les logiciels de contrôle dans l'automobile. Ces logiciels opèrent dans des systèmes électroniques complexes appelés Unités de Contrôle Electroniques (UCE). Pour cela, les auteurs proposent un ensemble de mesures qui intéressent les intégrateurs de ces UCE. Ces mesures concernent les nombres et les types de défaillances de ces logiciels de contrôle. Les modes de défaillance sont groupés en trois classes : 1) les défaillances entraînant le redémarrage de l'UCE, 2) les défaillances affectant les valeurs de sortie des UCE, et 3) les défaillances temporelles se produisant lorsque les UCE ne respectent pas les échéances imposées. Une activité composée d'un simulateur d'un accélérateur et des données correspondant aux variables internes a été utilisée pour activer les UCE, et des fautes transitoires sont injectées dans son espace mémoire. L'étalon présenté dans cet article est ensuite utilisé pour étalonner la sûreté de fonctionnement de deux logiciels de contrôle des automobiles Diesel. Le premier logiciel communique directement avec la couche matérielle de l'UCE, tandis que le deuxième utilise les services fournis par un micro-noyau temps réel (µ*C/OS II*).

2.3 Cadre conceptuel pour l'étalonnage de la sûreté de fonctionnement

Dans le premier chapitre, nous avons présenté un état de l'art concernant l'étalonnage de performance des systèmes informatiques et les méthodes expérimentales d'injection de fautes utilisées pour caractériser la sûreté de fonctionnement des systèmes d'exploitation. Cet état de l'art nous a été très utile dans la mesure où il nous a servi de point de départ et d'appui pour la définition d'un cadre conceptuel permettant de définir un étalon de sûreté de fonctionnement.

L'étalon de sûreté de fonctionnement peut hériter des étalons de performance la spécification du système cible à étalonner ainsi que le contexte d'étalonnage, la spécification de quelques mesures de performance, la représentativité de la méthode d'activation du système cible et les propriétés à satisfaire. L'étalon de sûreté de fonctionnement peut hériter de la méthode d'injection de fautes les spécifications des composants de l'ensemble ***FARM*** nécessaires pour la réalisation les expériences.

Dans le cadre du projet DBench [Kanoun 2004], nous avons défini un étalon de sûreté de fonctionnement comme étant une spécification d'une procédure permettant d'évaluer la sûreté de fonctionnement d'un système ou d'un composant informatique en présence de fautes qui peuvent être induites par son environnement opérationnel. La méthode utilisée est une méthode expérimentale qui consiste à soumettre le composant cible à un certain nombre d'expériences dans lesquelles le comportement du composant est caractérisé par rapport à des requêtes erronées. Par conséquent, nous avons spécifié une séquence de trois groupements de dimensions à définir dans l'ordre, permettant ainsi de définir un étalon de sûreté de fonctionnement d'un système informatique. Ces groupements de dimensions sont illustrés dans la Figure 2-1 : 1) les dimensions de catégorisation qui permettent de décrire le système cible ainsi que le contexte d'étalonnage, 2) les mesures de sûreté de fonctionnement à évaluer, et enfin 3) les dimensions d'expérimentation qui incluent tous les éléments nécessaires à la mise en œuvre des expérimentations. Parmi ces éléments, nous citons le *système à étalonner*, le *profil d'exécution* composé de l'activité et de l'ensemble des fautes à injecter, et enfin, les observations nécessaires pour calculer les mesures souhaitées.

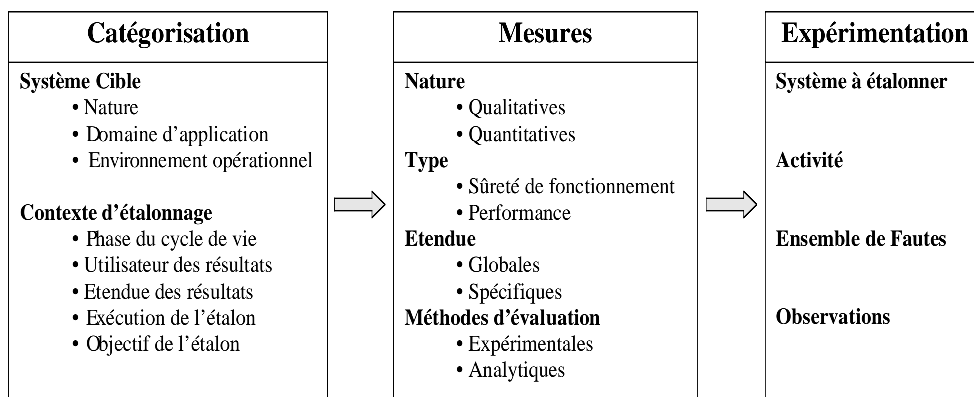


Figure 2-1 Trois groupements de dimensions

La dimension de *catégorisation* hérite essentiellement des principales caractéristiques des étalons de performance et les étend alors que les *mesures* sont spécifiées à partir des mesures de sûreté de fonctionnement, et des mesures de performance (en présence de fautes). Dans la dimension d'*expérimentation*, nous retrouvons l'ensemble des composants F, A et R (représentés respectivement par l'*ensemble de fautes*, l'*activité* et les *observations*) de l'ensemble $FARM$. Pratiquement, la dimension d'expérimentation définit tous les éléments nécessaire pour développer l'outil d'étalonnage à développer, et que nous appelons prototype par abus de langage.

2.3.1 Dimensions de catégorisation

Les dimensions de catégorisation influencent directement le choix des mesures requises de l'étalon ainsi que les dimensions d'expérimentation. Ces dimensions doivent définir clairement : 1) les frontières du système cible candidat et 2) le contexte d'étalonnage.

Le Système Cible (SC)

Le système cible est défini en fonction de 1) sa nature, 2) son domaine d'application et 3) son environnement opérationnel. Dans nos travaux, nous avons ciblé l'étalonnage de la sûreté de fonctionnement des systèmes d'exploitation à usage général.

En général, le système cible peut être décrit de deux manières différentes. Une description détaillée est appréciable parce qu'elle permet d'évaluer les différents mécanismes internes spécifiques à la sûreté de fonctionnement. Cependant, cette forme de description peut imposer des contraintes sur la portabilité de l'étalon et nécessite une connaissance approfondie de l'architecture du système cible. Le système cible peut aussi être décrit comme une architecture en couches qui interagissent entre elles, chacune constituant un niveau d'abstraction qui pourrait être porté sur d'autres systèmes. Par exemple, le système d'exploitation peut être vu comme une couche logicielle qui assure le lien entre la couche applicative et la couche matérielle. Elle fournit aux applications un certain nombre de services au travers d'une interface bien particulière (API). Cette façon de voir le système cible permet d'améliorer la portabilité de l'étalon sur différents systèmes. Il est important de noter que le système cible doit être défini sans aucune ambiguïté, et que ses limites doivent être bien définies.

La définition du domaine d'application constitue un aspect clef dans le processus d'étalonnage de sûreté de fonctionnement, dans la mesure où il influence directement l'environnement opérationnel, l'ensemble des mesures d'étalonnage et le profil d'exécution composé de l'activité et de l'ensemble de fautes à injecter et qui sera utilisé dans la phase d'expérimentation. La division du large spectre des applications en différents domaines facilite la gestion des différentes applications existantes. Des domaines d'application différents nécessiteront des étalons de sûreté de fonctionnement différents. Par exemple, les systèmes transactionnels n'ont pas les mêmes besoins que les systèmes de contrôle-commande.

L'environnement opérationnel caractérise l'environnement dans lequel le système est utilisé. Cet environnement regroupe plusieurs aspects clefs pour l'étalonnage, comme l'usage fonctionnel et quotidien du système cible dans sa vie opérationnelle et l'ensemble

des requêtes erronées d'origine extérieure, et auxquelles le système cible doit faire face. En conséquence, l'environnement opérationnel peut influencer le choix du profil d'exécution. Ce profil d'exécution doit être représentatif de l'ensemble des événements qui peuvent survenir lors de la vie opérationnelle du système cible. Une définition bien détaillée du domaine d'application aide à définir un environnement opérationnel typique qui sera utilisé dans la phase d'expérimentation.

Le Contexte d'étalonnage

L'étalonnage de sûreté de fonctionnement peut se faire avec des perspectives multiples. Le contexte d'étalonnage définit :

La phase du cycle de vie - l'étalonnage de sûreté de fonctionnement peut viser une phase particulière du cycle de vie du système cible (conception, vie opérationnelle, …). Les résultats de l'étalonnage sont ainsi utilisés pour caractériser la phase concernée du cycle de vie et peuvent éventuellement servir pour la caractérisation du système dans les phases suivantes.

L'exécuteur de l'étalon et l'utilisateur des résultats – l'exécuteur de l'étalon pourra être n'importe quelle entité ou personne qui applique l'étalon (le développeur du système, l'intégrateur, des organismes spécialisés…). Les résultats de l'étalonnage peuvent servir soit pour l'entité qui exécute l'étalon soit pour d'autres entités. Par exemple, l'intégrateur d'un système peut exécuter un étalon à des fins internes pour bien concevoir le système. Il peut également mettre les résultats à disposition des futurs acquéreurs du système.

L'étendue des résultats - comme pour les étalons de performance, les étalons de sûreté de fonctionnement peuvent fournir des mesures caractérisant le service du système global, ou bien des mesures caractérisant des mécanismes de sûreté de fonctionnement internes au système cible, qui servent à l'ajuster et l'améliorer. Il s'agit respectivement de macro-étalons et micro-étalons de sûreté de fonctionnement.

L'objectif de l'étalon - l'objectif final d'un étalon de sûreté de fonctionnement est de comparer des systèmes ou composants informatiques selon un ensemble des critères de sûreté de fonctionnement. Cependant, il pourrait être utilisé pour accomplir d'autres missions. Par exemple, pendant la phase de conception, un étalon de sûreté de fonctionnement pourrait être utilisé pour identifier les points faibles d'un prototype, et ainsi, aider le concepteur à prendre la bonne décision concernant le choix d'une plate-forme matérielle bien adaptée ou d'un logiciel COTS spécifique. Enfin, dans la phase opérationnelle du cycle de vie du système cible, les résultats de l'étalon peuvent être utilisés pour mesurer l'impact des fautes sur la sûreté de fonctionnement du système, et ainsi, aider l'utilisateur des résultats à régler son système pour un meilleur compromis entre la performance et la sûreté de fonctionnement.

2.3.2 Dimension de mesures

L'un des principaux objectifs des étalons de sûreté de fonctionnement est de comparer la sûreté de fonctionnement des différents systèmes informatiques candidats. Pour effectuer la meilleure comparaison, il faut s'appuyer sur une ou plusieurs mesures significatives et bien

adaptées du point de vue de l'utilisateur des résultats. Ces mesures doivent être clairement spécifiées et détaillées.

Un nombre élevé de mesures de l'étalonnage, qu'elles soient de performance ou de sûreté de fonctionnement, permet d'évaluer les différentes caractéristiques ciblées par l'étalon. Cependant, le nombre des résultats fournis par les étalons de performance existants est très petit, et dans certains cas, il est réduit à une seule mesure. La principale cause réside dans le fait que les utilisateurs des résultats de l'étalonnage ne sont pas forcément des experts. En plus, un petit nombre de mesures permet de réduire au maximum les ambiguïtés qui peuvent surgir, dans le cas où une mesure favorise un système candidat et une deuxième le défavorise.

La sûreté de fonctionnement couvre un large spectre de mesures : disponibilité, fiabilité, sécurité-innocuité, etc. (cf. section 1.2.1). Dans le domaine d'étalonnage de sûreté de fonctionnement, l'ensemble des mesures dépend largement des dimensions de catégorisation. Pour identifier les mesures de sûreté de fonctionnement d'un système, son comportement peut être perçu, par son ou ses utilisateurs, comme une alternance entre deux états de service par rapport à l'accomplissement de la fonction du système : service correct et service incorrect (cf. section 1.2.2). Des exemples des mesures de sûreté de fonctionnement peuvent être trouvés dans [Zhu 2002] et [Lightstone 2003]. D'autre part, l'occurrence ou l'injection des fautes n'aboutit pas forcément à une défaillance du système. Par contre, ces fautes peuvent dégrader la performance du système cible, ce qui implique un lien étroit entre la sûreté de fonctionnement et la performance. Dans ce cas, l'évaluation des mesures de performance en présence des fautes permet d'étendre, et parfois de compléter, la caractérisation des systèmes informatiques du point de vue sûreté de fonctionnement [Vieira 2002].

Comme mentionné dans le paragraphe précédent, le contexte d'étalonnage a un grand impact sur les types des mesures requises de l'étalon de sûreté de fonctionnement. Par exemple, l'utilisateur final d'un système informatique est généralement intéressé par des mesures globales qui caractérisent le service fourni par le système, tandis que le développeur ou l'intégrateur pourrait être plus intéressé par des mesures plus spécifiques des mécanismes ciblés du système.

Les mesures globales caractérisent le système par rapport au service fourni, en considérant tous les événements qui influencent son comportement. Les mesures spécifiques caractérisent des mécanismes particuliers du système, sans prendre en compte tous les processus qui influencent son comportement. Parmi les mesures spécifiques aux mécanismes de détection des erreurs, on peut citer l'efficacité des mécanismes de détection des erreurs (le pourcentage des erreurs détectées) et la latence de détection (le temps nécessaire pour détecter l'erreur). Ces mesures peuvent encore être plus détaillées en distinguant les fautes en fautes matérielles et fautes logicielles. Ces mesures sont généralement obtenues par expérimentation. La variété et le grand nombre des mesures spécifiques montrent la complexité de la caractérisation de la sûreté de fonctionnement. Certaines de ces mesures peuvent servir comme paramètres d'un modèle global qui permet de caractériser des mesures globales.

Il est à noter que la notion du système est récursive dans la mesure où un système peut être le composant d'un autre système. Ceci implique que le concept des mesures globales et spécifiques puisse être appliqué à la fois pour les systèmes de systèmes ainsi que pour ses composants.

L'ensemble des mesures fournies par un étalon de sûreté de fonctionnement est évalué par des méthodes expérimentales ou analytiques. Certaines mesures globales telles que la fiabilité ou la disponibilité d'un système ne peuvent être obtenues uniquement par expérimentation sur le système. Elles nécessitent généralement l'établissement d'un modèle analytique dont certains paramètres peuvent être obtenus par expérimentation. Dans ce contexte, nous citons l'exemple d'étalonnage de la sûreté de fonctionnement des systèmes transactionnels en ligne [Kanoun 2004] où des modèles ont été établis pour évaluer la disponibilité et le coût total des défaillances de ces systèmes. Ces étalons sont basés sur des modèles analytiques qui utilisent des informations fournies par les expérimentations effectuées sur le système cible (les pourcentages des différents modes de défaillance) et des informations obtenues par des sources externes (taux d'occurrence, taux de réparation et coût de réparation de chaque mode de défaillance).

2.3.3 Dimensions d'expérimentation

Après avoir défini les dimensions de catégorisation et de mesures, ce paragraphe traite les différentes étapes nécessaires à la conduite des expériences d'étalonnage de sûreté de fonctionnement. Ces étapes sont illustrées dans la Figure 2-2. Sur cette figure, nous distinguons l'étape d'expérimentation et l'étape de traitement des observations fournies par l'expérimentation. Les entités nécessaires à la conduite de la première étape sont le système à étalonner, le profil d'exécution composé de l'activité et de l'ensemble de fautes, et les observations à effectuer pour évaluer les mesures souhaitées. L'étape de traitement sert à analyser les données brutes fournies par l'étape d'expérimentation afin de fournir les mesures de l'étalon.

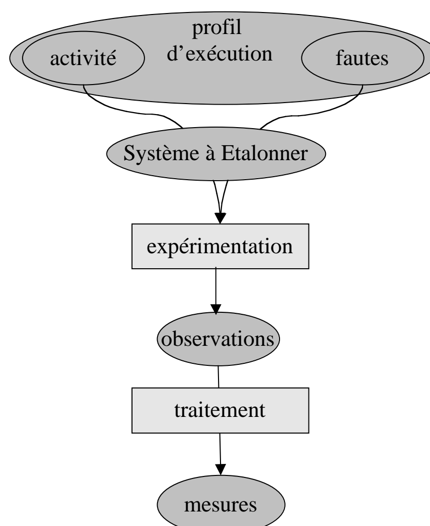


Figure 2-2 : Etapes de conduite de l'étalon

Nous définissons, dans ce qui suit, les différentes entités nécessaires pour la conduite d'un étalon de sûreté de fonctionnement des systèmes informatiques.

Système à étalonner

En pratique, un système cible ne peut pas être caractérisé tout seul, surtout quand il s'agit d'un système informatique logiciel comme les systèmes d'exploitation ou les systèmes de gestion de bases de données. Il faut l'inclure dans un système complet que nous appelons le *système à étalonner*. Le *système à étalonner* fournit au *système cible* le support matériel et logiciel pour que ce dernier puisse s'exécuter.

Pour clarifier ces concepts, nous prenons comme exemple les systèmes d'exploitation. Le *système cible* correspond au noyau avec le minimum de pilotes de périphériques nécessaires pour exécuter l'activité de l'étalon. Ce système cible s'exécute au dessus d'une couche matérielle, utilise un ensemble de libraires nécessaires pour communiquer avec d'autres composants informatiques. Il a également besoin de pilotes de périphériques pour communiquer avec des périphériques extérieurs. L'ensemble du système cible, des libraires et du matériel forment le *système à étalonner*.

Pour clarifier la liaison entre les deux notions de *système cible* et de *système à étalonner*, nous présentons l'exemple de la Figure 2-3. Le *système à étalonner* contient la couche applicative qui interagit avec le système cible via l'API (Application Programming Interface) comme le Win32, POSIX… En comparant cette figure avec la Figure 1-1 du premier chapitre, nous constatons que les pilotes des périphériques sont considérés inclus dans le système cible. Le système cible s'exécute sur une plate-forme matérielle pour servir la couche applicative qui est au-dessus.

Il est très important que le *système à étalonner* soit bien documenté, et que ses limites soient bien définies. Ceci permet de mieux porter l'étalon sur d'autres systèmes et de bien reproduire et surtout interpréter ses résultats. En plus, l'interface entre le *système cible* et le *système à étalonner* doit être bien précise, parce que l'activité et l'ensemble des fautes auxquels le système cible est soumis ne doivent, en aucun cas, modifier le système cible.

En pratique, ce qui est étalonné est le *système à étalonner* parce qu'il est impossible de dissocier le système cible du reste des composants des systèmes à étalonner.

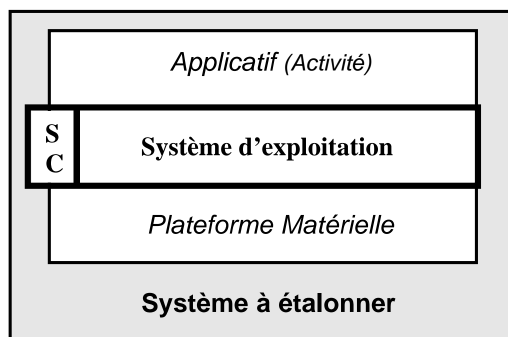


Figure 2-3 Système à étalonner

Activité

L'activité simule et représente le domaine d'application dans lequel le système cible sera utilisé. Elle permet d'activer le système cible, et par conséquent, les fautes injectées dans le but d'évaluer sa sûreté de fonctionnement. Le système cible peut être activé en utilisant plusieurs approches parmi lesquelles nous distinguons les activités synthétiques et les activités réalistes.

L'activité synthétique est une approche modulaire qui, en utilisant des activités artificielles, permet d'activer des composants fonctionnels différents du système cible (l'ordonnancement, la gestion de la mémoire, etc.). Les résultats de l'étalonnage peuvent ainsi caractériser les différents composants du système, ou être regroupés pour le caractériser en tant qu'une seule entité.

Si le domaine d'application est connu, une *trace du domaine d'application*, ou tout simplement une *activité réaliste*, peut être utilisée pour activer le système cible. Elle permet d'activer le système cible de la même manière qu'une application réelle et avoir la même trace d'exécution. Cependant, cette famille d'activités pose un certain problème de représentativité des mesures, surtout celles qui caractérisent les modes de défaillance de l'activité comme son état ou le temps de son exécution après l'injection d'une faute.

Un travail important a été mené dans ce domaine avec les étalons de performance, où l'on peut trouver des activités qui couvrent un large spectre de domaines d'application. En plus de leur capacité de mesurer la performance du système, les étalons de performance fournissent des activités bien étudiées, variées et représentatives (cf. section 1.2).

Ensemble de fautes

L'ensemble de fautes doit simuler les menaces réelles que le système est susceptible d'affronter. L'ensemble de fautes inclut les notions de fautes internes et externes et tout ce qui peut être considéré comme conditions exceptionnelles, où les valeurs peuvent appartenir au domaine de validité des entrées du système, mais pour lesquelles le système cible n'est peut être pas protégé.

D'une part, dans le cas des composants logiciels en général, l'injection de fautes internes dans le système cible risque de modifier le système initial. D'autre part, [Koopman 2002] s'est interrogé sur la représentativité de la technique d'injection de fautes comme méthode pour étalonner la sûreté de fonctionnement des systèmes informatiques. Il a conclu que l'injection des fautes internes, contrairement à celle des fautes externes, ne peut pas être utilisée pour étalonner la sûreté de fonctionnement des systèmes informatiques pour deux raisons :

1) ces fautes ne sont pas représentatives des fautes que le système peut endurer pendant sa vie opérationnelle, parce qu'une injection d'une faute interne ne prend pas en compte si cette faute est activée durant l'exécution opérationnelle du système ;

2) l'optimisation de développement d'un système informatique implique la réduction du coût et du temps du développement. Par conséquent, les tests de validité des valeurs ont tendance à être utilisés une seule fois. Comme la plupart de tests de validité sont installés

au niveau des interfaces, la faute interne injectée peut causer une défaillance qui n'est pas représentative pour le système à étalonner. L'exemple est montré dans [Jarboui 2002a] où l'ensemble des codes d'erreur retournés était plus petit en injectant des inversions de bits à l'intérieur du noyau qu'au niveau de l'interface, parce que les mécanismes de tolérance aux fautes sont implémentés au niveau de l'*API* et ne peuvent pas avoir d'effet quand la faute est injectée à l'intérieur du noyau.

Dans le cadre du projet DBench et de ce mémoire, seules les fautes externes sont considérées. Les fautes externes dépendent entièrement de l'environnement opérationnel et incluent les conditions exceptionnelles. L'injection de fautes externes permet d'analyser les failles de robustesse du système et peut aboutir à l'élaboration de mécanismes de confinement d'erreurs pour améliorer la robustesse du système. Nous notons que l'ensemble de fautes originaires des composants du *système à étalonner* autre que le *système cible* sont considérées comme *externes* au *système cible*.

La définition d'un ensemble représentatif de fautes constitue l'une des étapes les plus difficiles dans le processus de spécification et de développement d'un étalon de sûreté de fonctionnement. Dans le but de gérer plus facilement le déroulement des expériences, il est conseillé de considérer une seule classe de fautes à la fois (matérielles, logicielles, provenant de l'utilisateur humain…). En même temps, l'étalonnage de sûreté de fonctionnement doit considérer un domaine d'application bien spécifique ce qui peut simplifier davantage la tâche de la définition de l'ensemble de fautes à injecter.

La majorité des fautes matérielles peuvent être modélisées au niveau informationnel par des fautes transitoires et particulièrement par des inversions de bits.

Les techniques d'injection de fautes par logiciel semblent être les plus pertinentes dans le cadre d'un étalon de sûreté de fonctionnement. En effet, ces techniques ont la capacité de simuler à la fois les fautes matérielles et les fautes logicielles [Jarboui 2002b] [Durães 2002].

Observations

Les observations relevées sur le système cible permettent de caractériser son comportement après l'exécution de l'activité en présence des fautes injectées. Elles sont directement dictées par l'ensemble des mesures désirées, définies dans le deuxième groupement de dimensions. À la fin des expériences, ces observations sont traitées pour évaluer les mesures de sûreté de fonctionnement.

Parmi les observations de base, nous citons l'identification des différentes issues ou états résultants de l'application du profil d'exécution au système cible. Pour un système d'exploitation, ces issues sont de type rapporté comme le retour d'un code d'erreur ou la levée d'une exception, ou bien de type non-rapporté comme le gel ou la panique du noyau. Ainsi on évalue le comportement du système cible vis-à-vis des requêtes erronées, c'est-à-dire sa robustesse.

D'autres observations importantes sont nécessaires pour l'évaluation des mesures temporelles désirées en présence de fautes. Dans ce cas, le plus important est de comparer

les performances du système cible en présence et en absence de fautes, plutôt que juste évaluer ces performances. Pour cela, il faut procéder à plusieurs expériences, en appliquant l'activité au système cible avec et sans fautes [Vieira 2003a].

Le *système à étalonner* fournit tous les éléments nécessaires au *système cible* pour qu'il puisse s'exécuter (matériel, logiciel, pilotes de périphériques, etc.). En pratique, la caractérisation du *système cible*, isolé du reste du *système à étalonner*, nécessite une intrusion au sein du *système cible*. Il est plus judicieux d'utiliser les observations relevées au niveau du *système à étalonner* plutôt qu'au niveau du *système cible* pour caractériser la sûreté de fonctionnement de ce dernier.

2.4 Propriétés d'un étalon de sûreté de fonctionnement

Dans ce qui suit, nous décrivons l'ensemble des propriétés que nous jugeons indispensables pour qu'un étalon de sûreté de fonctionnement soit accepté par le plus large spectre des utilisateurs : la répétitivité, la reproductibilité, la représentativité, la portabilité, la non-intrusivité, l'interférence, la mise à l'échelle, l'automatisation et le coût de l'étalonnage. Certaines propriétés doivent être prises en considération dès les premières phases de définition de l'étalon parce qu'elles ont un grand impact aussi bien sur les mesures à évaluer que sur les expérimentations à réaliser, et par conséquent elles sont vérifiées par construction. D'autres, comme la répétitivité ou le coût de l'étalonnage, ne peuvent être vérifiées que par expérimentation.

2.4.1 La répétitivité

Une des propriétés essentielles qu'un étalon de sûreté de fonctionnement doit satisfaire est la répétitivité : c'est la propriété qui permet de garantir des résultats statistiquement équivalents quand l'étalon est appliqué plusieurs fois dans le même environnement (même système à étalonner, même profil d'exécution – activité et ensemble de fautes –, même prototype, etc.). La répétitivité est une propriété centrale d'un étalon de sûreté de fonctionnement car sans elle, l'étalon n'a aucun sens d'exister et dans ce cas, il constitue une source de conflit plutôt que d'accord. Sans répétitivité, personne n'aura confiance dans les résultats obtenus par les expériences de l'étalonnage. Tous les étalons crédibles de sûreté de fonctionnement doivent être répétitifs.

2.4.2 La reproductibilité

La reproductibilité est la propriété qui garantit d'obtenir statistiquement des résultats équivalents quand l'étalon est développé à partir des mêmes spécifications et est utilisé pour caractériser le même système à étalonner.

La reproductibilité est fortement liée aux détails fournis dans les spécifications. En pratique, il faut envisager un certain compromis qui garantit que les spécifications soient assez générales pour être appliquées à une classe des systèmes variés, et en même temps, suffisamment spécifiques pour qu'elles soient appliquées à chacun de ces systèmes, sans avoir besoin de déformer les spécifications originales pendant le développement de l'étalon.

La différence entre la répétitivité et la reproductibilité est que la première intervient au niveau de l'expérimentation alors que la deuxième est assurée au niveau des spécifications. La répétitivité est beaucoup plus facile à vérifier que la reproductibilité : il suffit d'exécuter le prototype plusieurs fois sur le même système à étalonner et calculer la variation des résultats pour vérifier la répétitivité, alors que pour vérifier la reproductibilité, il faut implémenter des prototypes (à partir des spécifications) par plusieurs personnes et les exécuter sur le même système à étalonner pour tester l'équivalence des résultats.

2.4.3 La représentativité

La représentativité concerne toutes les dimensions de l'étalonnage. Plus particulièrement, l'ensemble des mesures, l'activité et l'ensemble de fautes doivent être les plus représentatifs possibles.

Les mesures sont directement liées au contexte d'étalonnage. Ainsi, ces mesures devraient tenir compte des besoins et des contraintes exprimés dans ce contexte (utilisateurs finaux, phase de cycle de vie, …).

Une activité bien représentative de l'activité opérationnelle du système cible est un élément essentiel à prendre en compte lors de la spécification d'un étalon de sûreté de fonctionnement. En effet, cette propriété doit être examinée dans le contexte du domaine d'application. Plusieurs études ont été effectuées pour évaluer la représentativité de l'activité, notamment dans les étalons de performance, où des efforts considérables ont été dépensés pour développer des activités qui simulent bien la réalité. Ces activités peuvent être reprises dans le domaine de l'étalonnage de sûreté de fonctionnement comme un point de départ, même s'il reste beaucoup de travaux et de recherche à accomplir dans ce domaine, surtout quand il s'agit de la combinaison de l'activité et de l'ensemble des fautes à appliquer au système pour produire le comportement erroné souhaité.

L'ensemble des fautes qui doivent être injectées, et vis-à-vis desquelles le système cible est testé, doit représenter les fautes réelles que le système peut subir pendant sa vie opérationnelle. Plus les fautes envisagées sont réalistes, plus les résultats de l'étalonnage sont significatifs. Nous avons montré dans le premier chapitre que la technique d'injection de fautes peut être appliquée à plusieurs endroits d'un système informatique pour évaluer sa sûreté de fonctionnement. En plus, plusieurs types de fautes peuvent être injectés. Dans ce cas, l'évaluateur de la sûreté de fonctionnement d'un système informatique peut s'interroger sur les modèles de fautes qu'il va injecter. La représentativité des fautes est un facteur clé dans le contexte d'étalonnage de sûreté de fonctionnement. Jusqu'à présent, aucun modèle de fautes n'a été adopté unanimement pour représenter les fautes logicielles. Parmi les modèles développés, nous citons *ODC* (Orthogonal Defect Classification) [Chillarege 1992] qui essaie de classifier les différents types de fautes qui ont été observées lors de la phase de conception et en exploitation de grands systèmes d'exploitation développés par IBM. Ce problème de représentativité de fautes constitue encore l'objectif de plusieurs travaux de recherche.

Les recherches établies jusqu'à présent ont ciblé la comparaison :1) de techniques d'injection de fautes par rapport aux fautes réelles [Madeira 2000] [Jarboui 2002b] [Jarboui

2003] et 2) de plusieurs techniques d'injection de fautes [Choi 1992] [Fuchs 1996] [Jarboui 2002a] [Jarboui 2002c]. Ces recherches ont montré des résultats assez partagés. Quelques techniques ont été identifiées comme étant équivalentes alors que d'autres comme étant complémentaires. La tendance actuelle est de favoriser l'injection de fautes par logiciel pour simuler les fautes physiques [Barton 1990] [Kanawati 1995] [Carreira 1998]. Cette approche facilite l'application de l'injection de fautes en surmontant beaucoup de problèmes causés par les techniques d'injection physique comme la contrôlabilité des expériences. De plus, des études récentes ont montré, en partie, que la technique d'injection de fautes par logiciel est capable de simuler des fautes logicielles [Madeira 2000]. C'est pourquoi la technique d'injection de fautes par logiciel semble être la technique privilégiée pour la génération de l'ensemble de fautes dans le cadre des étalons de sûreté de fonctionnement.

Les travaux menés dans le cadre du projet DBench portant sur la représentativité des fautes réelles sont basés sur le cadre fourni dans [Arlat 2002a] pour étudier la pathologie, l'équivalence et la représentativité des fautes injectées par rapport aux fautes réelles. Ces travaux nous apprennent plusieurs leçons. Nous avons expliqué dans le paragraphe 2.3.3 les travaux réalisés dans [Jarboui 2002a]. Les conséquences de l'injection des fautes au niveau de l'interface noyau-application (l'*API*) de *Linux* ont été étudiées et résumées dans le Tableau 2-1 :

Tableau 2-1 Analyse comparative des techniques d'injection de fautes à l'*API*

	Durée	Facilité d'application	Codes d'erreur provoqués	Propagation d'erreurs	Pression de mémoire	Représentativité des expériences
Inversion de bits	-	+	+	-	-	-
Substitution sélective	+	-	-	+	+	+

Ce tableau compare les deux techniques d'injection de fautes par substitution systématique par inversion de bit ou par substitution sélective. L'utilisation de ces deux méthodes de corruption de paramètres au niveau de l'API forme un moyen très efficace pour évaluer la robustesse du noyau. La substitution systématique par inversion de bit des paramètres des appels système est plus facile à implémenter que la substitution sélective. Par contre, la durée nécessaire pour exécuter des expériences en utilisant des inversions de bit est beaucoup plus longue que dans le cas de la deuxième technique, malgré le fait qu'un certain temps soit nécessaire pour définir la base de données contenant les valeurs incorrectes qui doivent remplacer les valeurs des paramètres à corrompre. Il est à noter que cette analyse a priori de types des paramètres corrompus est effectuée une seule fois par interface, comme dans le cas de *Ballista*. D'autre part, malgré l'équivalence des deux techniques d'injection de fautes en termes de modes de défaillances produits, la substitution systématique a généré des codes d'erreurs plus diversifiés que la substitution sélective. La substitution sélective a plutôt tendance à propager des erreurs entre les différents composants fonctionnels, et à

exercer plus de pression sur la mémoire. Le Tableau 2-1 montre que la technique d'injection par substitution sélective offre plus d'avantages que la substitution systématique par bit-flip.

2.4.4 La portabilité

La comparaison de la sûreté de fonctionnement de plusieurs systèmes informatiques est l'objectif majeur de l'étalonnage de sûreté de fonctionnement. Un étalon de sûreté de fonctionnement doit pouvoir être exécuté sur différents systèmes informatiques qui n'ont pas forcément les mêmes caractéristiques ou architecture. Par conséquent, l'étalon doit être portable sur ces différents systèmes.

La portabilité de l'étalon doit être prise en considération dès les premières étapes de l'étalonnage, dès la phase de spécification. La portabilité des spécifications est beaucoup plus facile à réaliser que celle de l'implémentation. Par conséquent, moins la spécification est détaillée, plus on augmente les chances de porter l'étalon sur d'autres systèmes. À l'inverse, la propriété de reproductibilité impose des détails sur les différents composants de l'étalon, afin de garantir la répétitivité des résultats. Un compromis est donc nécessaire. La leçon à retenir est que les propriétés doivent être prises en considération dès les premières phases de spécification.

Les expériences du développement logiciel ont montré que la programmation modulaire permet l'augmentation de la capacité de portabilité au niveau d'implémentation. De plus, éviter l'utilisation des logiciels et des matériels spécifiques permet de mieux porter l'étalon sur différents systèmes.

Un étalon de sûreté de fonctionnement devrait à la fois être portable et permettre l'injection de fautes et l'observation de leurs effets à différents niveaux du système à étalonner. Le fait d'étalonner des systèmes COTS ou basés sur des COTS facilite cette tâche, dans la mesure où les points d'accès et d'observation sont définis comme étant les points d'entrée et de sortie (les interfaces) de ces systèmes, ce qui permet de satisfaire à la fois l'assurance de la portabilité et la flexibilité aux niveaux d'injection et d'observation.

2.4.5 La non-intrusivité

Certains systèmes doivent être modifiés afin de permettre l'étalonnage de leur sûreté de fonctionnement. Ces modifications peuvent affecter le système cible lui-même au travers de son code source, ou peuvent intervenir au niveau physique par l'ajout de matériels sur le système à étalonner. Dans ce cas, l'étalon est dit intrusif. L'intrusivité est très coûteuse du point de vue temporel, et il est souvent impossible de modifier les systèmes cibles s'ils sont des COTS ou des systèmes incluant des COTS, étant donné que la plupart des COTS existants sur le marché sont de type « boîte noire ». De plus, l'intrusivité réduit la reproductibilité et la portabilité de l'étalon sur d'autres systèmes cibles. En conséquence, un étalon de sûreté de fonctionnement doit éviter au maximum l'intrusivité, ce qui revient à éviter l'injection des fautes à l'intérieur des systèmes cibles, et à privilégier plutôt l'injection au niveau de l'interface du système cible.

2.4.6 L'interférence

Appliqué à un système, un étalon de sûreté de fonctionnement peut avoir des influences indésirables. Ces influences, comme l'effet d'exécution de l'étalon sur l'ordonnanceur ou la mémoire d'un système d'exploitation, sont regroupées sous le thème de l'interférence. La différence avec l'intrusivité est que cette dernière est directe, volontairement mise en place et généralement contrôlée par l'exécuteur ou le développeur de l'étalon, tandis que l'interférence est un effet secondaire, prévu ou non, qui apparaît contre toute volonté.

Par définition, l'injection des fautes interfère avec le système cible, et par conséquent, l'interférence est inévitable dans le domaine de l'étalonnage de sûreté de fonctionnement qui utilise la méthode d'injection de fautes. Cependant, une distinction doit être effectuée entre les interférences qui peuvent avoir des effets acceptables sur les mesures de l'étalon, et celles qui peuvent avoir des conséquences significatives sur les mesures obtenues, et dont il faut essayer de minimiser l'impact.

2.4.7 La mise à l'échelle

L'étalonnage de la sûreté de fonctionnement devrait pouvoir s'appliquer à des systèmes qui n'ont pas nécessairement la même taille. La mise à l'échelle de l'étalon est assurée par la définition d'un ensemble de règles de mise à l'échelle, lors de la spécification de l'étalon. Ces règles concernent en particulier l'activité de l'étalon et l'ensemble de fautes à appliquer.

2.4.8 L'automatisation

L'étalonnage de la sûreté de fonctionnement d'un système informatique se fait généralement en plusieurs étapes, pendant lesquelles des interventions humaines seraient très probables. La propriété d'automatisation consiste à exécuter l'étalon avec le minimum possible d'interventions.

Cette automatisation a une grande influence sur d'autres propriétés de l'étalon, comme celle du « coût de l'étalonnage » et la facilité d'utilisation. Un bon étalon de sûreté de fonctionnement doit assurer un niveau assez élevé d'automatisation.

2.4.9 Le coût de l'étalonnage

Le temps de l'étalonnage de sûreté de fonctionnement est le temps nécessaire pour obtenir les résultats d'un étalon. Il est essentiellement composé de trois parties : 1) le temps nécessaire pour l'implémentation et la préparation, 2) le temps d'exécution de l'étalon et 3) l'analyse des données brutes pour obtenir les mesures finales. Il est souhaitable et fort recommandé que ce temps d'exécution soit le plus petit possible.

Un étalon idéal est un étalon représentatif qui caractérise le système le plus fidèlement possible et dans les meilleurs délais. Cela peut être en contradiction avec d'autres propriétés comme la représentativité des fautes et de l'activité. L'automatisation totale du processus de l'étalonnage permet de diminuer le temps d'exécution de l'étalon.

2.5 Conclusion

Dans ce chapitre, nous avons d'abord présenté un état de l'art sur les travaux réalisés dans le cadre de l'étalonnage de sûreté de fonctionnement des systèmes informatiques. Ensuite, nous avons discuté les différentes propriétés des étalons.

Dans un premier temps, nous avons présenté les différents étalons de sûreté de fonctionnement ciblant un large spectre de systèmes informatiques (systèmes d'exploitation, systèmes de gestion de base de données, etc.).

Dans un deuxième temps, nous avons présenté un cadre générique qui permet de définir un étalon de sûreté de fonctionnement. Ce cadre concerne une catégorie de systèmes allant des logiciels COTS, tels que les systèmes d'exploitation, jusqu'aux systèmes complexes à base de COTS. Nous avons identifié les dimensions nécessaires pour définir un étalon de sûreté de fonctionnement. Les dimensions de catégorisation définissent clairement le système cible (y compris son environnement opérationnel) pour lequel l'étalon s'applique et le contexte d'étalonnage (l'utilisateur des résultats, le développeur de l'étalon…). Les mesures requises sont définies en fonction des dimensions de catégorisation. Ensuite, nous avons défini les dimensions nécessaires pour réaliser les expérimentations : le système à étalonner, l'activité, l'ensemble de fautes et les observations à relever.

En pratique, ces points ne sont pas totalement indépendants les uns des autres. D'un point de vue expérimental, les dimensions de catégorisation et les mesures ont un grand impact sur la définition des dimensions expérimentales. En plus, les résultats de l'étalonnage ne sont significatifs et interprétables que s'ils sont accompagnés par la définition de toutes ces dimensions.

Étant donné qu'un étalon de sûreté de fonctionnement se distingue d'autres techniques d'évaluation et de validation par l'accord qu'il doit assurer entre les différentes communautés qu'ils utilisent, nous avons défini un ensemble de propriétés que l'étalon doit satisfaire. Entre autres, l'étalon de sûreté de fonctionnement doit assurer la représentativité des mesures, de l'activité utilisée et de l'ensemble de fautes à appliquer. Il doit également être capable de reproduire les mêmes résultats s'il est appliqué deux fois sur le même système.

Basés sur le cadre conceptuel présenté dans le présent chapitre, nous présentons dans le chapitre suivant la spécification de trois prototypes destinés à étalonner la sûreté de fonctionnement des systèmes d'exploitation à usage général. Ces prototypes nous serviront dans un deuxième temps à évaluer et à comparer la sûreté de fonctionnement des systèmes d'exploitation d'une même famille (cf. chapitre 4) ou de familles différentes (cf. chapitre 5).

Chapitre 3 Spécification d'étalons de systèmes d'exploitation

3.1 Introduction

Les étalons de sûreté de fonctionnement que nous proposons sont définis sous forme d'un ensemble de spécifications. Inspirés du cadre conceptuel élaboré par le projet DBench et décrit dans le deuxième chapitre (cf. section 2.4), nous présentons dans ce chapitre la spécification des différentes entités et procédures nécessaires pour développer trois étalons de sûreté de fonctionnement destinés à caractériser la sûreté de fonctionnement des systèmes d'exploitation en présence de fautes [Kalakech 2004a]. Ces trois étalons de sûreté de fonctionnement se différencient essentiellement par les profils d'exécution utilisés.

Les trois étalons que nous proposons dans nos travaux sont des étalons de robustesse. La robustesse étant définie comme la capacité du système d'exploitation à réagir à des entrées exceptionnelles ou à des conditions d'environnement stressantes. La robustesse de l'OS peut être vue comme une indication de sa capacité à résister/réagir à des entrées fautives induites par 1) l'interface matérielle, 2) l'interface entre le noyau et les pilotes de périphériques et 3) l'interface entre le noyau et les applications (l'*API*). La robustesse concerne également la capacité du système à résister aux malveillances comme les intrusions ou les virus, ou bien aux fautes d'interaction des utilisateurs ou des administrateurs du système. Vis-à-vis des virus et des intrusions, la présence des trous de sécurité dans le système est considérée comme manque de sécurité. Des outils, dits de tests de pénétration, existent déjà comme MUCUS [Mutz 2003] et NESSUS [NESSUS 2004].

Pour étalonner la sûreté de fonctionnement d'un système d'exploitation, l'idéal serait de tenir compte de toutes ces possibilités, de façon séparée ou combinée. Dans nos travaux, nous nous sommes intéressés à l'étalonnage de la sûreté de fonctionnement des systèmes d'exploitation par rapport aux fautes d'origine applicative.

Le problème majeur que nous avons rencontré lors de la spécification des étalons est de nous assurer que ces dernières soient suffisamment générales pour être applicables à un grand nombre de systèmes d'exploitation. En même temps, la spécification doit être suffisamment précise pour assurer la reproductibilité de l'étalon et son implémentation d'une manière équivalente par des personnes ou équipes différentes.

Les trois étalons définis dans ce chapitre diffèrent par le profil d'exécution, et plus particulièrement par l'activité d'étalonnage, ce qui entraîne des ensembles de fautes différents. Cependant, la technique d'injection de fautes est la même pour les trois étalons. Le premier étalon utilise le client TPC-C comme activité, le deuxième utilise Postmark et le dernier la machine virtuelle java. Toutes les autres dimensions sont identiques. Nous appelons ces trois étalons DBench-OS-TPCC, DBench-OS-Postmark et DBench-OS-JVM respectivement.

Ce chapitre est structuré en deux parties. La première partie concerne la spécification des étalons. La deuxième partie est consacrée à la vérification des propriétés de ces étalons.

Dans la première partie, nous présentons le système cible des étalons et le contexte d'étalonnage, avant de spécifier l'ensemble de mesures fournies par les étalons que nous proposons. Les expériences acquises de l'étalonnage de performance montrent que les mesures fournies par un étalon doivent être claires pour qu'elles puissent être comprises par les utilisateurs finaux qui ne sont pas nécessairement des experts dans le domaine. Ensuite, nous présentons les éléments nécessaires pour l'expérimentation où nous spécifions en particulier les profils d'exécution utilisés pour chacun des trois étalons, la méthode suivie pour corrompre les appels système et les observations à relever pour calculer les mesures. Enfin, nous spécifions les éléments nécessaires à la mise en oeuvre d'un environnement d'étalonnage de sûreté de fonctionnement d'un OS, et la conduite des expériences.

La mise en œuvre des étalons de sûreté de fonctionnement doit tenir compte des propriétés présentées dans le deuxième chapitre. Certaines de ces propriétés peuvent être directement vérifiées par construction lors de la spécification de l'étalon. Parmi ces propriétés nous citons la reproductibilité, la représentativité des mesures et de l'activité, la portabilité. Dans la deuxième partie de ce chapitre, nous montrons comment certaines propriétés de l'étalon (définies dans la section 2.4) ont été prises en compte lors de la spécification.

Par abus de langage, nous utilisons dans ce chapitre et les chapitres suivants le terme OS ou système d'exploitation pour désigner les systèmes d'exploitation à usage général, et étalon pour les étalons de sûreté de fonctionnement de systèmes d'exploitation, sauf si c'est spécifié autrement.

3.2 Spécification des étalons

Un étalon de sûreté de fonctionnement doit clairement définir le système cible, le contexte d'étalonnage, les mesures à évaluer, les dimensions d'expérimentation et l'environnement de l'étalonnage. Nous mentionnons que les résultats de l'étalon ne peuvent être utilisables et interprétés que s'ils sont accompagnés par la description de tous ces éléments.

3.2.1 Système cible et contexte d'étalonnage

La cible de l'étalonnage est considérée comme étant l'ensemble des services que l'OS met à la disposition de l'application pour s'exécuter comme indiqué dans la Figure 2-2. En d'autres termes, c'est l'union des ensembles contenant le noyau proprement dit du système d'exploitation et les composants fonctionnels. Ces composants fonctionnels peuvent être

intégrés au noyau comme dans le cas du noyau monolithique des systèmes *Linux*, ou à l'extérieur du noyau comme le cas des systèmes *Windows*. Toutefois, pour pouvoir évaluer la cible, il est nécessaire de disposer d'une plate-forme matérielle sur laquelle l'OS s'exécutera ainsi que d'un ensemble de bibliothèques et pilotes de périphériques. La cible associée à la plate-forme matérielle et les bibliothèques nécessaires à son exécution sous un profil d'étalonnage donné constituent le système à étalonner. Bien qu'en pratique les résultats fournis par l'étalon caractérisent l'ensemble du système utilisé lors de l'étalonnage, en raison, par exemple, du fort impact du matériel sur des mesures telles que le temps de réaction ou de redémarrage de l'OS, nous afficherons les résultats de l'étalon comme une caractérisation de l'OS. Pour comparer la sûreté de fonctionnement de deux OSs, il est essentiel d'utiliser la même plate-forme matérielle.

Dans notre étude, nous avons mis l'accent sur l'analyse de la robustesse vis-à-vis de comportements erronés des applications, et plus précisément de possibles appels système erronés délivrés par le niveau applicatif à l'OS via son API. Les appels système erronés, censés représenter l'application erronée, sont obtenus par la corruption de paramètres des appels système issus des applications. Cette corruption consiste à remplacer les valeurs des paramètres des appels système interceptés par des valeurs erronées. Par la suite, nous utiliserons le terme de fautes pour faire référence à de tels paramètres erronés.

L'étalon de sûreté de fonctionnement proposé est principalement destiné à l'utilisateur d'un OS, ou plus précisément à un concepteur de système utilisant l'OS en tant que composant de base pour son futur système. De ce fait, la mise en œuvre de l'étalon est telle qu'elle n'exige pas de connaissance approfondie de l'OS cible. En effet, l'architecture interne n'étant pas prise en compte, les techniques d'évaluation que nous proposons peuvent être réutilisées sur des systèmes différents. L'OS est considéré comme une boîte noire et, par conséquent, il n'est pas nécessaire de disposer de son code source. Ce qui permet d'assurer la portabilité de l'étalon sur différents systèmes d'exploitation, et par conséquent, l'atteinte de l'objectif principal correspondant à la comparaison de ces différents systèmes. La seule information nécessaire est la description de l'OS en termes d'appels système, en plus bien sûr de la description des services offerts par l'OS. Les résultats de l'étalonnage doivent permettre d'appréhender de manière significative le comportement d'un OS en présence de fautes et comparer sur ce point plusieurs alternatives d'OS.

3.2.2 Mesures fournies par l'étalon

Les mesures fournies par l'étalon sont déduites de l'analyse des différentes issues de l'OS suite à des appels système erronés qui lui sont soumis via son API[2]. Les mesures que nous

[2] Ces mesures sont différentes des mesures conventionnelles de fiabilité du logiciel telles que sont définies dans [Li 2000]. Nos mesures caractérisent le comportement de l'OS en présence de fautes qui lui sont externes. Les mesures conventionnelles caractérisent essentiellement la qualité du logiciel vis-à-vis de ses fautes internes. Ces dernières sur une connaissance fine du logiciel qui sont généralement la propriété du développeur du logiciel. Nos mesures permettent de les compléter.

avons considérées concernent 1) les mesures de robustesse qui permettent de caractériser de manière qualitative et quantitative la résistance de l'OS aux fautes du niveau applicatif, 2) des mesures temporelles permettant de caractériser le temps de réaction de l'OS à des appels système erronés, ainsi que la durée de redémarrage de l'OS. Ces mesures de base constituent l'ensemble de résultats fournis par l'étalon. Cependant, des analyses complémentaires peuvent être effectuées pour les affiner. Nous présentons dans ce paragraphe les mesures de base suivies des mesures complémentaires.

3.2.2.1 Mesures de base

Avant de définir les mesures de robustesse proposées, nous définissons tout d'abord les différentes issues qu'il est possible de distinguer à la fin du traitement d'un appel système erroné. Suite à l'exécution d'un appel système erroné, il est possible de distinguer au niveau de l'OS quatre issues principales :

- **Er** : retour de **code erreur** vers l'application**,**
- **Xp** : une **exception** est levée par l'OS. Deux sortes d'exceptions peuvent être distinguées suivant que l'exception survient au cours de l'exécution de l'application (mode utilisateur) ou durant l'exécution de primitives du noyau (mode noyau). Dans le mode utilisateur, l'OS traite l'exception et la notifie à l'application, qui prend elle explicitement en compte ou non cette information. Toutefois, dans certaines situations critiques, l'application est automatiquement interrompue par l'OS. Une exception dans le mode noyau est automatiquement suivie par un état *panique* (par exemple, « écran bleu » pour Windows ou messages « oops » pour Linux). Par la suite, ce dernier type d'exceptions est confondu avec l'état de panique et le terme exception fait référence uniquement aux exceptions survenant en mode utilisateur.
- **Pq** : ce cas correspond à l'état de **panique** dans lequel l'OS reste « vivant », mais ne peut plus servir les applications. Dans certains cas, un redémarrage logiciel est suffisant pour faire repartir le système.
- **Bc** : il s'agit d'un état de **blocage** dans lequel un redémarrage matériel de l'OS est nécessaire.

Aux quatre issues précédentes, nous ajoutons le cas où aucune de ces quatre issues ne surviendrait et que nous nommons par la suite issue de « Non-signalement », **NS**.

Les deux premières issues (retour de code d'erreur et exception) se manifestent par des sorties de l'OS vers l'application. Ces deux issues sont préférables aux autres issues car, si l'application est bien conçue, elle est capable de les traiter tandis que les deux cas de Blocage et de Panique de l'OS sont considérés comme critiques.

Le Tableau 3-1 synthétise les différentes issues possibles pour un OS lors de la prise en compte d'appels système erronés.

Tableau 3-1 : Issues possibles pour l'OS

Er	*Un code erreur est retourné*
Xp	*Une exception est levée, notifiée à l'application et éventuellement traitée par celle-ci*
Pq	*Etat de Panique*
Bc	*Etat de Blocage*
NS	*Aucune des issues précédentes n'est observée (Non-signalement)*

Remarques :

- Les issues *Panique* et *Blocage* correspondent à un état particulier de l'OS et nécessitent une surveillance externe pour être identifiées. À l'opposé, les issues Er et Xp correspondent à des événements et elles peuvent être facilement identifiées lorsqu'un code d'erreur est retourné ou une exception est notifiée par l'OS.
- Il est possible que, au cours d'une seule exécution de l'activité (au sein d'une même expérience), plusieurs codes d'erreur et/ou exceptions apparaissent successivement. L'ordre dans lequel apparaissent ces événements peut être utilisé pour mieux caractériser le comportement du système comme dans [Rodríguez 2002] et [Albinet 2004]. Dans le cas de l'étalon de sûreté de fonctionnement proposé, le comportement du système est d'abord caractérisé en se basant uniquement sur le premier événement. Si nécessaire, un raffinement ultérieur pourrait être effectué en se basant sur l'ensemble des événements qui auront été relevés au cours d'une expérience et consignés dans un fichier historique associé à l'expérience.

Mesures de robustesse

Pour obtenir des mesures, une cible sera soumise, au cours d'une campagne, à une série d'expériences indépendantes. Chaque expérience consiste à exécuter l'activité retenue en présence d'une seule faute, c'est-à-dire un seul appel système erroné. Le système est redémarré après chaque expérience.

La mesure de robustesse, $\mathbf{P_{OS}}$, correspond aux pourcentages des expériences conduisant à chacune des issues répertoriées dans le Tableau 3-1. Il s'agit donc d'un vecteur composé de cinq éléments.

Temps de réaction de l'OS

Cette mesure correspond au temps moyen pris pour l'OS pour réagir à un appel système erroné, soit en notifiant une exception soit en retournant un code d'erreur, ou encore en exécutant l'ensemble des instructions de l'appel (que le résultat soit correct ou non).

Nous nommerons **Texec**, le temps moyen de réaction en présence de fautes, et **τexec**, la durée moyenne d'exécution de l'appel en absence de fautes.

Temps de redémarrage de l'OS

Le temps d'un redémarrage de l'OS est aussi une mesure importante pour les concepteurs d'applications critiques car, pendant ce temps, le système est indisponible. Bien que, en mode nominal (c'est-à-dire en l'absence de fautes), le temps de redémarrage d'un OS soit presque déterministe, il peut être affecté par l'appel système erroné, surtout s'il provoque un état de panique ou un blocage de l'OS. En effet, le redémarrage peut requérir un temps supplémentaire en raison de possibles actions nécessaires au recouvrement par exemple du système de fichiers, recouvrement qui peut se révéler plus ou moins efficace en fonction du type d'OS considéré et dépendre des dégâts occasionnés par l'appel système erroné.

Nous nommerons **Tred** la durée moyenne de redémarrage en présence de fautes et **τred** la durée moyenne de redémarrage en absence de fautes.

3.2.2.2 Mesures complémentaires

Les étalons que nous avons développés vont au-delà de ces mesures de base et proposent un ensemble de mesures complémentaires qui permettent d'affiner l'analyse du comportement de différents OS en présence de fautes en prenant en compte l'état de l'activité après la corruption des paramètres des appels système. Ces mesures complémentaires incluent 1) la caractérisation de l'état de l'activité en combinaison avec l'état de l'OS et 2) l'affinement des temps de réaction de l'OS.

Caractérisation de l'activité

Avant de définir les mesures proposées pour caractériser l'activité, nous présentons d'abord les différentes manifestations et issues de l'activité que nous pouvons distinguer après l'exécution d'un appel système erroné ; ces issues possibles de l'activité permettent de définir dans un premier temps des mesures de sûreté de fonctionnement propres à l'activité, et dans un deuxième temps, des mesures combinées de l'OS et de l'activité.

L'observation de l'état final de l'activité après le traitement de l'appel système erroné permet d'identifier l'impact de la réaction de l'OS sur l'activité. L'activité est caractérisée par une des manifestations suivantes : 1) elle termine correctement son exécution, 2) elle termine son exécution mais en fournissant des résultats erronés, 3) l'activité abandonne son exécution, ou 4) elle peut être bloquée. Ces issues sont synthétisées dans le Tableau 3-2. Nous considérons que *TAc* correspond aux cas où l'activité a terminé, correctement ou incorrectement, son exécution ($TAc = TC \cup TI$).

Tableau 3-2 Issues possibles pour l'activité

TC	*Terminaison correcte*
TI	*Terminaison incorrecte*
Ab	*Abandon*
Bc	*Blocage*

Le Tableau 3-3 synthétise les différents états possibles résultants de la combinaison de différents états de l'OS et de l'activité. L'activité peut terminer son exécution, être en état d'abandon ou de blocage dans les trois issues possibles Er, Xp et NS de l'OS. Dans le cas où l'issue du système d'exploitation correspond à l'état de panique, il est clair que l'activité n'est pas capable de terminer son exécution alors que l'issue correspondant au blocage de l'OS implique le blocage de l'activité.

Du fait que dans les cas de non signalement de l'OS nous ne disposons pas de renseignements sur l'état de l'OS, il est intéressant de raffiner les mesures de robustesse de l'OS en les combinant avec les différents états de l'activité. Ainsi, la robustesse de l'OS ($\mathbf{P_{OS}}$) peut être raffinée en tenant compte des différents états de l'activité. Plus particulièrement ceux correspondant aux différents états de l'activité dans les cas de non signalement de l'OS (dernière colonne de ce tableau). Ce vecteur composé de quatre éléments, $\mathbf{P_{NS}}$, est appelé la *robustesse de l'activité* pour des raisons de simplification.

Tableau 3-3 Issues combinées possibles

OS → *↓ Activité*	*Code d'erreur*	*Exception*	*Panique*	*Blocage*	*Non signalement*
Terminaison Correcte	*Er – TC*	*Xp – TC*	—	—	*NS – TC*
Terminaison incorrecte	*Er – TI*	*Xp – TI*	—	—	*NS – TI*
Abandon	*Er – Ab*	*Xp – Ab*	*Pc – Ab*	—	*NS – Ab*
Blocage	*Er – Bc*	*Xp – Bc*	*Pc – Bc*	*Bc – Bc*	*NS – Bc*

En ce qui concerne les mesures temporelles de l'activité, nous nous intéressons à la durée moyenne nécessaire à l'activité pour terminer son exécution, ce qui revient à calculer la moyenne des durées d'exécution de l'activité dans les différents cas de *TAc*. Considérons ***TTAc*** le temps moyen nécessaire à la terminaison de l'exécution de l'activité en présence de fautes et $\boldsymbol{\tau}$***TAc*** la durée moyenne d'exécution de l'activité en absence de fautes.

Affinement du temps de réaction de l'OS

Le temps de réponse en présence de fautes peut être évalué par rapport aux différentes issues répertoriées dans le Tableau 3-1. Les temps associés seront respectivement notés TEr, TXp et TNS. Ils correspondent à la moyenne de l'intervalle de temps séparant l'instant de soumission de l'appel système erroné à l'OS de l'instant d'apparition de l'événement correspondant : 1) retour d'un code d'erreur, 2) notification d'une exception, 3) retour de la réponse à l'appel système.

3.2.2.3 Récapitulatif

Nous résumons dans ce paragraphe l'ensemble des mesures que nous proposons pour étalonner la sûreté de fonctionnement des systèmes d'exploitation à usage général.

Le Tableau 3-4 présente les deux mesures caractérisant la robustesse de l'OS (P_{OS}) et de l'activité (P_{NS}). Les mesures temporelles fournies dans le Tableau 3-5 sont les valeurs moyennes obtenues sur l'ensemble des expériences réalisées. L'association à ces valeurs d'autres valeurs telles que l'écart type, valeurs minimum et maximum peut aussi présenter un certain intérêt.

Tableau 3-4 Mesures de robustesse fournies par l'étalon

Mesure	Définition
P_{OS}	*Robustesse de l'OS* *Comportement de l'OS en présence de fautes — vecteur de 5 éléments*
P_{NS}	*Robustesse de l'activité* *Comportement de l'activité dans les cas de non signalement de l'OS— vecteur de 4 éléments*

Tableau 3-5 Mesures temporelles fournies par l'étalon

Temps de réaction

τexec	*Moyenne du temps d'exécution de tous les appels système dont les paramètres sont corrompus pour les besoins de l'étalonnage, en absence de fautes*	
Texec	*Moyenne du temps d'exécution de tous les appels système dont les paramètres sont corrompus pour les besoins de l'étalonnage, en présence de fautes*	
	TEr	*Temps moyen pris pour retourner un code d'erreur*
	TXp	*Temps moyen pris pour notifier une exception*
	TNS	*Temps moyen pris pour exécuter l'appel système erroné (non-signalement de l'erreur)*

Temps de redémarrage

τred	*Durée moyenne de redémarrage de l'OS en absence de fautes*
Tred	*Durée moyenne de redémarrage de l'OS en présence de fautes*

Durée d'exécution de l'activité

τTac	*Durée moyenne d'exécution de l'activité en absence de fautes*
Ttac	*Durée moyenne d'exécution de l'activité en présence de fautes*

L'ensemble des mesures formé par Pos, Texec et Tred constitue l'ensemble de mesures de base d'un étalon de sûreté de fonctionnement. Le reste des mesures présentées dans le Tableau 3-4 et le Tableau 3-5 forme l'ensemble des mesures complémentaires de l'étalon. Elles permettent d'améliorer la compréhension par l'utilisateur des résultats de l'étalon par

rapport au comportement de l'OS en présence de fautes. Nous rappelons que, en général, les étalons de performance existants fournissent un nombre limité de mesures. Les utilisateurs des résultats des étalons que nous proposons peuvent ignorer les mesures qui ne les intéressent pas. Cependant, toutes les mesures des tableaux ci-dessous sont calculées à partir des informations enregistrées au cours du déroulement des expériences, et ne nécessitent pas d'expériences supplémentaires. Seule P_{NS} nécessite une analyse spécifique car elle nécessite la connaissance de l'état de l'activité.

3.2.3 L'expérimentation

Dans ce paragraphe, nous présentons tout d'abord la technique de corruption de paramètres utilisée dans les étalons que nous avons définis. Nous passons ensuite en revue chacun des trois étalon pour présenter à la fois l'activité d'étalonnage et l'ensemble des fautes utilisées. Enfin, nous spécifions les observations à relever pour évaluer les mesures définies dans le paragraphe précédent.

3.2.3.1 Techniques de corruption des paramètres des appels système

La technique de corruption de paramètres utilisée dans nos travaux est la technique de *substitution sélective*[3], utilisée dans [Koopman 1997] et qui se base sur une analyse complète des paramètres des appels système pour définir une sélection des substitutions qui sont appliquées à ces paramètres.

Un paramètre est soit une *donnée*, soit une *adresse*. La valeur d'une donnée peut être substituée par une valeur *hors limites* du domaine admissible (par exemple une valeur négative pour un paramètre de type entier non signé) ou par une valeur *incorrecte* (mais non hors limite, comme une valeur aléatoire choisie dans le domaine admissible), tandis qu'une adresse peut être remplacée par une adresse *incorrecte* (mais existante) contenant une donnée incorrecte ou hors limite. Le choix de la substitution sélective comme technique de corruption de paramètres est le résultat d'une étude de sensibilité décrite dans [Crouzet 2004] dont les résultats sont présentés dans le quatrième chapitre.

Afin de réduire le nombre d'expériences, les types de données associés aux paramètres sont regroupés en classes. Un ensemble de valeurs de substitution est alors associé à chaque classe, en fonction de la définition de la classe. Nous avons défini 19 classes de type de

---

[3] Plusieurs techniques de corruption de paramètres peuvent être utilisées pour simuler les fautes que l'applicatif peut communiquer au système d'exploitation. Nous distinguons en particulier la substitution systématique par inversion de bit et la substitution sélective. Des travaux de comparaison entre ces deux techniques ont montré leur équivalence en termes d'erreurs provoquées [Jarboui 2002]. L'application de la technique de substitution systématique nécessite un temps d'expérimentation beaucoup plus élevé que celui relatif à la technique de substitution sélective.

paramètre pour Windows et 13 classes de type de paramètre pour Linux. Ces classes peuvent correspondre à des types de données de base (`integer` *(entiers)*, `pvoid` *(pointeur qui pointe vers rien)*, …) comme elles peuvent correspondre à des types de données composées (comme le *pointeur vers un entier*). Un type de donnée composée hérite des valeurs de substitution des types de base dont il dérive. Par exemple, l'ensemble de valeurs de substitution d'un paramètre de type « *pointeur vers un entier* » hérite toutes les valeurs de substitution des types de données *Pvoid* et *Integer*. Etant donné que les appels système de ces deux familles d'OS sont développés en langage C, les mêmes types de données de base sont utilisés par les deux familles d'OS, et par conséquent, les mêmes valeurs de substitution sont appliquées aux appels système des OSs étudiés. De plus, la notion d'héritage des valeurs de substitution d'un type de donnée composée reste valide pour les deux familles d'OS. Le Tableau 3-6 illustre des valeurs de substitutions associées aux classes de type de données de base.

Tableau 3-6 Valeurs de substitution par classes de type de données

Classe de type de donnée	Valeurs de substitution					
Pvoid	*NULL*	*0xffffffff*	*1*	*0xFFFF*	*-1*	*Random*
Integer	*0*	*1*	*(Max INT)*	*(Min INT)*	*0,5*	
unsigned integer	*0*	*1*	*0xffffffff*	*-1*	*0,5*	
Boolean	*0*	*0xff (Max)*	*1*	*-1*	*0,5*	
String	*Vide*	*très grand (> 200)*	*très loin (+ 1000)*			

3.2.3.2 Profils d'exécution

Deux manières peuvent être envisagées pour solliciter les OSs avec des requêtes erronées. Dans la première, l'ensemble de fautes est appliqué en intégrant les fautes avec l'activité (programme instrumenté ou modifié pour produire une activité avec des appels erronés). La deuxième manière consiste à utiliser un module séparé pour corrompre les paramètres des appels système activés. Pour assurer une plus grande flexibilité, nous avons opté pour la deuxième solution qui permet une plus grande portabilité. Ainsi, n'importe quel étalon de performance disponible sous format d'exécutable peut être utilisé, sans avoir besoin de modifier son code source. Notre premier étalon spécifié utilise comme activité privilégiée le client TPC-C issu des étalons de performance de systèmes transactionnels [TPC]. L'étalon de performance TPC-C n'est pas forcément le plus adapté pour caractériser les OSs, mais nous l'avons utilisé pour des raisons d'homogénéité avec les autres partenaires du projet DBench. La version du client TPC-C dont nous disposons est destinée à être exécutée sur les OSs de la famille Windows seulement. Une analyse des étalons de performance existants nous a permis de retenir d'autres activités réalistes comme celle de l'étalon de

performance *Postmark* utilisé pour étalonner la performance de systèmes de fichiers et dont le code source est disponible (en langage C), et par conséquent, capable de s'exécuter sur différentes familles d'OS. La machine virtuelle Java (*JVM : Java Virtual Machine*) a été également retenue comme activité réelle capable d'être exécutée sur les OSs de différentes familles.

Le ***client TPC-C*** est une activité réaliste utilisée dans le cadre des étalons de performance des systèmes transactionnels en ligne. Il existe sous forme de spécification. L'activité permet de simuler un utilisateur qui envoie des requêtes à un serveur de base de données (cf. section 1.3). L'implémentation utilisée est la même que celle des autres partenaires de DBench [Vieira 2003b].

Postmark [Katcher 1997] est un micro-étalon de performance de systèmes de fichiers. Développée en langage C pour modéliser une charge de type « courrier électronique » et « nouvelles sur le réseau », l'activité de Postmark effectue des opérations sur le système de fichiers, telles que la création et l'effacement de petits fichiers, l'effacement et l'insertion des données dans ces fichiers. En effet, cette activité va tout d'abord créer un nombre de fichiers spécifié par l'utilisateur puis va effectuer une séquence de transactions. Chaque transaction consiste en deux opérations aléatoires : une création ou un effacement de fichier et une lecture ou une écriture. À la fin, les fichiers et les répertoires sont effacés. Postmark a été choisi d'une part pour sa portabilité sur Windows et Linux et pour la facilité de sa configuration et de sa mise en œuvre.

La machine virtuelle Java (JVM) est une couche logicielle qui assure le lien entre le système d'exploitation et les applications Java. De nos jours, la *JVM* est de plus en plus répandue du fait de l'utilisation grandissante du langage de programmation Java. La *JVM* garantit aux applications développées en Java de pouvoir être exécutées sur n'importe quelle plate-forme avec des résultats identiques. Les spécifications de la machine virtuelle sont indépendantes des plate-formes. Cependant, chaque plate-forme a besoin d'une implémentation particulière de la JVM. Pour solliciter la machine virtuelle Java, nous avons utilisé un programme Java qui affichait le message « *Hello World* » à l'écran.

Appels système ciblés

La première étape pour choisir les appels système sur lesquels porteront les expériences consiste à identifier tous les appels système utilisés par l'activité retenue ainsi que leurs occurrences. De manière idéale, dans le cas d'une non-limitation du temps alloué à la réalisation des expériences, tous les appels système avec paramètres et utilisés par une activité devraient être corrompus. Ceci est envisageable pour de petits programmes qui n'activent pas un nombre important d'appels système.

Nous nous sommes fixés une durée maximale de cinq jours pour qu'un étalon soit accepté dans un contexte industriel. Pour satisfaire la propriété de coût temporel d'étalonnage et respecter les limites que nous nous sommes fixées, il est donc important de ne cibler qu'un sous ensemble des appels système. La sélection effectuée dépend des appels système qui sont les plus pertinents et de la durée de l'expérimentation qui est acceptable.

La durée d'exécution d'un étalon dépend de la durée d'une expérience élémentaire et du nombre d'expériences à réaliser. Dans le cas où l'on cherche à traiter toutes les configurations de fautes possibles (traitement exhaustif), le nombre d'expériences dépend : 1) du nombre d'appels système à corrompre, 2) du nombre de paramètres à corrompre pour un appel système, et 3) du nombre de valeurs de substitution associées à chaque paramètre. À partir de nos travaux sur différents OS, nous avons pu constater que la moyenne du nombre de paramètres par appel système est approximativement trois, et que la moyenne du nombre de valeurs de substitution par classe de type de données est approximativement égale à 7. Nous avons également pu estimer que le temps moyen d'une expérience est de l'ordre de 5 minutes. Avec toutes ces informations et en utilisant un environnement d'étalonnage totalement automatique, il est possible d'exécuter approximativement 1400 expériences en 5 jours. Cela permet de considérer de 50 à 80 appels système (en fonction du nombre de paramètres à substituer) dans le cas d'un étalonnage portant sur 5 jours.

Postmark sollicite approximativement 27 appels système de la famille Windows et 16 appels système de la famille Linux.

La JVM active 75 appels système de la famille Windows et 35 appels système de la famille Linux.

Si tous les appels système sollicités par ces deux activités sont ciblés, les limites des cinq jours que nous nous sommes fixées sont largement respectées.

Le client TPC-C sollicite plus que 150 appels système, dont 132 avec paramètres. Avec ce grand nombre d'appels système, l'expérimentation pourrait nécessiter plus d'une semaine. Dans ce cas, et de façon à assurer la portabilité de l'étalon, nous nous sommes focalisés sur les composants fonctionnels de base de l'OS. Ce critère de sélection des appels système facilite la comparaison entre des OSs qui n'appartiennent pas à la même famille (Windows/Linux par exemple), et qui ont des API différentes. En effet, même si les différents OS ne comportent pas nécessairement les mêmes appels système, ils sont basés sur des composants fonctionnels comparables. Les composants fonctionnels de base que nous avons identifié dans un système d'exploitation à usage général et qui sont donc ciblés par notre étalon sont les suivants : processus et brins (*Processes and Threads*), le système de fichiers (*File Input/Output*), la gestion de mémoire (*Memory Management*) et la gestion de configuration (*Configuration Manager*). Cela nous a permis de réduire le nombre des appels système à corrompre, activés par TPC-C, de 132 à 28 appels système avec paramètres.

3.2.4 Environnement d'étalonnage et conduite d'expériences

Dans la mesure où les expériences peuvent conduire au blocage de l'OS, il est difficilement envisageable d'assurer la gestion de l'étalonnage uniquement avec la machine sur laquelle s'exécute l'OS cible. Il est impératif d'utiliser une autre machine pour assurer une gestion totalement automatique des expériences. En conséquence, deux machines sont nécessaires pour exécuter un étalon de sûreté de fonctionnement : 1) la machine cible qui accueille l'OS à étalonner et l'activité, et 2) la machine contrôleur qui assure la collecte de données, le

diagnostic et le redémarrage de la machine cible dans le cas de blocage ou de panique de l'OS.

La Figure 3-1 décrit les différents composants de l'environnement d'étalonnage des OSs. Ces composants doivent être répartis sur la machine cible où s'exécute le système d'exploitation à étalonner et sur la machine contrôleur, les deux machines étant reliées via un réseau.

L'expérimentation se base sur le système de gestion de l'étalonnage qui a trois principales composantes :

- l'intercepteur qui a pour rôle d'intercepter les appels système sollicités par l'activité;
- l'injecteur qui est chargé de l'injection de la faute en substituant le paramètre de l'appel système ciblé par une valeur prédéfinie;
- l'observateur qui doit observer le comportement du système et enregistrer les résultats des expériences.

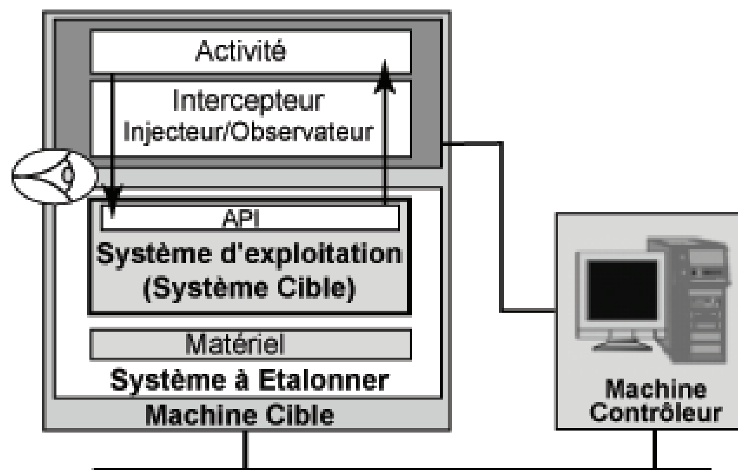


Figure 3-1 Environnement d'étalonnage

L'injecteur doit remplacer le paramètre de l'appel système retenu pour l'expérience par une valeur de substitution, permettant ainsi la simulation du comportement défectueux de l'application. Donc, les valeurs de substitution doivent être préparées avant le début des expériences. L'injecteur doit contenir la liste des paramètres à corrompre ainsi que les valeurs de substitution (incorrectes ou hors limite pour les données et adresse incorrecte pour les adresses), afin d'effectuer la substitution.

L'observateur a pour rôle de recueillir les données d'une expérience et de les stocker dans des fichiers de résultats qui seront utilisés ensuite dans la phase de traitement. En fait, il faut que l'observateur soit réparti sur les machines cible et contrôleur. Sur la machine cible, l'observateur enregistre les observations correspondant aux temps d'exécution de l'appel

système, la réaction de l'OS (retour d'un code d'erreur ou d'une exception) et le temps d'exécution de l'activité. La partie de l'observateur qui est sur la machine contrôleur détermine l'état de blocage ou de panique de l'OS ainsi que les différents états de l'activité, et enregistre le temps de redémarrage de la machine cible.

Les observations à relever et enregistrer au cours de chaque expérience sont illustrées par la Figure 3-2.

Au début de chaque expérience, la machine cible doit enregistrer l'instant de démarrage de l'expérience (tExpDeb) et envoyer cette valeur à la machine contrôleur, en lui notifiant le démarrage de l'expérience. À cet instant, la machine contrôleur doit armer un chien de garde. Ensuite, après le démarrage de l'activité sur la machine cible, l'observateur effectue une série d'enregistrements : instant de démarrage de l'activité (tAcDeb), noms des appels système activés et temps de réponse associés.

Durant une expérience, l'injecteur vérifie à chaque interception d'un appel système s'il s'agit de l'appel système à corrompre. Si tel est le cas, la valeur du paramètre doit être substituée avant de reprendre le flot d'exécution de l'activité avec un paramètre corrompu. Sinon, le flot d'exécution normal est repris. L'instant de reprise du flot (tContinue) est aussi enregistré. Lors du traitement des résultats, le temps de réaction de l'OS (TEr, TXp ou TNS) est calculé comme la différence entre tReponse (instant de fin d'exécution de l'appel système corrompu) et tContinue.

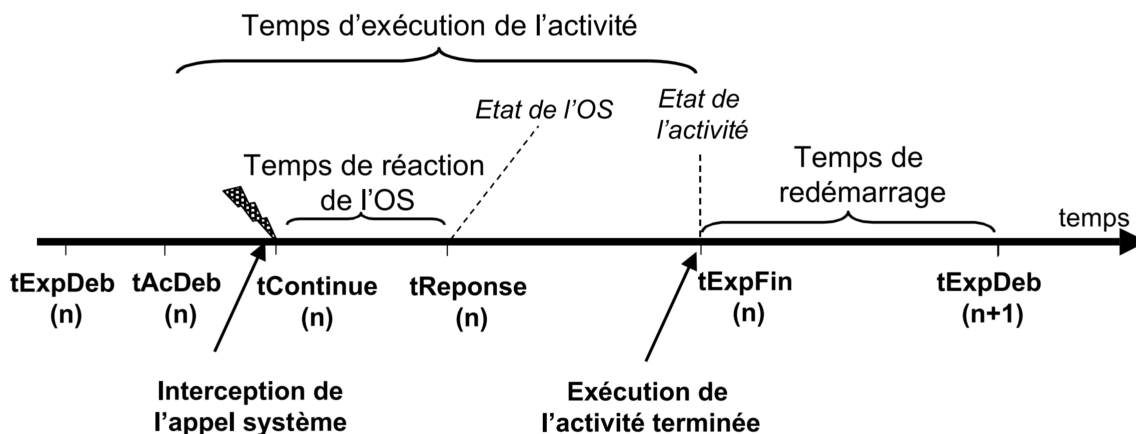


Figure 3-2 Séquence d'exécution de l'étalon et mesures temporelles (cas d'exécution complète de l'activité)

Ces enregistrements permettent d'avoir une trace d'exécution de l'activité au cours d'une expérience. Cette trace permet de caractériser le comportement de l'OS vis-à-vis de l'activité (code d'erreur retourné, type d'exception levée, temps de réaction de l'OS). La trace doit être enregistrée dans un fichier et envoyée à la machine contrôleur au début de l'expérience suivante, pour la préserver d'éventuels dommages et suppressions au cours e futurs dues aux expériences suivantes.

À la fin de l'exécution de l'activité, la machine cible doit notifier à la machine contrôleur la fin de l'exécution ainsi que l'instant de fin d'expérience (tExpFin) en envoyant un signal de fin.

Quand l'exécution de l'activité n'arrive pas à terme (par exemple, dans le cas d'un blocage de l'activité ou de l'OS), l'instant de fin d'expérience (tExpFin) est dicté par la valeur du chien de garde qui, sur la machine contrôleur, doit surveiller l'exécution de l'activité comme le montre la Figure 3-3. La définition de la valeur du chien de garde dépend du temps moyen nécessaire à l'OS pour exécuter l'activité en absence de fautes. Cette valeur est spécifiée comme étant quatre fois ce temps moyen d'exécution. Cette valeur est déterminée ainsi car parfois la durée d'exécution de l'activité en présence de fautes peut doubler par rapport à sa valeur en absence de fautes. La machine contrôleur doit détecter s'il y a eu un arrêt inopiné, un blocage de l'activité ou un blocage de l'OS. Dans le cas d'un blocage de l'OS, la machine contrôleur doit ordonner le redémarrage de la machine cible via une réinitialisation matérielle. En revanche, si le diagnostic révèle un abandon/blocage de l'activitié, la machine contrôle doit ordonner un redémarrage par logiciel de la machine cible.

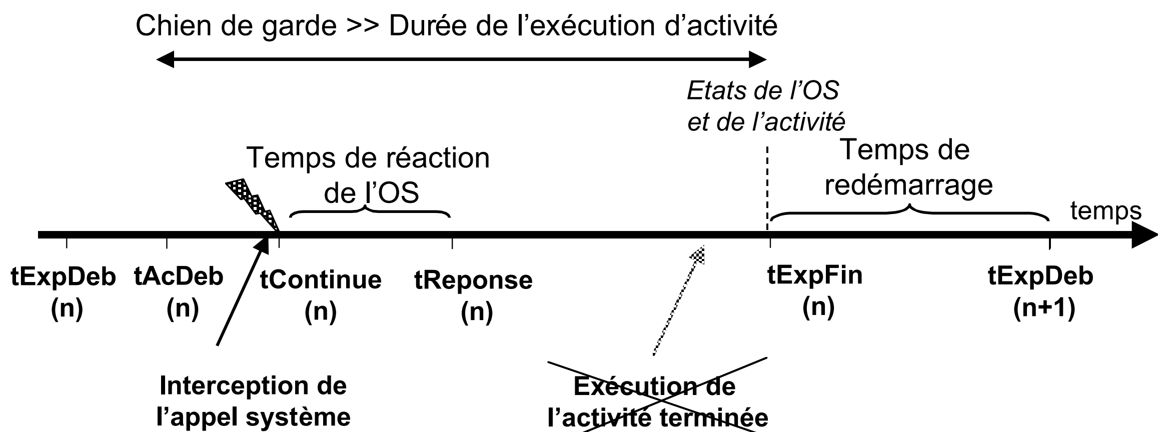


Figure 3-3 : Séquence d'exécution de l'étalon et mesures temporelles (cas d'abandon/blocage de l'activité)

La durée de redémarrage de la machine cible de l'expérience n (correspondant en fait à celle de l'OS) est déterminée par l'intervalle de temps entre l'instant tExpFin de l'expérience n et l'instant tExpDeb de l'expérience n+1, compte tenu du fait qu'une nouvelle expérience démarre automatiquement dès que l'OS a fini de redémarrer.

La dernière étape de l'étalonnage de sûreté de fonctionnement des OSs correspond au traitement des informations fournies lors des expériences. À la fin de l'étape d'expérimentation, tous les relevés des expériences sont enregistrés dans des fichiers sous forme de données brutes. Trois types de fichiers sont à différencier : les fichiers générés par l'outil d'interception (la trace), ceux générés par l'observateur installé sur la machine cible, et les fichiers contenant les informations récoltées par la machine contrôleur. Tous ces

fichiers doivent être rassemblés sur la machine contrôleur où leur traitement fournit les mesures finales.

Enfin, nous notons que l'étape d'expérimentation doit être entièrement automatisée. En effet, l'utilisateur de l'étalon doit pouvoir lancer les expériences au début et relever les résultats finaux à la fin de l'exécution de l'étalonnage sans avoir à intervenir entre ces deux moments.

3.3 Validation des propriétés par construction

Lors de la définition des étalons présentés dans ce chapitre, nous avons tenu compte des propriétés que l'étalon doit satisfaire. Dans ce qui suit, nous explorons quelques unes des propriétés définies dans le chapitre 2, et nous analysons comment elles ont été prises en compte dans la spécification. En revanche, d'autres propriétés comme la représentativité et la portabilité nécessitent également une validation expérimentale. Elles seront revues dans les chapitres 4 et 5.

3.3.1 La reproductibilité

La reproductibilité est la propriété qui garantit l'obtention de résultats statistiquement équivalents par des étalons développés par différentes personnes à partir de la spécification présentée dans ce chapitre. Nous montrons dans ce paragraphe la reproductibilité des étalons proposés à partir des spécifications fournies.

Pour étalonner la sûreté de fonctionnement des systèmes d'exploitation, nous avons adopté une approche expérimentale. Ainsi, l'étalonnage est composé d'une série d'expériences. Chacune de ces expériences commence après le redémarrage du système à étalonner, et ceci pour garantir un état de départ du système commun à toutes les expériences. Les expériences sont indépendantes les unes des autres, et l'ordre selon lequel les expériences sont menées n'est pas important.

Les activités utilisées par les trois étalons spécifiés sont disponibles sur le marché, prêtes à être utilisées comme la *JVM*, ou sous format de spécifications (*TPC*-C) ou de code source prêt à être compilé et exécuté (*Postmark*). Par conséquent, la reproductibilité des activités est garantie.

En ce qui concerne la corruption de paramètres, nous avons proposé d'intercepter tous les appels système activés pour *Postmark* et *JVM*, ou bien cibler des composants fonctionnels de l'OS pour *TPC*-C. Par conséquent, celui qui veut développer un étalon connaît exactement les appels système et les composants qu'il doit cibler. La technique de corruption des paramètres utilisée est la substitution sélective précisée dans la spécification des étalons (cf. section 3.2.3.1).

3.3.2 La représentativité

Les spécifications de trois étalons permettent de satisfaire les représentativités de mesures et de l'activité. Dans ce paragraphe, nous discutons de la représentativité des mesures

fournies et des activités utilisées dans les trois étalons présentés dans ce chapitre. La représentativité de l'ensemble de fautes sera discutée dans le chapitre 4.

Les trois principales mesures fournies par notre étalon de sûreté de fonctionnement proposé sont d'une grande importance pour les concepteurs des systèmes informatiques intégrant des OSs à usage général, dans la mesure où elles lui permettent de sélectionner l'OS le plus adapté à ses besoins (en termes de sûreté de fonctionnement). Ces mesures couvrent la robustesse et la performance en présence des fautes, et peuvent également être affinées par des mesures complémentaires pour mieux comprendre le comportement du système.

Les activités sélectionnées pour solliciter les appels système des OSs ciblés sont très connues et utilisées dans le domaine d'étalonnage de performance. Le client TPC-C est très répandu pour l'étalonnage de performance des systèmes transactionnels ; la principale raison de l'avoir utilisée est qu'il correspond à l'activité retenue par les autres partenaires de DBench. Le client TPC-C simule un client envoyant des requêtes à une base de données. Par ailleurs, nous avons considéré d'autres activités pour valider les résultats obtenus et comparer des systèmes d'exploitation appartenant à des familles différentes des OSs ; spécifiquement, nous avons utilisé la machine virtuelle Java comme activité réelle et l'activité réaliste de l'étalon de performance Postmark. Nous rappelons toutefois que la sélection de l'activité n'affecte pas les concepts et les spécifications des étalons présentés dans ce chapitre.

3.3.3 La portabilité

Nous vérifions dans ce paragraphe la portabilité de la spécification des trois étalons. Cette portabilité concerne les activités utilisées et les fautes appliquées.

En ce qui concerne les activités utilisées, nous avons choisi des activités portables par nature. Ces activités sont utilisées par des étalons de performance bien connus qui ont montré leur portabilité (TPC-C, Postmark), ou correspondent à des applications portables par définition (JVM).

En ce qui concerne l'ensemble des fautes, le système cible est considéré comme étant l'ensemble de ses composants fonctionnels qu'il fournit à l'applicatif, et sur cette base, son API est définie. Idéalement, il faut intercepter tous les appels système sollicités par l'activité utilisée. Les appels système à corrompre appartenant à ces composants fonctionnels ne sont pas définis par leur nom, parce que les systèmes d'exploitation de différentes familles n'ont pas nécessairement les mêmes appels système (comme le cas de l'API Win32 de la famille Windows et de l'API Posix de la famille UNIX). L'ensemble de services fournis à l'applicatif est commun à tous les systèmes d'exploitation, ce qui assure la portabilité des spécifications sur les différents systèmes d'exploitation.

D'un autre côté, la spécification de l'ensemble de valeurs de substitution est portable sur les différents OSs par ce que cet ensemble est défini par rapport à l'ensemble d'appels système à corrompre et non pas au niveau de chaque appel système . En fait, même s'ils utilisent des fonctions programmées en assembleur, les appels système des systèmes d'exploitation sont développés en langage C. Ainsi, l'ensemble de type de données de base est commun à tous

les systèmes d'exploitation, et le principe d'héritage des valeurs de substitution de ces types de base ne change pas.

Par conséquent, les spécifications des différents profils d'exécution sont facilement portables sur les systèmes des différentes familles d'OS. En revanche, la portabilité des implémentations de ces prototypes n'est jamais garantie. Cette propriété sera revisitée dans le chapitre 4 pour discuter de la portabilité des prototypes implémentés.

3.3.4 La non-intrusivité

L'activité sollicite le système cible à travers des appels système fournis par l'API. La technique de substitution de paramètres adoptée consiste à intercepter ces appels système activés et à corrompre leurs paramètres par des valeurs de substitution avant leur traitement par l'OS. Les valeurs de substitution doivent être fournies par un module appelé par l'injecteur de fautes. Ce module doit être installé à l'extérieur du système cible. Ainsi, aucune modification n'est introduite sur le système cible pour l'injection de fautes.

Les mécanismes d'observation qui doivent être installés sur la machine cible sont appelés après la fin d'exécution de l'appel système corrompu. Ces mécanismes se trouvent à l'extérieur des frontières du système cible. En effet, les interfaces des systèmes d'exploitation implémentent, sous forme d'appels système, des méthodes d'observation qui peuvent être appelées sans avoir besoin d'implémenter ces mécanismes dans le système cible.

En conclusion, la substitution des valeurs des paramètres est insérée avant l'exécution de l'appel système à l'extérieur de l'API du système cible. l'observateur qui intervient à la fin de l'exécution de l'appel système se trouve lui aussi à l'extérieur du système cible. Par conséquent, aucun de ces deux composants n'est introduit dans le système cible.

3.3.5 La mise à l'échelle

La mise à l'échelle d'un étalon de sûreté de fonctionnement d'OSs de la même famille ne pose pas un réel problème, parce que le nombre d'appels système sollicité pour chacun d'eux est comparable. En fait, le même nombre d'appels système est sollicité par Windows NT et de Windows 2000, alors que seulement deux appels systèmes supplémentaires sont sollicités par Windows XP. La spécification de DBench-OS-TPCC n'a pas été modifiée, et nous l'avons mis à l'échelle en tenant compte de ces deux appels système supplémentaires.

3.4 Conclusion

Basés sur le cadre d'étalonnage de sûreté de fonctionnement développé dans le chapitre précédent, nous avons présenté dans ce chapitre la spécification de trois étalons de sûreté de fonctionnement destinés à caractériser la sûreté de fonctionnements des systèmes d'exploitation en présence de fautes. Ils permettent d'analyser la robustesse des systèmes d'exploitation vis-à-vis des appels système erronés provenant de la couche applicative. Ces trois étalons sont basés sur trois activités différentes utilisées pour solliciter les services du système cible : l'activité de l'étalon de performance de systèmes transactionnels TPC-C,

l'activité de l'étalon de performance *Postmark*, et la machine virtuelle Java. Ces trois étalons sont respectivement appelés DBench-OS-TPCC, DBench-OS-Postmark et DBench-OS-JVM. Ils sont essentiellement destinés à l'utilisateur final d'une OS qui n'a pas forcément une profonde connaissance dans l'OS. L'OS est considéré comme boîte noire ; la seule information nécessaire correspond à la description de l'OS en termes de fonctions et d'appels système.

Nous avons spécifié deux ensembles de mesures que ces étalons peuvent fournir : le premier (mesures de base) permet de caractériser le système d'exploitation proprement dit, tandis que l'autre contient des mesures complémentaires destinées à affiner les informations fournies par les mesures de base. Pour chacun de ces deux ensembles, nous avons distingué deux classes de mesures : 1) les mesures de robustesse, et 2) les mesures temporelles en présence de fautes, comme les temps moyens d'exécution des appels système et les temps moyens de redémarrage du système à étalonner.

Notons que cette liste de mesures présentées dans ce chapitre n'est pas exhaustive. D'autres mesures de sûreté de fonctionnement ou de performance en présence de fautes peuvent faire partie de l'ensemble de mesures que les étalons de sûreté de fonctionnement des systèmes d'exploitation peuvent fournir. Par exemple, nous citons le risque de propagation d'erreurs, à travers le système d'exploitation, entre les différents processus. Il s'agit d'évaluer l'impact d'un processus erroné sur le comportement d'un autre processus actif sur le même système d'exploitation. Dans ce cas, d'autres mesures peuvent être considérées telles que la probabilité et le temps de propagation de cette erreur.

Dans le cas d'un étalon de sûreté de fonctionnement pour des systèmes d'exploitation, le profil d'exécution inclut, en plus de l'activité, un ensemble fautes qui sont appliquées aux paramètres des appels système. Au travers de cet ensemble de fautes, le but est de modifier les différents appels système soumis à l'OS pour simuler les valeurs erronées de paramètres qu'une application défaillante pourrait communiquer à l'OS. L'expérimentation se base sur un système de gestion d'étalonnage composé d'un intercepteur des appels système, d'un injecteur de fautes et d'un observateur. L'ensemble des fautes injectées consiste à substituer les valeurs correctes des paramètres des appels système par des valeurs incorrectes. Idéalement, tous les appels système sollicités par l'activité doivent être corrompus comme dans les cas de Postmark et JVM. Parfois, le nombre d'appels système à corrompre est très grand ce qui rend impossible de réaliser l'étalonnage dans des délais acceptables comme dans le cas de l'activité TPC-C qui active 132 appels système avec paramètres. Dans ce cas, nous avons proposé d'intercepter les appels système appartenant aux composants fonctionnels les plus utilisés. Les observations relevées sur l'OS lors de l'application du profil d'exécution permettent de calculer les mesures souhaitées de l'étalon.

À partir de cette spécification, nous avons pu valider certaines propriétés de l'étalon de sûreté de fonctionnement présentées dans le deuxième chapitre. Ces propriétés correspondent à la reproductibilité, la représentativité des mesures et des activités, la portabilité et la non-intrusivité. Le reste de l'ensemble de propriétés ne peut être validé qu'expérimentalement.

Les étalons présentés dans ce chapitre sont destinés à caractériser les différentes familles de systèmes d'exploitation à usage général. DBench-OS-TPCC est mis en œuvre pour les OSs de la famille Windows uniquement. Cette mise en œuvre de l'étalon, que nous appelons prototype de l'étalon, fait l'objet du quatrième chapitre. Nous rappelons que l'activité de TPC-C a été utilisée pour assurer la cohérence entre les différents partenaires du projet DBench. Les deux autres étalons DBench-OS-Postmak et DBench-OS-JVM ciblent les OSs des deux familles Windows et Linux et feront l'objet du cinquième chapitre.

Chapitre 4 Étalonnage de la sûreté de fonctionnement des systèmes Windows

4.1 Introduction

Le but de ce chapitre est de mettre en œuvre l'étalon DBench-OS-TPCC défini dans le chapitre 3. Cette mise en œuvre est utilisée pour caractériser la sûreté de fonctionnement de trois systèmes d'exploitation de la famille Windows : Windows NT4 Workstation, Windows 2000 Professional et Windows XP.

C'est en 1985 que la première version de Windows (Windows 1.0) fit son apparition. Cette version n'a eu pratiquement aucun succès auprès du public. Ensuite, plusieurs versions de Windows ont été mises sur le marché sans un grand succès. En 1995, la compagnie Microsoft lance Windows 95 qui a connu un énorme succès. Ensuite, une version beaucoup plus stable de Windows 95, Windows NT 4.0, a été lancée l'année suivante. En tout, six *service packs* ont été publiées pour NT4 par Microsoft pour l'améliorer. Depuis, plusieurs versions des systèmes d'exploitation de la famille Windows ont vu le jour. Toutes ces versions de Windows ont été utilisées lors de la conception de systèmes critiques, sans avoir énormément d'informations sur leur sûreté de fonctionnement. En effet, peu d'études et de travaux de recherche ont été effectués sur la caractérisation de la sûreté de fonctionnement des systèmes d'exploitation Windows.

Dans un premier temps, nous présentons la mise en œuvre de la plateforme expérimentale utilisée pour étalonner ces trois systèmes Windows. En particulier, nous présentons l'outil utilisé pour intercepter les appels système.

Nous présentons dans le paragraphe 4.3 les résultats relatifs aux mesures de base de sûreté de fonctionnement de notre étalon : robustesse de chacun de ces trois systèmes, temps de réaction et de redémarrage en présence de fautes.

Ces mesures fournies sont ensuite affinées dans le paragraphe 4.4 pour analyser plus en détail le comportement de ces systèmes en présence de fautes. Ceci nous permettra de mieux comprendre certaines mesures de l'étalon fournies dans la paragraphe 4.3 .

Dans le dernier paragraphe, nous analysons et vérifions les propriétés de l'étalon de sûreté de fonctionnement (cf chapitre 2) qui ne peuvent être vérifiées que par expérimentation dans le cas d'étalon pour la même famille d'OSs. Ces propriétés concernent la répétitivité, la représentativité et le coût de l'étalonnage. Nous mettons l'accent sur l'importante

propriété de représentativité de l'ensemble de fautes utilisé lors de l'étalonnage de sûreté de fonctionnement.

4.2 Mise en œuvre

Comme nous l'avons expliqué dans le premier chapitre (cf. section 1.4), Windows est un système de type *temps partagé* qui ne possède pas un noyau monolithique ; en effet, les composants fonctionnels de Windows sont placés juste au dessus de son micro noyau. Pour accéder aux services des composants fonctionnels dans une architecture Windows, les applicatifs effectuent des appels aux services de l'OS au travers d'un ensemble de bibliothèques dynamiques de base, connues dans le monde Windows sous le nom de DLLs (*Dynamic Link Libraries*). L'ensemble des bibliothèques DLLs (tels que kernel32.dll, Advapi32.dll, User32.dll, et Gdi.dll) implémente les fonctions de l'API Win32. Même si Windows est conçu pour supporter plusieurs interfaces de programmation (OS/2, POSIX), l'API Win32 est l'interface principale et préférée. De plus, Windows ne peut pas fonctionner sans le sous-système Win32 [Solomon 2000]. Par conséquent, c'est cette interface qui a été considérée et sur laquelle ont été appliqués les différents étalons. Par la suite, ses fonctions seront considérées comme les appels système de Windows.

Nous avons développé un prototype d'étalon de sûreté de fonctionnement, basé sur la spécification de l'étalon DBench-OS-TPCC présenté dans le chapitre 3, pour caractériser la sûreté de fonctionnement de trois systèmes d'exploitation de la famille Windows : Windows NT4, 2000 et XP [Kalakech 2004b]. Pour pouvoir réaliser des comparaisons entre les différents OSs, notamment temporelles, nous avons utilisé la même plate-forme matérielle sur laquelle tous les OSs ciblés sont exécutés successivement. Une seule implémentation a été réalisée pour chaque série d'expériences. Cette plate-forme est composée d'un processeur Pentium III 800 MHz, d'une mémoire de 512 MB et d'un disque dur de 18 GB, ULTRA DMA 160 SCSI. La machine contrôleur correspond à une station Sun qui permet de surveiller plusieurs machines cibles en parallèle. Les deux machines contrôleur et cible communiquent entre elles grâce à des scripts *shell* et *Expect* via le réseau Ethernet qui les relie.

En absence de fautes, la durée d'exécution du client TPC-C s'élève à une minute. Nous avons défini une durée de cinq minutes pour le chien de garde (voir Figure 3.3), au delà de laquelle l'activité est considérée dans un état de blocage ou d'abandon. Au cours de son exécution, le client TPC-C active 132 appels système avec paramètres, dont seulement 28 appartiennent aux composants fonctionnels définis lors de la spécification de cet étalon (processus et brins, système de fichiers, gestion de mémoire et gestion de configuration). Pour ces appels système, 75 paramètres ont été corrompus de plusieurs façons, conduisant à la réalisation de 552 expériences pour chacun des trois systèmes ciblés.

La première étape de la mise en œuvre consiste à préparer pour chaque appel système à corrompre la liste des valeurs de substitution.

Pour intercepter les appels système Win32, nous nous sommes basés sur l'outil *Detours* développé par Microsoft [Hunt 1999]. Cet outil est destiné à générer une trace d'exécution

d'un programme en interceptant n'importe quel appel système Win32 sur une plate-forme matérielle x86. Cet outil a été modifié pour pouvoir substituer de manière contrôlée les valeurs des paramètres des appels système. Par ailleurs, nous avons aussi étendu les fonctionnalités de cet outil pour nous permettre d'observer la réaction de l'OS suite à l'injection d'une faute et de récupérer les informations nécessaires, notamment des informations temporelles et les retours de codes d'erreur et d'exceptions. En fait, les interfaces des systèmes d'exploitation implémentent des méthodes sous forme d'appels système, tels que `GetExceptionCode()` et `GetLastErrorCode()`, qui peuvent être appelées pour observer si un code d'erreur ou une exception ont été générés. Pour calculer le temps de réaction de l'OS, on calcule la différence de deux compteurs placés avant et après l'exécution de l'appel système à corrompre. Après l'interception de l'appel système, l'outil développé doit remplacer la valeur de l'un de ses paramètres par une autre valeur prédéfinie.

L'outil *Detours* original continent 30000 lignes de codes C approximativement. Les deux modules d'injection et d'observation que nous avons ajoutés correspondent approximativement à 3000 et 15000 lignes de code respectivement. La mise en œuvre, sur la machine contrôleur, des mécanismes de supervision de la machine cible a nécessité 300 lignes de code. Un exemple d'une fonction de *Detours* qui permet d'intercepter un appel système, injecter la faute et observer la réaction de l'OS est présenté dans l'annexe 1.

À la fin des expériences, les différents fichiers contenant les résultats de chacune des expériences sont traités sur la machine contrôleur par un simple programme qui permet de fournir des données prêtes à être exploitées dans une base de données. Ce programme dépend de la famille d'OSs ciblée, parce que chacune d'entre elles peut avoir une trace d'exécution spécifique. Nous avons développé un squellette d'une base de données commune à tous les étalons proposés. Elle reçoit les données traîtées durant la première phase et fournit les mesures désirées des étalons.

4.3 Mesures de l'étalon

Dans les trois sections suivantes, nous présentons les mesures de l'étalon DBench-OS-TPCC caractérisant le comportement global des trois OSs ciblés en considérant les 28 appels système des composants fonctionnels identifiés dans les spécifications de cet étalon (cf. section 3.2.3.2).

4.3.1 Robustesse de l'OS

La robustesse (cf. section 3.2.1) des trois OSs est donnée par la Figure 4-1. Pour les trois OS, aucun état de panique ou de blocage n'a été constaté. Des exceptions ont été notifiées dans 11,4% à 12% des cas, alors que le nombre d'expériences avec un retour de code d'erreur varie entre 31,2% et 34,1%. Plus de la moitié des expériences conduisent à un état de non signalement. La Figure 4-1 montre un comportement similaire des trois OS vis-à-vis de la robustesse, étant donné que la plus grande différence, qui a été constatée pour le cas de non signalement, est inférieure à 3%. Les trois OSs sont donc équivalents de point de vue robustesse.

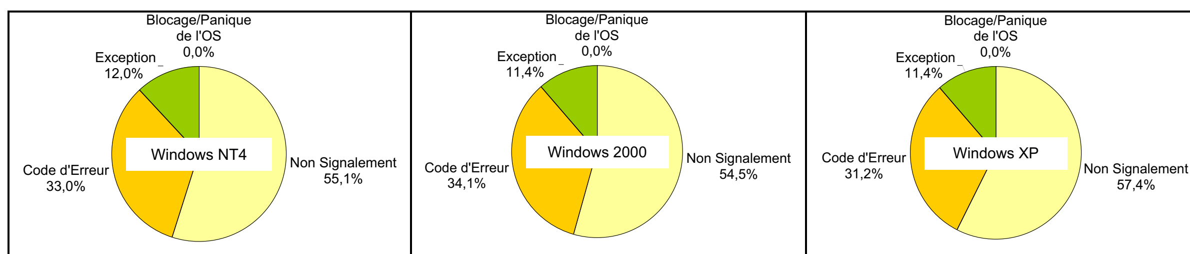


Figure 4-1 : Robustesse des trois OSs

4.3.2 Temps de réaction de l'OS

Le temps de réaction de l'OS en absence de fautes, τexec, est évalué comme la moyenne des temps de réaction des 28 appels système retenus. Le Tableau 4-1 montre que, en absence de fautes, les trois OS ont des temps de réaction différents : le plus rapide étant Windows XP, suivi par Windows NT4, et enfin, Windows 2000.

Le temps de réaction Texec correspond à la moyenne des temps observés, en présence de fautes, des 28 appels système corrompus. Le tableau montre que l'ordre de rapidité en présence de fautes est le même qu'en absence de fautes. Le temps de réaction le plus court est obtenu pour Windows XP avec une moyenne de 111 µs, tandis que le temps le plus long est observé pour Windows 2000 avec un temps moyen de réaction de 1782 µs. Pour Windows XP, ce temps est légèrement plus grand que le temps de réaction en l'absence de fautes, alors qu'il est, de manière significative, plus petit pour les deux autres OS. Ceci peut être expliqué par le fait que, dans à peu près 45 % des cas, l'OS détecte la faute injectée. Dans ces cas, il interrompt l'exécution de l'appel système et retourne un code d'erreur ou notifie une exception. L'écart type est, de manière significative, plus grand que la moyenne, pour les trois OS. Ceci montre la variabilité des temps de réaction. Les mesures complémentaires de l'étalon présentées dans le paragraphe 4.4.2 permettent d'expliquer et mieux comprendre cette variabilité des temps de réaction.

Tableau 4-1 : Temps de réaction de l'OS

	Windows NT4		*Windows 2000*		*Windows XP*	
	Moyenne	*Ecart type*	*Moyenne*	*Ecart type*	*Moyenne*	*Ecart type*
τexec	*344* µs		*1782* µs		*111* µs	
Texec	*128* µs	*230* µs	*1241* µs	*3359* µs	*114* µs	*176* µs

4.3.3 Temps de redémarrage du système

Les temps de redémarrage des systèmes, en absence et en présence de fautes (τres et *Tres* respectivement), sont fournis dans le Tableau 4-2. Ce dernier montre que le temps de redémarrage de Windows XP (en présence et en absence de fautes) est le plus court. Il correspond à 70 % de celui de Windows 2000, en absence de faute et à 73 % de ce temps en présence de fautes. Pour tous les systèmes ciblés, le temps de redémarrage en présence de fautes est supérieur de seulement quelques secondes à celui sans fautes.

Tableau 4-2 : Temps de redémarrage de l'OS

	Windows NT4		*Windows 2000*		*Windows XP*	
	Moyenne	*Ecart type*	*Moyenne*	*Ecart type*	*Moyenne*	*Ecart type*
τred	*92 s*		*105 s*		*74 s*	
Tred	*96 s*	*4 s*	*109 s*	*8 s*	*80 s*	*8 s*

4.3.4 Récapitulatif

Les résultats ci-dessus révèlent que Windows XP, Windows NT4 et Windows 2000 sont équivalents d'un point de vue de la robustesse, dans la mesure où aucune différence significative n'a été révélée en termes de pourcentages de différentes issues de l'OS après la corruption de ses paramètres. La différence entre les trois systèmes a été notée au niveau des mesures temporelles, où Windows XP obtient des temps de réaction et de redémarrage plus courts, en présence ainsi qu'en présence de fautes. A son tour, Windows NT4 atteint également des temps de réaction et de redémarrage plus courts que ceux de Windows 2000.

4.4 Mesures complémentaires et affinement

Nous montrons dans cette section comment l'ensemble des mesures de base de l'étalon peut être enrichi, en considérant les informations additionnelles que la spécification et les prototypes développés peuvent fournir. En fait, nous affinons les trois mesures globales en considérant l'ensemble des mesures complémentaires spécifiées dans le chapitre précédent.

4.4.1 Robustesse de l'OS

Les mesures globales de robustesse peuvent être complétées par les observations de l'état final de l'activité. Autrement dit, la robustesse de l'OS est raffinée en étudiant l'influence de l'OS sur l'exécution de l'activité.

Le prototype utilisé pour étalonner la sûreté de fonctionnement des systèmes Windows ne permet pas la distinction entre la terminaison correcte et la terminaison incorrecte de l'activité, ni la distinction entre le blocage et l'abandon de l'application. Ceci est dû à

l'indisponibilité du code source du client TPC-C. Par conséquent, il ne nous était pas possible d'ajouter à cette activité un quelconque moyen de diagnostic de l'état final de son activité. Dans ce qui suit, nous distinguons deux états de l'activité : état d'*exécution complète* et état *d'abandon* ou du *blocage* de l'activité.

Nous présentons dans le Tableau 4-3 la distribution des expériences réalisées sur les trois systèmes Windows par rapport aux états finaux considérés pour l'activité. Ce tableau montre que les trois systèmes ciblés sont équivalents avec des taux de terminaison de l'activité très proches (allant de 76,8% pour Windows XP jusqu'à 81,7% pour Windows NT4).

Tableau 4-3 : Distribution des expériences par rapport à l'état de l'activité

	Windows NT4	*Windows 2000*	*Windows XP*
Terminaison	*451(81,7%)*	*445(80,6%)*	*424(76,8%)*
Abandon/Blocage	*101 (18,3%)*	*107 (19,4%)*	*128 (23,2%)*

Le Tableau 4-4 illustre d'une façon plus détaillée les mesures combinées concernant les états finaux de l'activité et des systèmes d'exploitation ciblés. Il montre que :

- Dans le cas de retour de codes d'erreur, l'activité est en état d'*Abandon/Blocage* dans 25% (46 / 182) des cas pour Windows NT4 (respectivement 28% et 42% pour Windows 2000 et XP);
- Après la notification d'exceptions, l'activité est dans un état d'Abandon/Blocage dans 12% des cas des expériences établies sur Windows NT4 et 9% pour Windows 2000 et Windows XP. Nous notons qu'une seule exception (Exception_Access_Violation) a été notifiée par les trois systèmes d'exploitation.

Nous constatons que le taux de terminaison d'exécution de l'activité avec un code d'erreur retourné est nettement supérieur à celui correspondant au cas d'une exception notifiée. Ces résultats montrent que, même dans le cas de détection du paramètre corrompu par l'OS, l'application ne traite pas correctement le code retourné.

La distribution de l'état final de l'activité en cas de non signalement ($\mathbf{P_{NS}}$) est illustrée dans la dernière colonne du Tableau 4-4. Avec le prototype utilisé, $\mathbf{P_{NS}}$ est composée de deux éléments uniquement. Nous constatons que l'activité a abouti à des états d'Abandon/Blocage dans 16 % des cas de non signalement, et ceci pour les trois systèmes d'exploitation. Une fois de plus, ce résultat confirme l'équivalence des trois systèmes en termes de robustesse.

Tableau 4-4 : Les états combinés de l'OS et de l'activité

Windows NT4→ **↓Activité**	*Code d'erreur (182)*	*Exception (66)*	*Panique*	*Blocage*	*Non signalement (304)*
Terminaison	*136 (24,2%)*	*58 (10,3%)*		—	*257 (47,5%)*
Abandon/Blocage	*46 (8,2%)*	*8 (1,4%)*	*0*	*0*	*47 (8,4%)*

Windows 2000 → **↓Activité**	*Code d'erreur (188)*	*Exception (63)*	*Panique*	*Blocage*	*Non signalement (301)*
Terminaison	*136 (24,6%)*	*57 (10,3%)*	—	—	*252 (45,7%)*
Abandon/Blocage	*52 (9,4%)*	*6 (1,1%)*	*0*	*0*	*49 (8,9%)*

Windows XP → **↓Activité**	*Code d'erreur (172)*	*Exception (63)*	*Panique*	*Blocage*	*Non signalement (317)*
Terminaison	*99 (17,9%)*	*57 (10,3%)*	—	—	*268 (48,6%)*
Abandon/Blocage	*73 (13,2%)*	*6 (1,1%)*	*0*	*0*	*49 (8,9%)*

4.4.2 Temps de réaction de l'OS

Le temps de réaction de l'OS peut être raffiné en considérant les issues de l'OS après les corruptions de paramètres. Ainsi, les trois dernières lignes du Tableau 4-5 complètent les mesures déjà présentées dans le Tableau 4-1. Elles correspondent respectivement au :

- Temps moyen de retour code d'erreur (*TEr)* ;
- Temps moyen de notification d'exception (*TXp)* ;
- Temps moyen d'exécution des appels système dans les cas où l'OS n'a rien détecté (*TNS).*

Pour les trois systèmes d'exploitation ciblés, nous constatons que le temps moyen d'exécution des appels système qui ont retourné un code d'erreur est nettement inférieur à celui des appels système qui ont retourné des exceptions. A son tour, le temps moyen pour retourner des exceptions est inférieur à celui du cas de non signalement de l'OS. Ceci peut s'expliquer par le fait que les mécanismes de test des paramètres d'entrée sont exécutés avant le code des appels système, et par conséquent, l'appel système interrompt son exécution directement après la détection de l'anomalie. Par contre, l'exception est détectée par le matériel, et par conséquent, le temps de détection est plus important. Quant aux cas de non signalement, le paramètre corrompu n'a pas été détecté, et par conséquent, l'appel système essaie d'achever son exécution.

Tableau 4-5 : Temps détaillé de réaction de l'OS

	Windows NT4		Windows 2000		Windows XP	
	Moyenne	Ecart type	Moyenne	Ecart type	Moyenne	Ecart type
τexec	344 µs		1782 µs		111 µs	
Texec	128 µs	230 µs	1241 µs	3359 µs	114 µs	176 µs
TEr	17 µs	18 µs	22 µs	28 µs	23 µs	17 µs
TXp	86 µs	138 µs	973 µs	2978 µs	108 µs	162 µs
TNS	203 µs	281 µs	2013 µs	4147 µs	165 µs	204 µs

Pour les trois OSs ciblés, les temps moyens de retour de codes d'erreur sont comparables, avec un écart type assez faible. Ces temps de réaction (TEr) sont assez courts par rapport aux temps moyens de réaction des systèmes en absence ainsi qu'en présence de fautes. La Figure 4-2 illustre les temps moyens d'exécution des différents appels système qui ont généré un code d'erreur suite à la corruption de l'un de leurs paramètres. Cette figure montre que les temps d'exécution des différents appels système sont comparables pour les trois OSs. En plus, ils sont très proches du temps moyen général de réaction de l'OS en cas de retour de codes d'erreur (TEr).

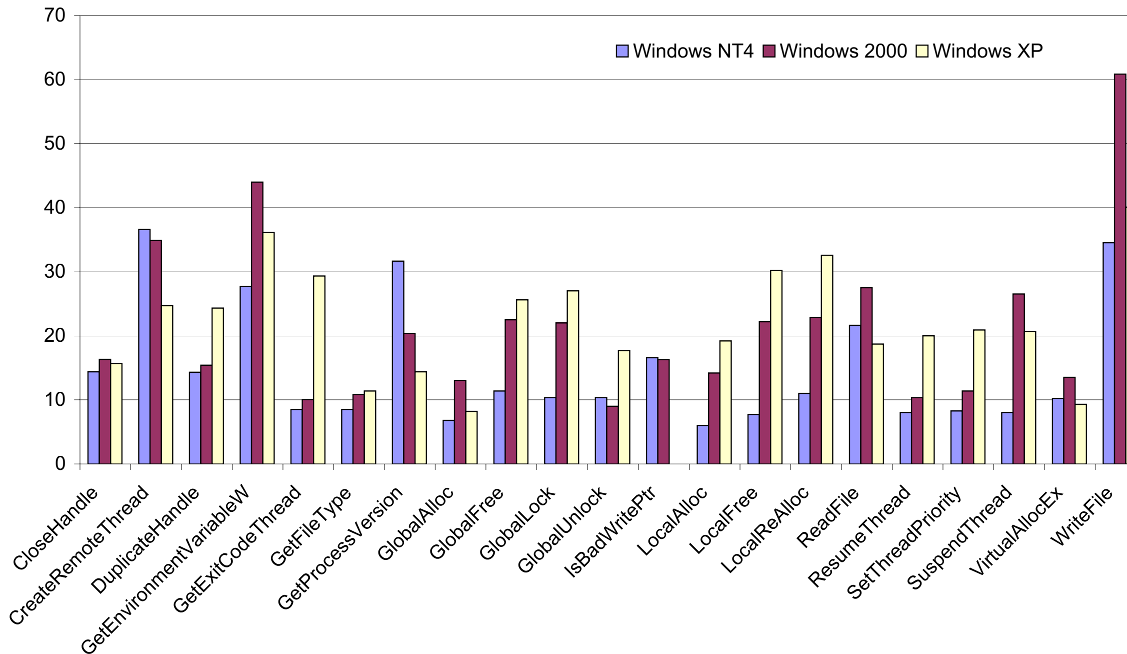


Figure 4-2 : Temps détaillé de réaction de l'OS, cas de retour de code d'erreur

Les temps nécessaires pour notifier des exceptions pour Windows NT4 et Windows XP restent comparables, tandis que celui de Windows 2000 leur est très supérieur. Nous constatons également que la moyenne des temps d'exécution des appels système en cas de non signalement est largement supérieure pour Windows 2000, alors que ces moyennes restent comparables pour les deux autres systèmes. En effet, nous trouvons un écart type très important pour les TEr et TNS de Windows 2000, ce qui suggère une large variation des temps d'exécution des appels système autour de ces moyennes. Les Figure 4-3 et Figure 4-4 confirment cette large variation. Elles permettent d'identifier les différents appels système qui ont retourné des codes d'erreur ou des exceptions. Ces figures donnent les temps de réaction de ces appels système en présence de fautes.

Dans la Figure 4-3, tous les appels système qui ont notifié des exceptions sont présentés pour les trois systèmes d'exploitation. La grande valeur de l'écart type de Windows 2000 est due au temps d'exécution de l'appel système `GetPrivateProfileStringA` qui a atteint une durée de 2480 µs alors que cette durée ne dépasse pas les 150 µs pour Windows NT4 et Windows XP.

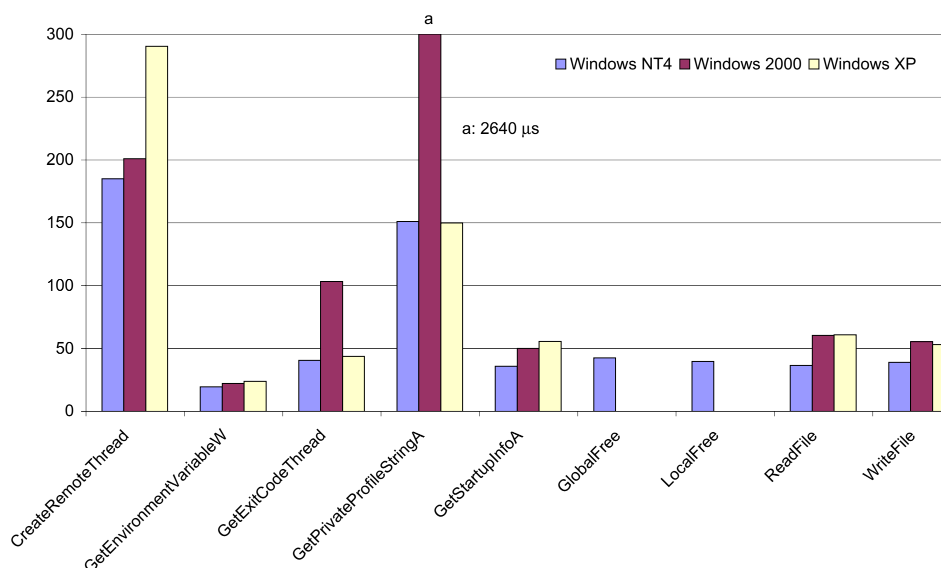


Figure 4-3 : Temps détaillé de réaction de l'OS, cas de notification d'une exception

La Figure 4-4 illustre tous les appels système donnant lieu à un non signalement. Sur cette figure, nous constatons que les durées d'exécution des appels système `GetPrivateProfileStringA` et `GetPrivateProfileIntA` ont été la cause directe de l'augmentation de la valeur de l'écart type du temps moyen de réaction de Windows 2000.

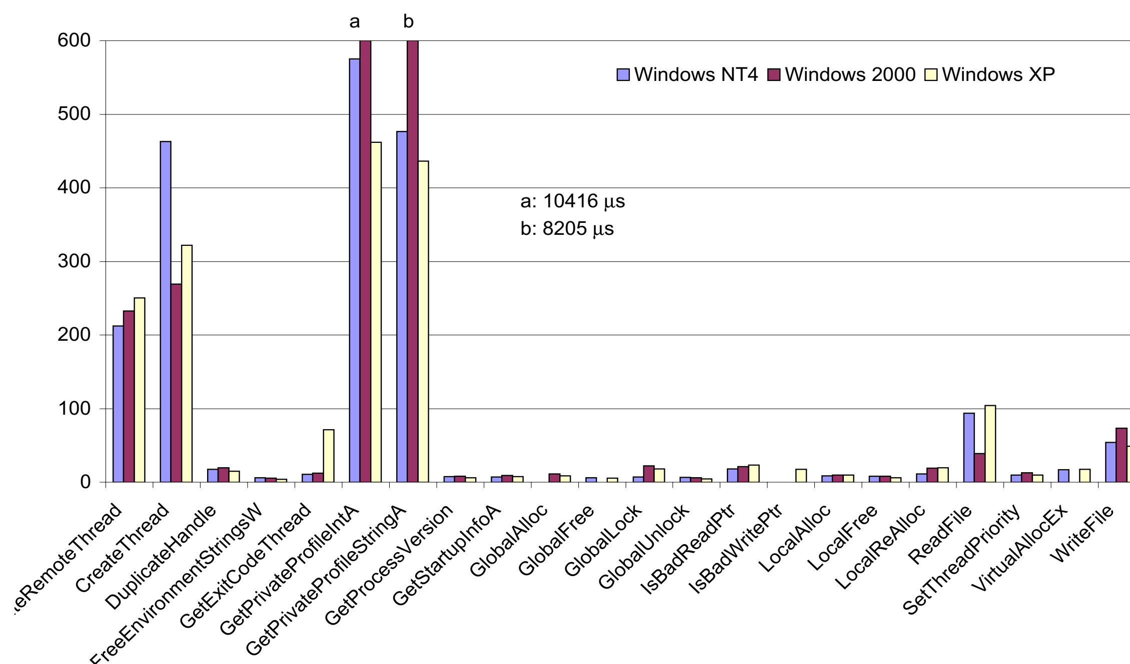


Figure 4-4 : Temps détaillé de réaction de l'OS, cas de Non Signalement

Les Figure 4-3 et Figure 4-4 montrent que les deux appels système `GetPrivateProfileStringA` et `GetPrivateProfileIntA` constituent des points faibles dans le système d'exploitation Windows 2000, et suggèrent qu'une analyse plus profonde serait nécessaire pour mieux comprendre leur fonctionnement lors de la corruption de leurs paramètres d'entrée. Toutefois, si nous excluons du calcul les durées d'exécution de ces deux appels système (Tableau 4-6), nous trouvons que les temps de réaction des systèmes sont considérablement réduits par rapport aux temps présentés dans le Tableau 4-5. Ces temps de réaction sont comparables pour les trois systèmes d'exploitation.

Tableau 4-6 : Temps détaillé de réaction de l'OS sans les appels système `GetPrivateProfileIntA` et `GetPrivateProfileStringA`

	Windows NT4		*Windows 2000*		*Windows XP*	
	Moyenne	*Ecart type*	*Moyenne*	*Ecart type*	*Moyenne*	*Ecart type*
τexec	*323* µs		*1054* µs		*60* µs	
Texec	*72* µs	*180* µs	*61* µs	*96* µs	*63* µs	*120* µs
TEr	*16* µs	*18* µs	*21* µs	*28* µs	*23* µs	*16* µs
TXp	*56* µs	*60* µs	*79* µs	*76* µs	*85* µs	*127* µs
TNS	*117* µs	*241* µs	*87* µs	*120* µs	*96* µs	*146* µs

Nous présentons enfin une comparaison entre les temps de réaction des appels système qui ont retourné des codes d'erreur ou des exceptions ou qui ont abouti à des cas de non signalement en présence de fautes pendant les expériences menées sur les trois OSs. Les temps moyens d'exécution de ces appels système pour les trois OSs sont présentés dans la Figure 4-5. Sur cette figure, nous constatons que pour chacun des trois OSs considérés, les temps nécessaires aux appels système pour retourner des codes d'erreur sont généralement plus petits que les temps de retour d'exceptions ou les temps d'exécution dans le cas de non signalement. À leurs tours, les temps de notification d'exceptions sont généralement plus petits que les temps d'exécution en cas de non signalement.

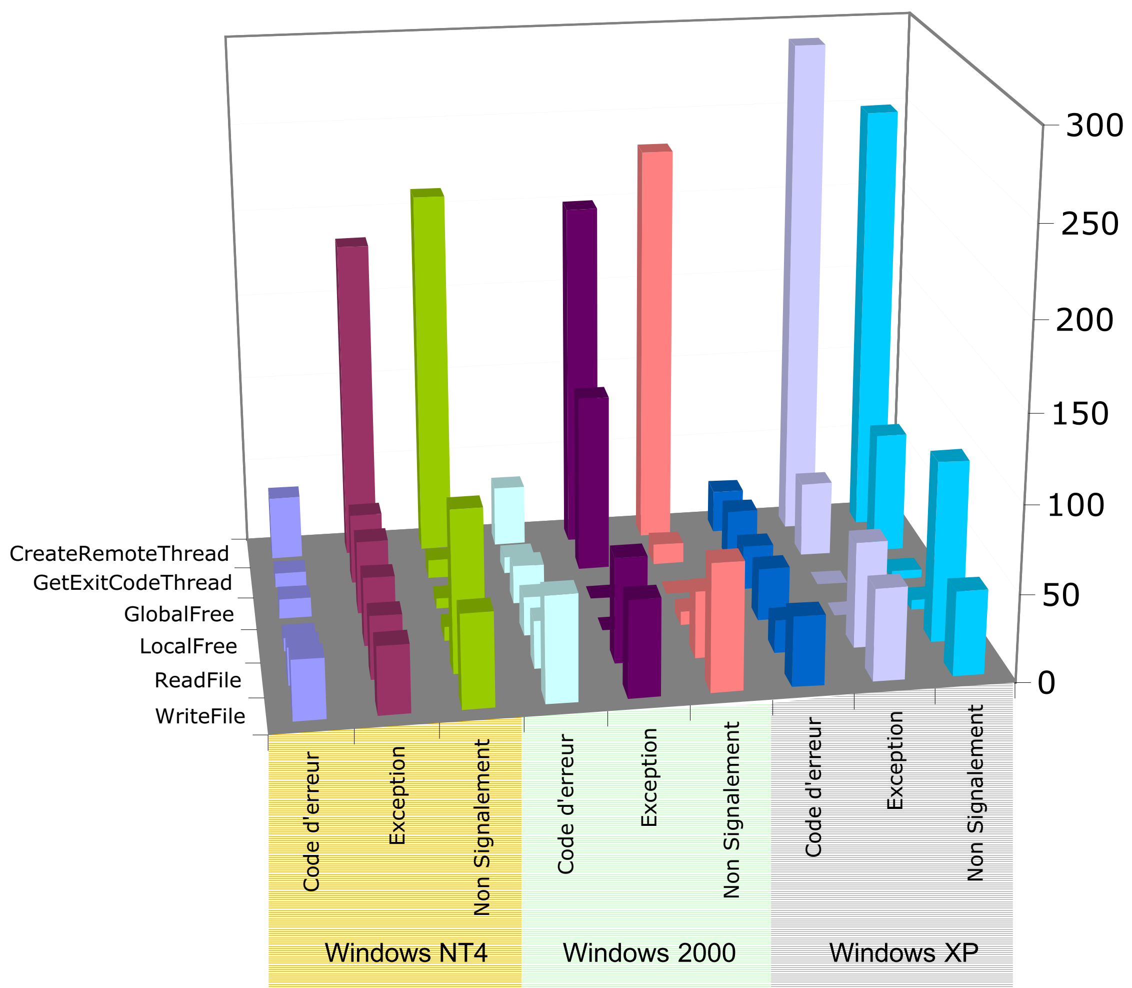


Figure 4-5 : Temps de réaction détaillé des appels systèmes pour les trois OSs, cas de notification de code d'erreur, d'exception ou cas de non signalement

4.4.3 Temps de redémarrage

Bien que le temps de redémarrage en présence de fautes ne soit pas trop différent de celui en absence de fautes, l'analyse détaillée de l'ensemble des données collectées pour les trois OS fait apparaître une corrélation entre le temps de redémarrage et l'état de l'activité après l'exécution d'un appel système erroné. Quand l'activité arrive à son terme, la moyenne du temps de redémarrage est très proche de celle observée en absence de fautes. Au contraire, lorsque l'activité est dans l'état d'*Abandon* ou de *Blocage*, le temps de redémarrage est nettement supérieur.

Nous présentons dans les figures 4-5 à 4-7 les détails des temps de redémarrage en présence de fautes pour toutes les expériences réalisées pour les trois OSs. Sur ces figures, nous distinguons clairement deux valeurs distinctes représentées par deux lignes : la ligne inférieure correspond aux temps de redémarrage dans les cas de terminaison de l'activité, tandis que la ligne supérieure correspond aux temps de redémarrage des expériences dans le cas où l'activité a fini par être bloquée.

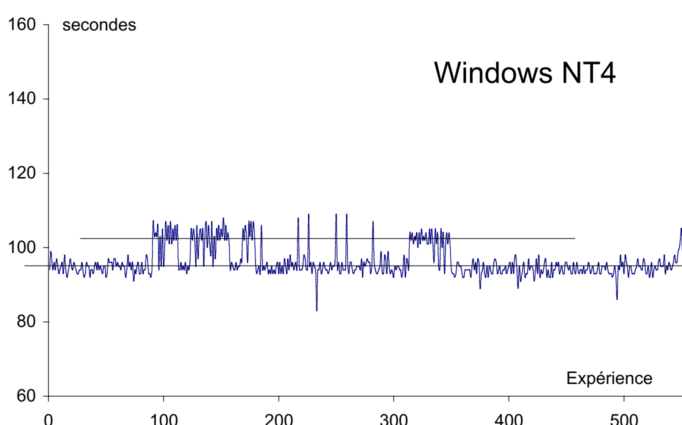


Figure 4-6 : Détails des temps de redémarrage de Windows NT4

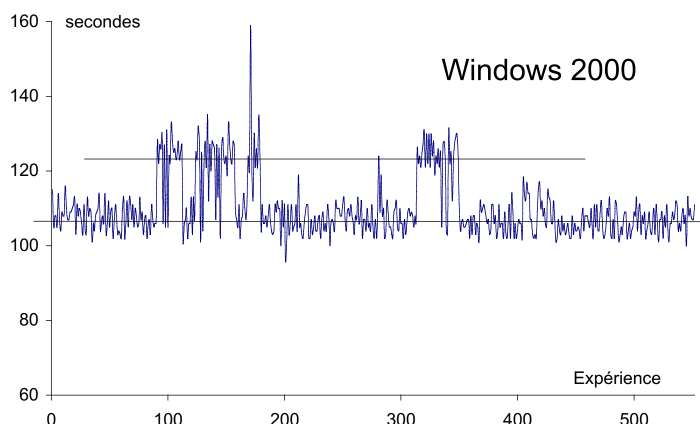


Figure 4-7 : Détails des temps de redémarrage de Windows 2000

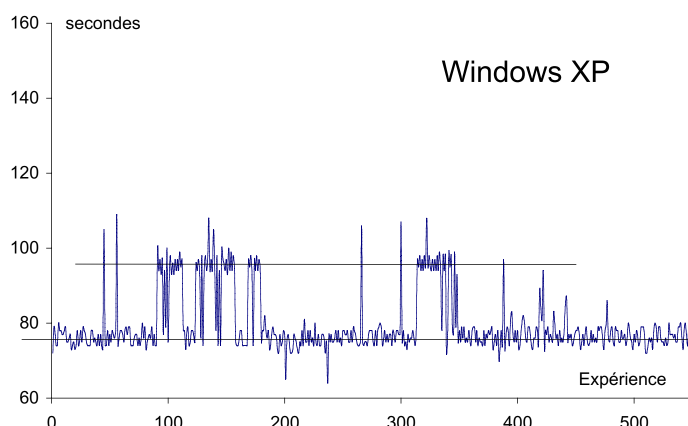


Figure 4-8 : Détails des temps de redémarrage de Windows XP

Le nombre d'expériences qui ont abouti à des cas de Blocage/Abandon de l'activité sont respectivement 101, 107 et 128 pour Windows NT4, Windows 2000 et Windows XP. Bien que Windows XP ait généré plus de cas d'Abandon/Blocage de l'activité, il a toujours le temps de redémarrage le plus faible comme indiqué dans le Tableau **4-7**. Dans ce tableau, les deux premières lignes rappellent le temps de redémarrage du système en absence de fautes, τred, et en présence de fautes, Tred, déjà illustrés dans le Tableau 4-2. Les deux dernières lignes, raffinant le temps de redémarrage de l'OS par rapport à l'état final de l'activité, montrent que le temps de redémarrage avec un cas d'Abandon/Blocage de l'activité augmente de 8 % à 18 % par rapport au temps de redémarrage en absence de fautes.

Tableau 4-7 : Temps détaillé de redémarrage du système

	Windows NT4	*Windows 2000*	*Windows XP*
τred	*92 s*	*105 s*	*74 s*
Tred	*96 s*	*109 s*	*80 s*
Tred après terminaison de l'activité	*95 s*	*106 s*	*76 s*
Tred après Abandon/Blocage de l'activité	*102 s*	*123 s*	*90 s*

4.4.4 Temps d'exécution de l'activité

Le Tableau **4-8** illustre, pour les trois systèmes d'exploitation, la durée d'exécution de l'activité en absence (τTac) et en présence de fautes (TTac). Comparée à la durée d'exécution en absence de fautes, nous constatons que TTac est supérieure de 3% pour Windows XP, 6 % pour Windows 2000 et 8% pour Windows NT4. De plus, les écarts type relatifs aux différentes moyennes des trois OSs sont très petits. Finalement, en comparant

les trois systèmes d'exploitation, Windows XP a la plus petite durée d'exécution de l'activité en l'absence et en présence de fautes.

Tableau 4-8 : Temps d'exécution de l'activité

	τTac	TTac	Ecart Type
Windows NT4	74 s	80 s	12 s
Windows 2000	70 s	74 s	13 s
Windows XP	67 s	69 s	10 s

4.4.5 Récapitulatif

Les résultats concernant les mesures temporelles sont en concordance avec la déclaration de Microsoft qui apparaît lors de l'installation de Windows XP. Microsoft proclame que Windows XP est la version la plus rapide des OSs de la famille Windows. En effet, lors de l'installation de Windowx XP, le message suivant apparaît :

> *Your Computer will be faster and more reliable*
>
> Windows XP professional not only starts faster than any previous version, but it also runs your programs more quickly and reliably than ever. If a program becomes unstable, you can close it without having to shutdown Windows[4].

4.5 Validation des propriétés par expérimentation

Dans ce paragraphe, nous mettons l'accent sur la vérification de certaines propriétés d'un étalon qui ne peuvent être validées que par expérimentation en considérant la même famille d'OSs. Ces propriétés correspondent à la reproductibilité, la répétitivité, la représentativité de l'ensemble de fautes, la portabilité, l'interférence et le coût de l'étalonnage.

4.5.1 La reproductibilité

4 Il est important de noter que le terme « reliability » tel que défini dans ce message est différent de la mesure de robustesse évaluée dans le cadre de notre étalon de sûreté de fonctionnement.

La vérification de la reproductibilité nécessite le développement de plusieurs prototypes par des personnes différentes, à partir de la spécification du chapitre 3. Nous n'avons malheureusement pas pu vérifier cette propriété expérimentale.

4.5.2 La répétitivité

La répétitivité est la propriété qui garantit l'obtention des résultats statistiquement équivalents quand l'étalon est exécuté plusieurs fois dans le même environnement (les mêmes système cible, système à étalonner, profil d'exécution et prototype). Pour chacun des trois systèmes d'exploitation ciblés dans ce chapitre, les expériences ont été répétées trois fois dans les mêmes conditions.

En termes de robustesse, les résultats obtenus étaient exactement identiques pour chacun des systèmes d'exploitation.

En ce qui concerne les mesures temporelles, aucune différence signaficative n'a été enregistrée pour les temps de réaction de chacun des trois OSs ciblés. Les différences entre les temps moyens de réaction de chacun des systèmes (Texec) n'ont jamais dépassé 4%, tout en conservant l'avantage de Windows XP par rapport à Windows NT4 et Windows 2000. La variation des temps de redémarrage des OSs n'a pas dépassé 2 secondes, ce qui représente moins de 3% de variation par rapport aux valeurs moyennes présentées dans le Tableau 4-2.

4.5.3 La représentativité

Les résultats obtenus pour les trois systèmes d'exploitation de la famille, concernant leur robustesse et les mesures temporelles, ne contredisent pas les déclarations de *Microsoft*, la société qui les a développés.

La représentativité de l'ensemble de fautes à injecter est sans aucun doute la dimension la plus critique d'un étalon de sûreté de fonctionnement en général. La technique retenue pour corrompre les paramètres des appels système interceptés est la substitution sélective (cf. section 3.2.2.2). Nous présentons dans la suite de cette section une analyse de sensibilité vis-à-vis de l'ensemble des appels système retenus pour les expériences et de l'ensemble des fautes considérées. Cette analyse confirme l'équivalence des trois OS vis-à-vis de la mesure de robustesse quel que soit l'ensemble des appels système ciblés et l'ensemble des fautes considérées.

L'ensemble de fautes injectées consiste à remplacer les valeurs correctes des paramètres des appels système par des valeurs erronées. Comme expliqué dans le chapitre précédent, une association de trois techniques de corruption de paramètres a été utilisée 1) données hors-limite, 2) données incorrectes et 3) adresses incorrectes. Nous avons analysé dans un premier temps l'impact de corruption de chacun de ces trois types de paramètres sur les résultats de l'étalon. Dans un deuxième temps, nous avons effectué une analyse de sensibilité vis-à-vis de l'ensemble des appels système retenus. Finalement, nous présentons une analyse comparative entre les résultats obtenus par la technique de substitution sélective et ceux obtenus en utilisant la substitution systématique.

4.5.3.1 Impact de la technique de corruption des paramètres

Pour étudier la sensibilité des mesures obtenues dans ce chapitre, nous avons étudié l'impact de la technique de corruption par rapport à chaque type de paramètre. Pour cela, nous avons décomposé l'ensemble de fautes comme suit :

F0 : les valeurs des paramètres sont substituées par des données hors-limite. L'ensemble de fautes contient 113 substitutions sélectives.

F1 : les valeurs des paramètres sont substituées par des adresses incorrectes. L'ensemble de fautes contient 212 substitutions sélectives.

F2 : les valeurs des paramètres sont substituées par des données incorrectes. L'ensemble de fautes contient 227 substitutions sélectives..

Rappelons tout d'abord que les résultats obtenus avec l'ensemble complet des fautes (*F0+F1+F2*) ont montré que les trois OS sont équivalents en termes de robustesse. Cependant, il peut être argumenté que les données incorrectes ne sont pas représentatives des fautes d'origine applicative que l'OS doit détecter. Pour analyser leur impact sur les mesures obtenues, nous avons considéré un ensemble réduit de fautes correspondant aux données hors-limite et aux adresses incorrectes (*F0+F1*). La Figure 4-9 présente les résultats de robustesse des trois OSs ciblés avec l'ensemble de fautes *F0+F1.* La comparaison de ces résultats avec ceux présentés dans la Figure 4-1 montre qu'une légère différence existe entre les résultats obtenus après l'application de *F0+F1+F2* et *F0+F1.* Cependant, la Figure 4-9 montre que les trois OSs restent toujours équivalents en termes de robustesse. Comme dans le cas de la Figure 4-1, la plus grande différence observée pour les cas de Non Signalement ne dépasse pas les 3 %.

D'autre part, les adresses incorrectes que nous considérons pointent sur des données incorrectes ou des données hors-limite. Si nous considérons que les pointeurs pointent tous vers des données incorrectes, et non hors-limite (cas pessimite), l'ensemble F1 peut être écarté pour la même raison que F2. La Figure 4-10 montre que lorsqu'on applique l'ensemble de fautes *F0*, les trois systèmes d'exploitation ont des robustesses similaires. Encore une fois, la plus grande différence enregistrée pour les cas de non signalement ne dépasse pas les 4,3%.

Les résultats obtenus par l'application de *F0* sont très proches de ceux obtenus par l'application de l'ensemble complet de fautes utilisé dans notre prototype (Figure 4-1). Ce résultat est très important dans la mesure où il nous a permis, dans le paragraphe suivant, de corrompre tous les appels système activés par le client TPC-C en n'utilisant que des données hors-limite, sans forcément augmenter la durée d'exécution de l'étalon.

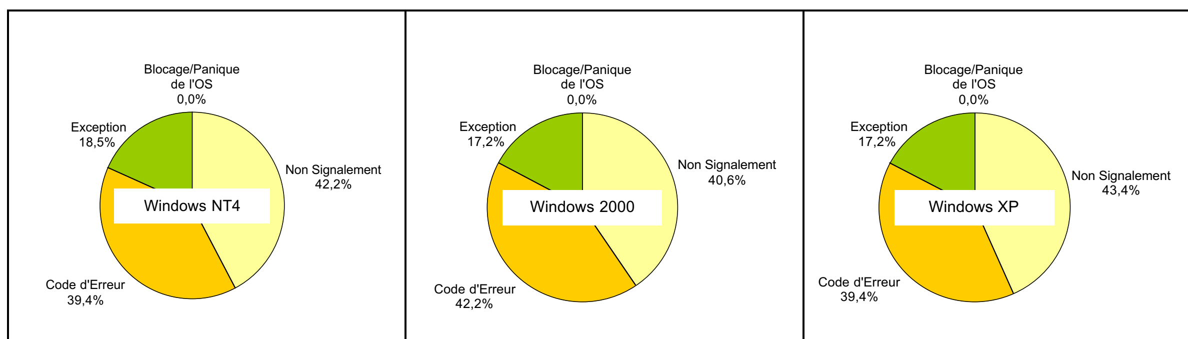


Figure 4-9 : Robustesse de l'OS par rapport aux données hors-limite et aux addresses incorrectes (F0+F1)

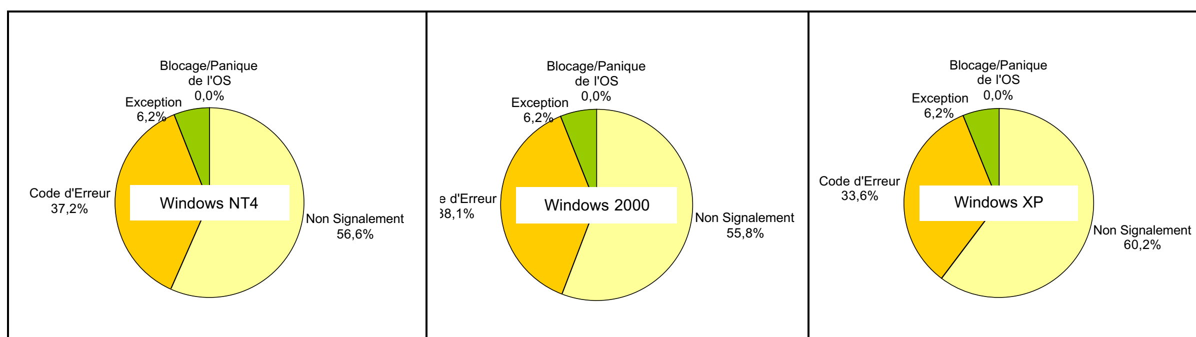


Figure 4-10 : Robustesse de l'OS par rapport aux données hors-limite (F0)

4.5.3.2 Impact des appels système considérés

Pour analyser l'impact du choix de l'ensemble des appels système dont les paramètres ont été corrompus, nous avons corrompu tous les 132 appels système avec paramètres sollicités par le client TPC-C. En nous basant sur les résultats du paragraphe précédent, nous avons substitué les valeurs de ces paramètres par des données hors-limite uniquement. Appelons *F3* l'ensemble de fautes où tous les 132 appels système activés par TPC-C sont considérés, mais la substitution des valeurs de paramètres est réduite à des données hors-limite. Cela conduit à un total de 468 substitutions sélectives, qui est même inférieur au cas de l'ensemble complet (*F0+F1+F2*).

Les résultats obtenus par l'application de *F3* sont présentés dans la Figure 4-11. Sur cette figure, nous constatons que les trois OSs de la famille Windows ont des robustesses similaires. Une fois de plus, la plus grande différence enregistrée ne dépasse pas les 5,3 % pour les cas de non signalement observés.

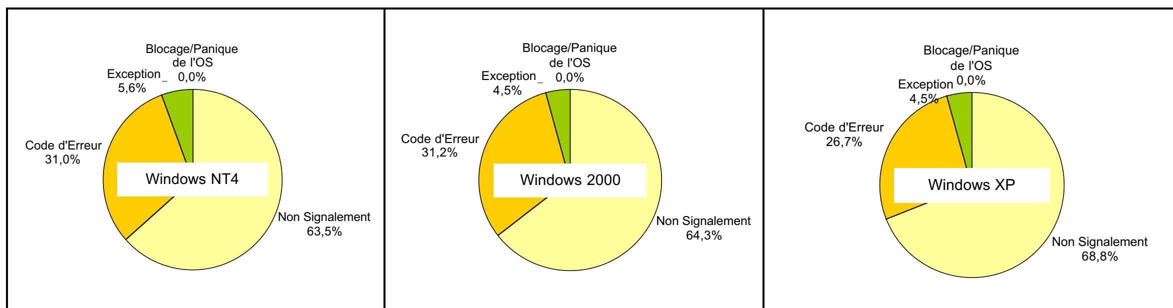


Figure 4-11 : Robustesse de l'OS par rapport aux données hors-limite, appliquées à tous les 132 appels système (F3)

Nous notons que le sous ensemble des appels système activés par le client TPC-C, sélectionnés selon l'appartenance à des composants fonctionnels les plus utilisés d'un OS, a donné des résultats de comparaison similaires à ceux obtenus en ciblant l'ensemble de tous les appels système activés ; Windows NT4, Windows 2000 et Windows XP sont toujours équivalents en termes de robustesse.

4.5.3.3 Comparaison entre substitution sélective et substitution systématique

Dans ce paragraphe, nous présentons une analyse supplémentaire de la sensibilité des résultats obtenus par rapport à la technique de corruption de paramètres, en considérant les deux techniques de corruption : la substitution sélective et la substitution systématique. Appelons *F4* l'ensemble de fautes où les 75 paramètres des 28 apppels système retenus sont soumis à la technique de substitution systématique, donnant lieu ainsi à 2400 expériences (32 substitutions par paramètres). Les résultats de ces substitutions sont donnés dans la Figure 4-12 pour Windows 2000. Cette figure montre que la robustesse de Windows 2000 ne varie pas significativement avec les deux techniques de corruption de paramètres, ce qui confirme les résultats obtenus dans [Jarboui 2002a]. En effet, la plus grande différence enregistrée dans le cas de retour de codes d'erreur ne dépasse pas les 10 %.

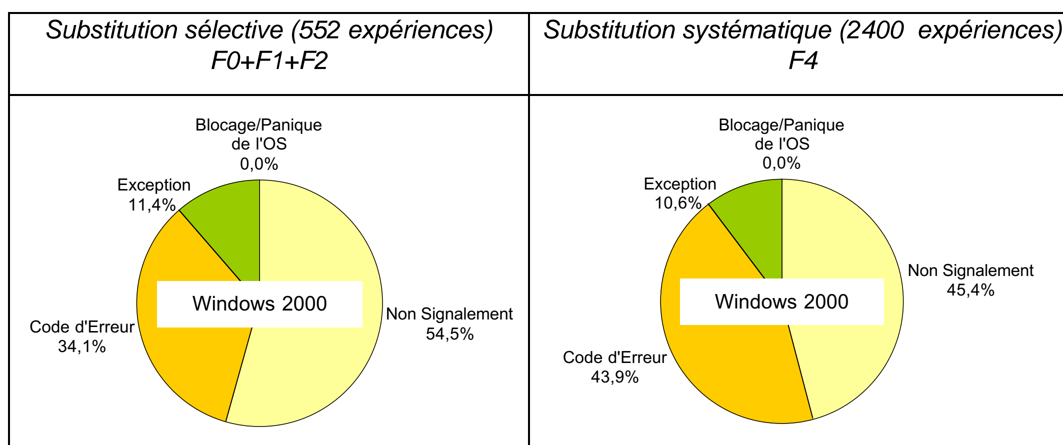


Figure 4-12 : Robustesse de Windows 2000 par rapport à (F0+F1+F2) et F4

4.5.3.4 Récapitulatif

D'après les résultats montrés jusqu'à maintenant, nous concluons que les trois systèmes d'exploitation de la famille Windows que nous avons ciblé sont équivalents en termes de robustesse, et cela quelle que soit la technique de corruption de paramètres, à condition de corrompre le même ensemble de paramètres appartenant au même ensemble d'appels système sollicités par l'activité. Le Tableau **4-9** synthétise les expériences effectuées pour étudier la sensibilité des résultats obtenus à l'aide du prototype présenté dans ce chapitre.

D'abord, nous avons étudié la sensibilité par rapport au choix des valeurs corrompues. Pour cela, nous avons considéré dans un premier temps seulement les données hors limite et les adresses incorrectes (*F0+F1*) avant de considérer seules les données hors limite (*F0*). Tous les résultats obtenus montrent une équivalence en termes de robustesse entre les trois OSs ciblés. De plus, l'application de l'ensemble *F0* a donné les mêmes résultats que l'ensemble complet *F0+F1+F2*, avec approximativement cinq fois moins d'expériences par rapport aux substitutions sélectives.

Nous avons également étudié la sensibilité des mesures obtenues par rapport au choix des appels système à corrompre, en substituant cette fois-ci les paramètres des 132 appels système activés par le client TPC-C par des données hors limite (*F3*). Une fois de plus, les trois systèmes ciblés sont équivalents en termes de robustesse.

Ensuite, nous avons étudié la sensibilité des résultats par rapport à la technique de substitution adoptée. Nous avons comparé la technique de substitution sélective avec l'ensemble complet des fautes à la technique de substitution systématique (*F4*). Les expériences menées sur Windows 2000 ont montré une équivalence de robustesse de Windows 2000 en utilisant ces deux ensembles de fautes.

Tableau 4-9 : Validation du modèle de fautes

	Substitution sélective			Substitution systématique	# Appels système	# expériences
	Données hors-limite	Adresses incorrectes	Données incorrectes			
F0+F1+F2	x	x	x		28	552
F0+F1	x	x			28	325
F0	x				28	113
F3	x				Tous (132)	468
F4				x	28	2400

Etant donné que sa durée d'exécution est nettement inférieure à celle de substitution systématique, notre recommendation est d'utiliser la technique de substitution sélective avec l'ensemble complet des fautes. Néanmoins, les résultats obtenus avec des données

hors-limite uniquement (*F0*) pour la famille Windows nous permettent de conseiller l'utilisation de cet ensemble de fautes pour l'étalonnage des OSs de cette famille. Cette étude sera réalisée pour les OSs Linux dans le chapitre suivant.

4.5.4 La portabilité

Nous avons montré dans le chapitre précédent (cf. section 3.3.3) que la spécification des étalons que nous proposons est portable pour différents systèmes d'exploitation qu'ils soient de la même famille ou de familles différentes.

Nous avons présenté dans ce chapitre les résultats du prototype qui a été porté sur les trois systèmes d'exploitation Windows NT4, Windows 2000 et Windows XP, ayant en commun l'API Win32. Cependant, Windows NT4 et Windows 2000 ont activé les mêmes 28 appels systèmes avec les même paramètres, alors que Windows XP a activé deux appels système de plus. Nous avons donc réalisé une légère adaptation consistant à prendre en compte ces deux nouveaux appels système. Nous avons donc ajouté ces deux appels système à la liste des appels à corrompre.

4.5.5 L'interférence

Comme dans toutes les activités d'étalonnage des systèmes informatiques, le problème d'interférence est toujours présent, parce que l'étalon doit obligatoirement utiliser les ressources du système à étalonner. Le but recherché est de minimiser les effets d'interférence sur les mesures de l'étalon.

Tous les composants du prototype de l'étalon de sûreté de fonctionnement que nous présentons s'exécutent hors du système cible. Par conséquent, ils n'ont aucune influence sur les mesures de robustesse obtenues.

En ce qui concerne les mesures temporelles, nous avons essayé d'avoir le moins d'impact sur l'OS. Nous avons calculé le temps de réaction de l'OS comme étant la différence entre l'instant de reprise d'exécution de l'appel système après substitution et l'instant de fin d'exécution d'appel système corrompu. Ceci nous a permi d'écarter du calcul le temps de substitution de valeur. Pour calculer ces deux instants, nous avons utilisé l'instruction `QueryPerformanceCounter` qui permet l'accès, en mode utilisateur, au compteur de l'horodateur. Ce dernier est un registre incrémenté à chaque cycle horloge. Pour obtenir notre mesure souhaitée, nous calculons la différence des deux valeurs de l'horodateur aux instants de reprise et de fin d'exécution de l'appel système, et divisons le résultat par la fréquence du processeur. Cette méthode de calcul nous a permi de réduire l'erreur de mesure jusqu'à 2 µs.

4.5.6 L'automatisation

L'étalonnage de la sûreté de fonctionnement des systèmes d'exploitation nécessite un nombre important d'expériences à réaliser (552 expériences dans le cas présenté dans ce chapitre), et par conséquent, il faut envisager l'automatisation du déroulement de cette activité d'étalonnage.

Toutes les étapes de l'étalonnage, allant des expériences jusqu'au traitement des résultats, ont été automatisées. En effet, des scripts sont chargés d'enchaîner les expériences, l'une après l'autre jusqu'à la fin de la série d'expériences. Ces scripts utilisent des fichiers de configuration, définis à priori, pour fixer les paramètres de chaque expérience (l'appel système à corrompre, le numéro de paramètre à corrompre et la valeur de substitution). Pour améliorer cette automatisation, nous avons mis en place un système de redémarrage matériel capable de redémarrer la machine cible suite à un cas de blocage ou de panique du système cible.

À la fin de la série d'expériences, les fichiers contenant les résultats de chaque expérience sont traîtés pour donner lieu à un seul fichier qui peut être exploité par une base de données pour fournir les mesures finales de l'étalon.

4.5.7 Le coût de l'étalonnage

Développer et exécuter un étalon de sûreté de fonctionnement d'OS demande un certain effort qui, de notre point de vue, est tout à fait acceptable. Dans notre cas, la plus grande partie de l'effort a consisté à définir les concepts et l'ensemble de fautes à considérer, et à étudier la sensibilité de l'ensemble de fautes utilisées.

L'implémentation de l'étalon pour Windows 2000 et Windows NT a demandé un effort d'environ un mois qui se répartit comme suit :

- l'installation du client TPC-C a pris trois jours[5].
- la mise en place de l'ensemble de fautes a pris une semaine durant laquelle nous avons défini l'ensemble des valeurs de substitution pour les types de paramètres des 28 appels système considérés et créé la base de données correspondante. En fait, cette durée est suffisante pour un développeur qui a une bonne connaissance des types de paramètres considérés dans l'OS ciblé et des valeurs de substitution qu'il faut définir. En revanche, une durée plus importante est nécessaire pour ceux qui ne connaissent pas le système cible. En fait, une durée de familiarisation avec la technique de substitution sélective est nécessaire dans ce cas.
- la mise en œuvre des différents composants du contrôleur a nécessité deux semaines, en incluant l'adaptation de l'outil *Detours* pour injecter les fautes, observer les réaction de l'OS, et calculer le temps de réaction des appels système.

Pour porter notre étalon sur Windows XP, il nous a fallu une journée pour implémenter dans l'outil *Detours*, les mécanismes d'injection et d'observation de fautes pour les deux nouveaux appels systèmes. Pour la mise à l'échelle de notre étalon dans le cas de l'ensemble *F3* (cf. section 4.5.3.2), le temps nécessaire pour adapter l'outil *Detours* a été de l'ordre de deux semaines à cause du nombre important d'appels système à considérer.

---

[5] Il convient toutefois de rappeler que nous avons bénéficié sur ce point de l'installation qui avait été faite par ailleurs par un autre partenaire du projet DBench [Vieira 2003].

L'exécution de l'étalon est de deux jours pour chaque OS. La durée d'une expérience dans le cas d'une terminaison normale de l'activité est inférieure à 3 minutes (incluant le temps d'exécution de l'activité et le temps de redémarrage), alors qu'elle est d'environ 7 minutes dans le cas où l'activité ne se termine pas normalement (incluant la durée du chien de garde de 5 minutes et le temps de redémarrage). En moyenne, nous avons observé une durée inférieure à 5 minutes par expérience. L'enchaînement des expériences étant totalement automatique (même en cas de blocage de l'OS), la durée totale d'exécution de l'étalon est alors d'environ 46h pour chaque OS. 552 expériences sont exécutées pour chaque OS. Cette durée peut être considérablement réduite en ne considérant que des substitutions de valeurs du type hors-limite.

4.6 Conclusion

Le prototype de l'étalon de sûreté de fonctionnement présenté dans ce chapitre est basé sur le client TPC-C et cible les systèmes d'exploitation de la famille Windows. Trois systèmes Windows ont été soumis à cet étalon : Windows NT4, Windows 2000 et Windows XP.

Après avoir détaillé la mise en œuvre de cet étalon pour les systèmes Windows, nous avons présenté les résultats obtenus. Ces résultats révèlent que Windows XP est équivalent à Windows NT4 et Windows 2000 d'un point de vue de la robustesse. Cette équivalence de robustesse entre les trois systèmes n'est pas surprenante, étant donné qu'ils appartiennent à la même famille de systèmes d'exploitation, et que la production d'une nouvelle version d'un système d'exploitation au sein d'une famille de systèmes se fait surtout pour ajouter de nouveaux modules à l'ancienne version. En revanche, nous avons trouvé que Windows XP détient des temps de réaction et de redémarrage plus courts que ceux de Windows NT4 et Windows 2000, en présence ainsi qu'en absence de fautes. Ces résultats ne contredisent pas les proclamations de la compagnie Microsoft, productrice de cette famille de systèmes d'exploitation. Par la suite, les mesures complémentaires ont confirmé les mesures de base obtenues.

Enfin, nous avons validé les propriétés de l'étalon qui nécessitent de l'expérimentation pour être vérifiées. Nous avons mis l'accent sur la sensibilité des mesures obtenues par rapport à l'ensemble de fautes utilisées. Ainsi, nous avons étudié l'impact de la technique de corruption de paramètres (données correctes et incorrectes, adresses incorrectes) des appels système corrompus sur les résultats obtenus. Nous avons également réalisé une comparaison entre la substitution sélective et la substitution systématique. Tous les résultats ont confirmé l'équivalence des trois systèmes d'exploitation.

Dans le chapitre 5, nous nous intéressons à la validation de ces résultats, en utilisant cette fois-ci deux autres étalons basés sur les activités *Postmark* et *JVM*. De plus, ces deux étalons seront utilisés pour comparer la sûreté de fonctionnement de différents systèmes d'exploitation des familles Windows et Linux.

Chapitre 5 Étalonnage de Windows et de Linux

5.1 Introduction

Les systèmes d'exploitation Windows ont longuement acquis la réputation d'être moins robustes que les systèmes d'exploitation Unix. Face à un choix entre ces deux systèmes, l'image principale qui surgit est le fameux *écran bleu* (blue screen) affiché lors de la défaillance des systèmes Windows. De plus, il est connu que les systèmes Windows ont besoin d'être redémarrés plus souvent que les systèmes Unix. Or, peu d'études quantitatives ont été publiées sur la sûreté de fonctionnement de ces deux grandes familles de systèmes d'exploitation. Dans ce chapitre, nous présentons les résultats de l'étalonnage de la sûreté de fonctionnement de différents systèmes d'exploitation de ces deux grandes familles.

Pour atteindre ce but, nous utilisons deux étalons de sûreté de fonctionnement pour caractériser la sûreté de fonctionnement de différentes versions et révisions de Linux et de différents systèmes Windows. Ces deux étalons de sûreté de fonctionnement sont basés sur les activités *Postmark* et *JVM* (cf. chapitre 3). Nos expérimentations ont porté sur les systèmes étudiés dans le quatrième chapitre (Windows NT4 Workstation, Windows 2000 Professional et Windows XP), et sur trois versions de la distribution *Debian* de Linux : 2.2, 2.4 et 2.6. La mise en œuvre de ces deux étalons est présentée dans la première section de ce chapitre.

Nous présenterons ensuite les résultats obtenus par l'étalon de sûreté de fonctionnement DBench-OS-Postmark suivis par les résultats de DBench-OS-JVM. Notons que *Postmark* est initialement conçu pour étalonner la performance des serveurs de fichiers. Pour cette raison, en plus des OSs de la famille Linux ciblés, nous avons jugé intéressant d'ajouter à l'ensemble initial des systèmes ciblés de la famille Windows les dernières versions des serveurs de Windows : Windows NT4 Server, Windows 2000 Server et Windows 2003 Server.

5.2 Mise en œuvre

Dans le chapitre précédent, nous avons présenté la mise en œuvre du prototype de l'étalon DBench-OS-TPCC. Ce prototype nous a servi à étalonner les OSs de la famille Windows. Cette mise en œuvre pour les systèmes Windows est exploitée pour développer les deux prototypes basés sur Postmark et JVM. La différence majeure entre ces étalons réside dans le profil d'exécution utilisé. Or, lors des spécifications des trois étalons présentés dans le

troisième chapitre, nous avons mentionné que la technique de corruption des paramètres des appels système et les mécanismes d'observation sont indépendants de l'activité. La seule modification apportée à la mise en œuvre de DBench-OS-TPCC pour obtenir les prototypes des deux étalons définis dans le troisième chapitre a consisté à modifier la liste des appels système à corrompre et les substitutions associées.

L'autre famille de systèmes d'exploitation ciblée dans nos travaux correspond à Linux. Faisant partie des systèmes Unix, Linux est un système d'exploitation de type *temps partagé*. Linux est de type *logiciel libre*. Son noyau est de type monolithique intégrant tous les composants fonctionnels du système d'exploitation. Ces composants fonctionnels sont disponibles via l'interface *Posix*, acronyme de ***P**ortable **O**perating **S**ystem **I**nterface for UNI**X***, conçue pour améliorer la portabilité des applications sur les différents systèmes d'exploitation de la famille Unix. Pour pouvoir s'exécuter, les applications peuvent solliciter les environ 250 appels système de l'interface *Posix*.

Plusieurs communautés du monde entier ont participé au développement de systèmes d'exploitation Linux, d'où l'existence de plusieurs distributions comme *Debian*, *RedHat*, *Suse*, etc. Dans nos travaux, trois versions de la distribution *Debian* ont été considérées : 2.2, 2.4 et 2.6. Afin de comparer les différences du point de vue de la sûreté de fonctionnement entre deux révisions de la même version, nous avons considéré deux révisions de la version 2.4. Au total, nous avons étalonné quatre systèmes d'exploitation Linux de la distribution *Debian* : 2.2.6, 2.4.5, 2.4.26 et 2.6.6.

Pour intercepter les appels système de l'interface Posix, nous avons utilisé l'outil *Strace* [McGrath 2004]. Cet outil permet de tracer tous les appels système sollicités par l'activité qui lui sont donnés en paramètres. Comme dans le cas de *Detours*, *Strace* intercepte les appels système, écrit les informations liées à leurs paramètres dans un fichier de trace et relance leur exécution. Cet outil a été modifié pour permettre la substitution contrôlée des valeurs des paramètres des appels-système. Des mécanismes d'observation du temps d'exécution et des retours des codes d'erreur ou des exceptions ont été également implémentés.

Les valeurs de substitution ont été choisies selon le même principe que celui présenté dans le chapitre 3. Les types de paramètres des appels système de Posix sont classés dans différentes classes, et des valeurs de substitution ont été définies pour chacune d'entre elles.

Au cours de son exécution, *PostMark* active environ 27 appels système Win32 avec paramètres. Ces appels système utilisent entre 53 et 64 paramètres sur lesquels nous avons effectué des substitutions sélectives. Cela a donné lieu à 418 substitutions environs, donc 418 expériences à réaliser pour chaque version de Windows. En ce qui concerne le système Linux, *Postmark* active 16 appels système environ avec 38 paramètres donnant lieu à 206 expériences approximativement. Notons que la plupart des appels système activés par *Postmark* (plus que 60% pour les deux familles de systèmes d'exploitation) fait partie du module *Système de fichiers*, ce qui est logique étant donné que cet étalon a été développé pour mesurer la performance de systèmes de fichiers.

Pour exécuter la JVM, nous avons utilisé un programme Java qui permet d'afficher le message « Hello World » sur l'écran. Pour chacun des systèmes Windows, 76 appels système Win32 avec paramètres sont sollicités, dont 65 sont communs aux trois OSs. Ces appels systèmes utilisent au total 205 paramètres sur lesquels nous avons appliqué une substitution sélective. Cela nous a conduit à la réalisation d'environ 1285 expériences pour Windows. La même activité sollicite approximativement 35 appels système POSIX avec environ 84 paramètres, donnant lieu à la réalisation de 408 à 457 expériences pour Linux. Pour les deux familles considérées de systèmes d'exploitation, les appels système activés par la JVM appartiennent en grande majorité aux composants fonctionnels Processus et brins, Système de fichiers et la Gestion de mémoire.

Le Tableau 5-1 récapitule le nombre d'expériences réalisées avec chacun des prototypes présentés dans ce paragraphe.

Tableau 5-1 Récapitulatif du nombre des valeurs corrompues

Activité→ *↓OS*	***DBench-OS-Postmark***		***DBench-OS-JVM***	
	appels système	**expériences**	**appels système**	**expériences**
Windows NT	*25*	*418*	*76*	*1285*
Windows 2000	*27*	433	*76*	*1294*
Windows XP	*27*	*424*	*76*	*1282*
Windows NT Server	*25*	*418*		
Windows 2000 Server	*27*	433		
Windows 2003 Server	*27*	433		
Linux 2.2.26	*16*	*206*	*37*	*457*
Linux 2.4.5	*16*	*206*	*32*	*408*
Linux 2.4.26	*16*	*206*	*32*	*408*
Linux 2.6.6	*17*	228	*31*	*409*

Dans le troisième chapitre, nous avons mentionné l'impossibilité de détecter l'état final de l'activité (terminaison correcte ou incorrecte, blocage ou abandon de l'activité) sans disposer du code source de ce dernier (cf. section 3.2.3). En effet, la détection de la terminaison correcte ou incorrecte de l'activité dépend de la nature de l'activité d'étalonnage. L'indisponibilité des codes source du client *TPC-C* et de la *JVM* nous a rendu impossible d'observer leur état à la fin de chaque expérience. D'un autre côté, la disponibilité du code source de l'activité de *Postmark* nous a permis de recueillir des observations concernant la terminaison de son exécution. En effet, il a fallu mettre en place

une méthode permettant de déduire si l'activité s'est exécutée correctement ou incorrectement jusqu'à sa fin ou bien si elle s'est bloquée en cours d'exécution ou si elle a été abandonnée. *Postmark* est essentiellement une application qui fait des opérations sur des fichiers et des répertoires. Pour savoir si l'activité s'est terminée correctement, il faut savoir si elle a effectué toutes les opérations qu'elle aurait dû réaliser dans un environnement normal, c'est-à-dire en absence de fautes. La méthode adoptée a été d'insérer des tests dans le logiciel. Si une opération a bien été effectuée, un message est écrit dans un fichier nommé *fichier d'exécution*. Si l'exécution de l'activité arrive à sa fin normale, alors un message particulier est écrit dans ce même fichier. Ainsi, en analysant le *fichier d'exécution*, il est possible de caractériser de manière pratique les quatre possibilités :

1) Terminaison correcte de l'activité : tous les messages correspondant à toutes les opérations sont présents dans le fichier d'exécution ;
2) Terminaison incorrecte de l'activité : il manque des messages correspondant à une ou plusieurs opérations dans le fichier d'exécution ;
3) Blocage de l'activité : il manque au moins le message de fin dans le fichier d'exécution ;
4) Abandon de l'activité : comme pour le blocage, il manquera au moins le message de fin dans le fichier d'exécution.

Les deux derniers cas sont distingués à l'aide des observations du chien de garde qui s'exécute sur le contrôleur : si le délai s'est expiré et le message de fin n'est pas présent dans le *fichier d'exécution*, c'est que l'activité s'est bloquée.

À la fin des expériences, les fichiers de données bruts sont traîtés pour produire un fichier exploitable dans la base de données définies dans le paragraphe 4.2, et qui permet de calculer les mesures finales de l'étalon.

5.3 Résultats de DBench-OS-Postmark

Dans cette section, nous présentons le résultat de l'étalonnage des systèmes d'exploitation de deux familles Windows et Linux. Le prototype utilisé est celui qui a *Postmark* comme activité (cf. Chapitre 3).

5.3.1 Mesures de l'étalon

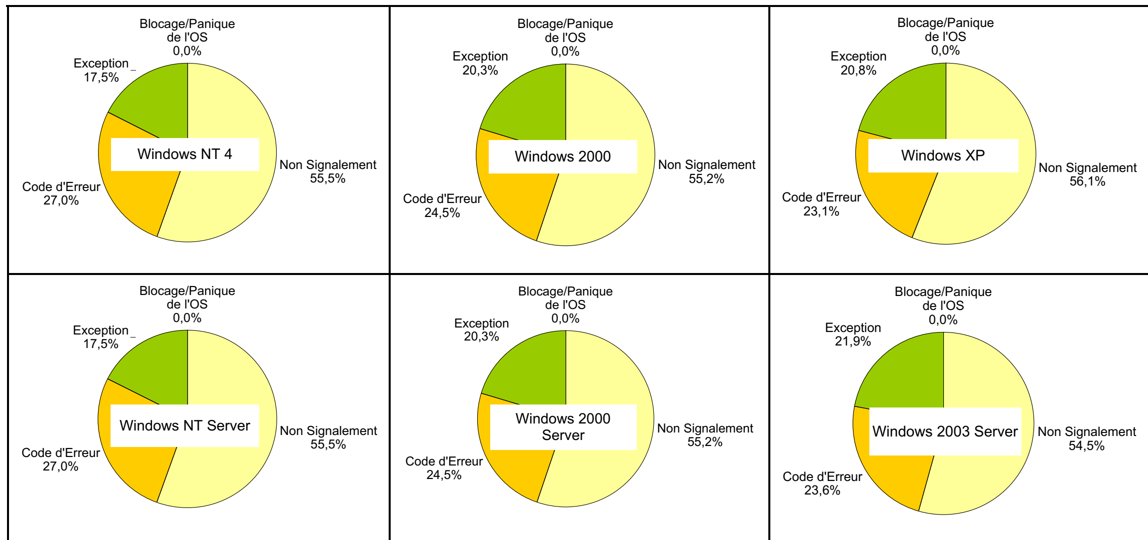
Les mesures de robustesse sont présentées dans la Figure 5-1. Aucun état de blocage ou de panique de l'OS n'a été observé pour les systèmes d'exploitation ciblés.

Concernant Windows, la corruption des paramètres des appels système a donné lieu, entre 17% et 21%, à des levées d'exception. Les taux de notification de codes d'erreur ont varié entre 23% et 27%. Les systèmes d'exploitation n'ont pas détecté les paramètres corrompus entre 54.5% et 56.1% des cas. Ces résultats obtenus montrent que les OSs ont des comportements similaires en présence de fautes, ce qui confirme les résultats présentés dans

le Chapitre 4. Cette équivalence peut être expliquée par le fait que tous les systèmes ciblés dans nos travaux sont basés sur la technologie NT.

En ce qui concerne les systèmes d'exploitation de la famille Linux, nous avons étudié la variation des comportements de différentes révisions en présence de fautes. Pour cela, nous avons considéré deux révisions de la version 2.4 : Linux 2.4.5 et Linux 2.4.26. Les résultats obtenus pour ces deux révisions montrent des comportements similaires. En effet, les mêmes pourcentages de retour de codes d'erreurs, d'exceptions levées et de cas de non signalement ont été observés pour les deux révisions. Nous concluons que la robustesse de différentes révisions de la même version ne varie pas, ce qui est justifié par le fait que, généralement, les modifications apportées à une ancienne révision d'une version correspondent surtout à l'ajout de nouvelles extensions. Les résultats de robustesse de la version 2.4, ainsi que de deux autres versions ciblées sont présentés dans la Figure 5-1. Les OSs de la famille Linux ont notifié la présence du paramètre corrompu par des exceptions dans 7.8% à 9.7% des cas, alors que dans 58.8% à 67.5% des cas, ces OSs ont généré des codes d'erreur. Les cas de non signalement observés varient entre 24.8% et 31.6%. Ces résultats permettent de conclure que ces quatre versions de la famille Linux sont également équivalentes en termes de robustesse.

Famille Windows



Famille Linux

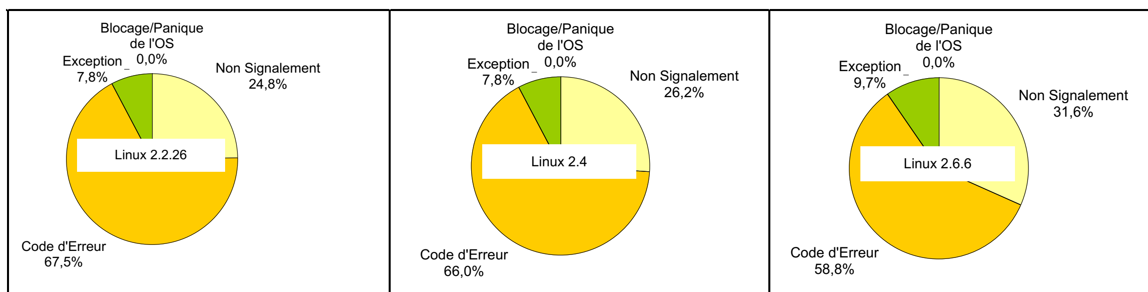


Figure 5-1 : Mesures de robustesse

Les temps de réaction des OSs sont présentés dans le Tableau 5-2. Nous rappelons que τexec correspond au temps moyen d'exécution des appels système à corrompre, en absence de fautes. En présence de fautes, ce temps moyen est désigné par Texec.

Le Tableau 5-2 montre que, en général, τexec est supérieur à Texec. Toutefois, une exception est notée pour les deux révisions de la version 2.4 des OSs de la famille Linux. Cette supériorité de τexec par rapport à Texec peut s'expliquer par le fait qu'après la corruption du paramètre, l'OS n'arrive pas à finir tout ce qui était prévu de faire et finit son exécution plus tôt. Nous notons que parmi les OSs de la famille Windows, *Windows 2003 Server* a le temps de réaction le plus faible en présence de fautes (*Windows 2003 server* est la dernière version de Windows lancée sur le marché). Pour les versions destinées aux utilisateurs finaux, Windows XP occupe la meilleure place avec 114 µs. Néanmoins, nous remarquons que les Texec correspondants aux différentes versions de Windows sont relativement proches, le plus grand écart étant de 50 µs.

Pour les OSs de la famille Linux, nous trouvons que Linux 2.6.6 bénificie du meilleur temps de réaction en présence de fautes. Nous remarquons également que, à la différence des systèmes de la famille Windows, les *τexec* des deux révisions de la version 2.4 sont supérieurs aux *Texec* correspondantes.

Tableau 5-2 : temps de réaction du système

Famille Windows

	Windows NT4		*Windows 2000*		*Windows XP*	
	Moyenne	*Ecart Type*	*Moyenne*	*Ecart Type*	*Moyenne*	*Ecart Type*
τexec	*248* µs		*136* µs		*315* µs	
Texec	*148* µs	*219* µs	*118* µs	*289* µs	*114* µs	*218* µs

	Windows NT4 Server		*Windows 2000 Server*		*Windows 2003 Server*	
	Moyenne	*Ecart Type*	*Moyenne*	*Ecart Type*	*Moyenne*	*Ecart Type*
τexec	*125* µs		*175* µs		*173* µs	
Texec	*110* µs	*221* µs	*131* µs	*290* µs	*102* µs	*198* µs

Famille Linux

	Linux 2.2.26		*Linux 2.4.5*		*Linux 2.4.26*		*Linux 2.6.6*	
	Moyenne	*Ecart Type*	*Moyenne*	*Ecart Type*	*Moyenne*	*Ecart Type*	*Moyenne*	*Ecart Type*
τexec	*485* µs		*479* µs		*450* µs		*953* µs	
Texec	*465* µs	*1505* µs	*695* µs	*2499* µs	*670* µs	*2533* µs	*369* µs	*1495* µs

En comparant les Texec correspondants aux dix OSs ciblés, nous trouvons que les systèmes appartenant à la famille Windows ont toujours un temps de réaction inférieur à celui correspondant aux systèmes Linux. Les écarts type sont très grands et toujours supérieurs aux Texec obtenus. Ce qui signifie que les valeurs ont une variation importante autour de la moyenne. Finalement, nous notons que *τexec* de Linux 2.6.6 est approximativement deux fois supérieur à celle des autres versions; en fait, le temps d'exécution de l'appel système *execve* a augmenté de 6000 μs pour les autres versions pour atteindre les 15000 μs dans le cas de Linux 2.6.6.

Le temps de redémarrage du système (Tableau 5-3) montre que le temps de redémarrage en présence de fautes (Tred) est dans tous les cas supérieur au temps de redémarrage en absence de fautes (τred). Toutefois, la différence entre ces deux temps est négligeable. Les écarts type sont assez faibles pour les OSs de la famille Windows alors qu'ils sont assez importants pour les OSs de la famille Linux. Nous notons également que Windows XP a le temps de redémarrage le plus court parmi les OSs de Windows. Pour les systèmes Linux, nous constatons que ce temps augmente au fur et à mesure que les versions (révisions) augmentent, à l'exception de la version 2.6 de Linux qui redémarre plus vite que les autres systèmes de sa famille. Cette exception est justifiée par le fait que cette version de Linux a été restructurée. Globalement, le temps de redémarrage (en absence et en présence de fautes) des systèmes Linux est plus petit que celui des systèmes Windows.

Tableau 5-3 : Temps de redémarrage du système

Famille Windows

	Windows NT4		*Windows 2000*		*Windows XP*	
	Moyenne	*Ecart Type*	*Moyenne*	*Ecart Type*	*Moyenne*	*Ecart Type*
τred	*91 s*		*95 s*		*74 s*	
Tred	*92 s*	*4 s*	*96 s*	*4 s*	*74 s*	*3 s*

	Windows NT4 Server		*Windows 2000 Server*		*Windows 2003 Server*	
	Moyenne	*Ecart Type*	*Moyenne*	*Ecart Type*	*Moyenne*	*Ecart Type*
τred	*90 s*		*111 s*		*76 s*	
Tred	*91 s*	*4 s*	*112 s*	*4 s*	*77 s*	*4 s*

Famille Linux

	Linux 2.2.26		*Linux 2.4.5*		*Linux 2.4.26*		*Linux 2.6.6*	
	Moyenne	*Ecart Type*	*Moyenne*	*Ecart Type*	*Moyenne*	*Ecart Type*	*Moyenne*	*Ecart Type*
τred	*64 s*		*74 s*		*82 s*		*77 s*	
Tred	*71 s*	*32 s*	*79 s*	*23 s*	*83 s*	*24 s*	*82 s*	*27 s*

5.3.2 Raffinement des mesures de l'étalon

Dans ce paragraphe, les trois mesures présentées dans le paragraphe précédent sont raffinées par rapport aux différentes issues de l'OS et à l'état final de l'activité. Ce raffinement a pour but de fournir d'éventuelles explications et de favoriser une meilleure compréhension des valeurs obtenues.

Mesures de robustesse

Comme nous l'avons précédemment mentionné, nous n'avons aucune information sur l'état du système dans les cas de non signalement. Pour cela, il est conseillé de raffiner ces cas en les combinant avec les différents états de l'activité pour mieux constater l'effet de la corruption des appels système. Nous rappelons que les issues de non signalement correspondent à 56% des expériences réalisées sur Windows alors que ce pourcentage varie entre 25% et 31% des expériences réalisées sur Linux.

Les mesures de robustesse conjointes aux différentes issues de l'activité montrent que l'activité finit son exécution dans 91% des cas de non signalement de Windows. Ce pourcentage augmente dans les cas des systèmes Linux pour atteindre les 96%. De plus, si nous classons l'ensemble des expériences menées sur les systèmes d'exploitation selon l'état final de l'activité, nous constatons que dans 94% des cas l'activité finit son exécution sur les systèmes d'exploitation Windows alors que pour les systèmes Linux, l'activité finit son exécution dans 73% à 76% des cas seulement.

Dans le cas d'utilisation de Postmark pour activer le système cible, nous avons la possibilité de détecter la terminaison correcte et incorrecte de l'activité. Le raffinement détaillé des mesures de robustesse par rapport aux différents états de l'activité est présenté dans l'Annexe 2. Nous notons enfin que nous avons réalisé une étude de sensibilité des résultats obtenus pour la technique de corruption des paramètres des appels systèmes. Les résultats détaillés de cette étude de sensibilité sont présentés dans l'annexe 3.

Temps de réaction de l'OS

Nous présentons dans le Tableau 5-4 le temps de réaction détaillé des OSs par rapport aux différentes issues de l'OS après l'exécution d'un appel système corrompu (Code d'erreur, Exception et Non Signalement). Par conséquent, trois moyennes s'ajoutent à la moyenne générale Texec présenté dans le paragraphe précédent. Nous notons également que les moyennes présentées dans ce tableau, relatives aux OSs de la famille Linux, ne prennent pas en compte la durée d'exécution de l'appel système *execve* qui a doublé le $\tau exec$ de Linux 2.6.6 par rapport aux autres versions ciblées de Linux.

Pour les systèmes Windows, nous remarquons que Texec est toujours inférieur à $\tau exec$. Par contre, Texec est supérieur à la moyenne en absence de fautes dans les cas des systèmes Linux. Le Texec le plus faible est enregistré pour Windows 2003. En ne considérant que les trois systèmes Windows ciblés dans le chapitre précédent, nous trouvons que Windows XP détient le Texec le plus petit. Cependant, nous avons vu dans le chapitre précédent que Windows XP avait le temps de réaction le plus rapide par rapport à Windows 2000 et Windows NT4, en présence et en absence de fautes. Dans le cas de Postmark, nous

constatons que Windows XP a un temps de réaction, en absence de fautes, plus grand que ceux des deux autres systèmes. En fait, Postmark active 27 appels système dont la moitié fait partie du composant fonctionnel « systèmes de fichiers » ; le $\tau exec$ relatif à ces appels système est de 574 µs pour Windows XP alors qu'il ne dépasse pas 250 µs pour Windows 2000 et 380 µs pour Windows NT4. Par conséquent, la grande valeur de $\tau exec$ de Windows XP est due aux temps d'exécution des appels systèmes appartenant au composant fonctionnel « système de fichiers » qui semble être un point de faiblesse dans Windows XP.

Tableau 5-4 : Temps détaillé de réaction de l'OS

Famille Windows

	Windows NT4		*Windows 2000*		*Windows XP*	
	Moyenne	*Ecart Type*	*Moyenne*	*Ecart Type*	*Moyenne*	*Ecart Type*
$\tau exec$	248 µs		136 µs		315 µs	
Texec	148 µs	219 µs	118 µs	289 µs	114 µs	218 µs
TEr	45 µs	107 µs	34 µs	61 µs	45 µs	118 µs
TXp	40 µs	15 µs	37 µs	15 µs	50 µs	96 µs
TNS	234 µs	437 µs	186 µs	375 µs	168 µs	265 µs

	Windows NT4 Server		*Windows 2000 Server*		*Windows 2003 Server*	
	Moyenne	*Ecart Type*	*Moyenne*	*Ecart Type*	*Moyenne*	*Ecart Type*
$\tau exec$	125 µs		175 µs		173 µs	
Texec	110 µs	221 µs	131 µs	289 µs	102 µs	198 µs
TEr	41 µs	66 µs	29 µs	33 µs	25 µs	61 µs
TXp	35 µs	15 µs	37 µs	15 µs	48 µs	20 µs
TNS	166 µs	280 µs	210 µs	396 µs	156 µs	252 µs

Famille Linux

	Linux 2.2.26		*Linux 2.4.5*		*Linux 2.4.26*		*Linux 2.6.6*	
	Moyenne	*Ecart Type*	*Moyenne*	*Ecart Type*	*Moyenne*	*Ecart Type*	*Moyenne*	*Ecart Type*
$\tau exec$	156 µs		73 µs		73 µs		74 µs	
Texec	167 µs	300 µs	466 µs	2276 µs	425 µs	2055µs	93 µs	12 µs
TEr	208 µs	361 µs	92 µs	105 µs	84 µs	6 µs	91 µs	10 µs
TXp	88 µs	5 µs	91 µs	8 µs	91 µs	8 µs	106 µs	13 µs
TNS	85 µs	5 µs	1545 µs	4332 µs	1405 µs	3912µs	91 µs	11 µs

Dans le cas de systèmes Windows, et à l'exception des deux OSs *Windows NT4* et *Windows NT4 Server*, le temps nécessaire pour retourner un code d'erreur est inférieur au temps nécessaire pour notifier une exception, ce qui confirme les résultats présentés dans le chapitre précédent. D'autre part, la supériorité des TEr de *Windows NT4* et *Windows NT4 server* par rapport aux correspondantes TXp est due à l'appel système *GetCPInfo* : la corruption de son premier paramètre entraîne une durée d'exécution assez élevée avant de retourner un code d'erreur.

Dans les cas des systèmes Linux, nous remarquons que les temps de réaction des deux révisions de la versions 2.4 dans le cas de non signalement (TNS) sont très grands par rapport à ceux des deux autres systèmes ciblés. En plus, les très grandes valeurs des écarts types correspondants suggèrent une grande variation des valeurs autour de cette moyenne. La Figure 5-2 illustre les temps d'exécution des différents appels système des OSs Linux dont les paramètres corrompus n'ont pas entraîné une réaction de la part de l'OS. Nous constatons clairement que les TNS des deux révisions de la versions 2.4 ont une grande valeur par rapport aux autres versions à cause du temps d'exécution de l'appel système *mkdir*. Son temps d'exécution est approximativement 120 fois plus grand que dans les autres cas.

Un autre point à signaler correspond au TEr de Linux 2.2.26 qui a une très grande valeur par rapport aux TEr des autres versions. Son écart type est également grand. A l'exception de cette révision, tous les autres systèmes Linux ont un TEr plus petit ou égal au temps nécessaire pour retourner une exception. La Figure 5-3 illustre tous les temps de réaction des appels système qui ont retourné des codes d'erreur après la corruption de leurs paramètres. Cette figure montre que les temps d'exécution des appels système sont similaires, à l'exception du temps correspondant à l'appel système *unlink* qui a une grande valeur dans le cas de Linux 2.2.26, ce qui explique la grande valeur de TEr de Linux 2.2.26 par rapport aux autres versions.

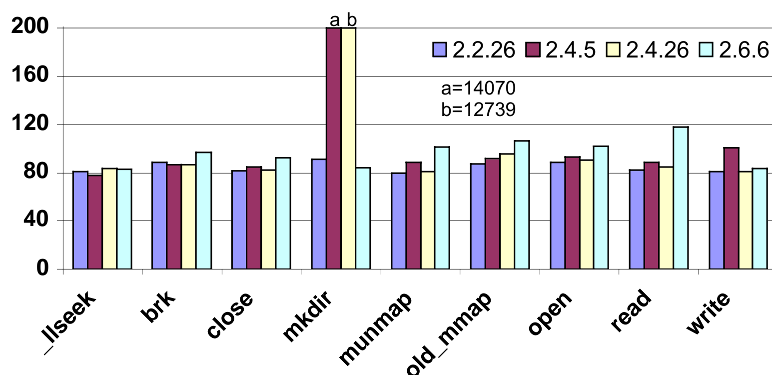


Figure 5-2 : Temps de réaction de Linux, cas de non signalement

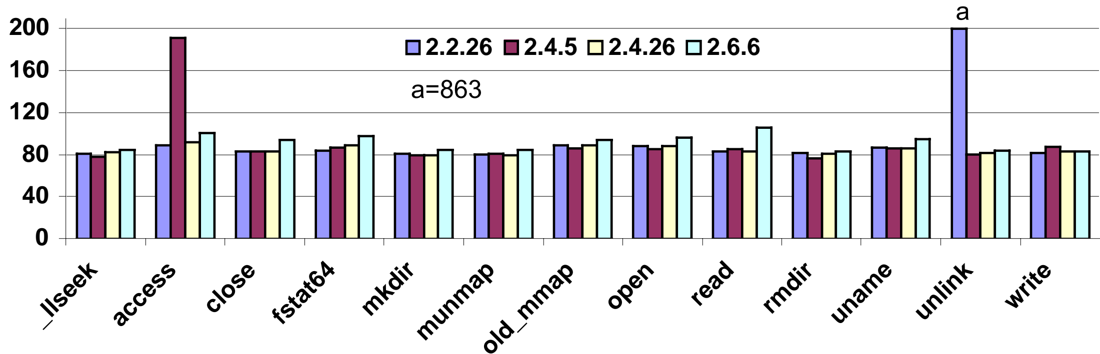


Figure 5-3 : Temps de réaction de Linux, cas de retour de code d'erreur

Notons enfin que nous avons refait le calcul des différentes moyennes en excluant cette fois les appels systèmes qui ont eu des valeurs exceptionnelles (*execve*, *mkdir* et *unlink*). Ce nouveau calcul a conduit à des temps d'exécution Texec équivalents pour les quatre systèmes d'exploitation Linux ciblés.

Temps de redémarrage du système

Malgré la ressemblance entre les temps de redémarrage pour chacun des systèmes en présence et en absence de fautes (avec un avantage pour les systèmes d'exploitation de la famille Linux), plusieurs points restent encore à explorer. Plus particulièrement, nous avons montré dans le chapitre précédent que le temps de redémarrage de Windows est influencé par l'état de terminaison de l'activité.

Dans la Figure 5-4, nous présentons les temps détaillés de redémarrage de deux systèmes d'exploitation chacun appartenant à une des deux familles : Windows NT4 et Linux 2.2.26. Sur cette figure, nous constatons que la distribution des valeurs de temps de redémarrage est très différente pour les deux systèmes.

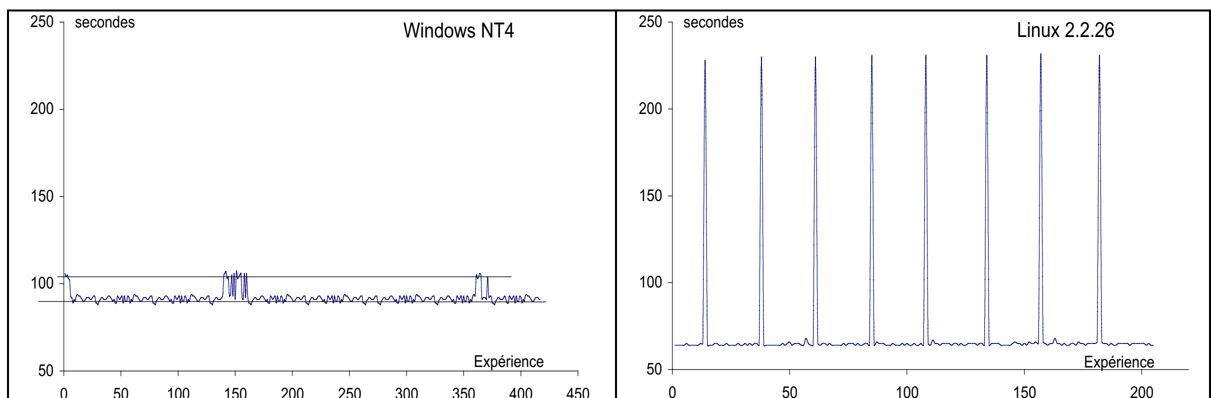


Figure 5-4 : Temps détaillé de redémarrage

Dans le cas de Windows NT4, la même corrélation entre le temps de redémarrage et l'état final de l'activité est retrouvée. En effet, le temps moyen de redémarrage du système après la fin de l'exécution de l'activité est très proche du temps global présenté dans le Tableau 5-3. Quand l'activité est en état de Blocage/Abandon, le système met plus de temps pour redémarrer. Ces deux ensembles d'expériences sont présentés par les deux lignes sur le graphique de redémarrage de Windows NT4 dans la Figure 5-4 : la ligne du haut correspond à la moyenne des temps de redémarrage de Windows NT4 après le Blocage/Abandon de l'activité (105 secondes) alors que la ligne du bas représente la moyenne des temps de redémarrage du système après exécution complète de l'activité (91 secondes).

Cette corrélation n'apparaît pas dans les cas des systèmes d'exploitation Linux : le temps de redémarrage des systèmes Linux n'est pas influencé par l'état final de l'activité. Cependant, la même figure illustre des apparitions périodiques de grandes valeurs de temps de redémarrage pour Linux 2.6.6. Ces valeurs correspondent au « check-disk » que Linux réalise tous les 26 redémarrages de la machine cible.

Finalement, nous notons que toutes les versions de Windows ciblées ont le même comportement. De même, toutes les versions de Linux ont le même comportement que la version 2.2.26. La liste complète des graphiques des temps redémarrage de ces systèmes est présentée dans l'Annexe 4. Tous ces graphiques confirment les résultats détaillés dans ce paragraphe.

5.4 Résultats de DBench-OS-JVM

Après avoir utilisé l'activité de l'étalon de performance *Postmark* pour comparer la sûreté de fonctionnement des systèmes d'exploitation des familles Windows et Linux, nous présentons dans cette section les résultats de l'étalonnage de ces systèmes d'exploitation, en utilisant cette fois le prototype de l'étalon DBench-OS-JVM qui utilise la *machine virtuelle Java* comme activité privilégiée.

5.4.1 Mesures de l'étalon

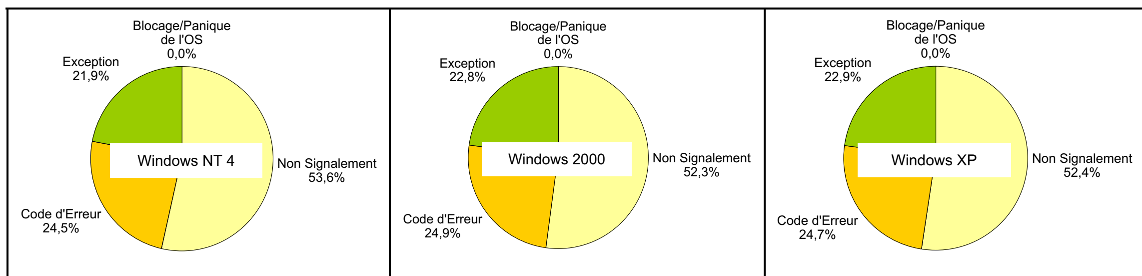
Les mesures de robustesse sont illustrées dans la Figure 5-5. Comme pour toutes les expériences réalisées sur les différentes OSs ciblés, aucun état de Panique ou de Blocage de l'OS n'a été détecté. La corruption des paramètres des appels système a généré des exceptions dans 22% des cas ; le taux de retour de codes d'erreur est de 25%. Les systèmes Windows ciblés n'ont pas signalé la présence de cette corruption dans 53% des cas. Une fois de plus, nous avons une équivalence, en termes de robustesse, des trois systèmes d'exploitation Windows ciblés.

Pour les systèmes Linux, une très légère différence a été enregistrée pour la robustesse des deux révisions 2.4.5 et 2.4.26 de la version 2.4. En effet, les pourcentages de retour de codes d'erreurs, des exceptions et des cas de non signalement ont été très proches pour ces deux révisions dont les résultats de robustesse sont représentés par un seul graphique dans la Figure 5-5. Sur cette figure, nous constatons que les quatre systèmes Linux sont également équivalents en termes de robustesse. Dans 7.8% à 9.7% des cas, les systèmes ont

retourné des exceptions, alors que le taux de retour de code d'erreur varie entre 58.8% et 67.5% des expériences. Les cas de non signalement de l'OS varient entre 24.8% et 31.6%.

Ces mesures valident les résultats obtenus précédemment avec les deux prototypes utilisant le client TPC-C et l'activité de Postmark pour activer les systèmes cibles. Les taux des différentes issues des OSs sont très proches.

Famille Windows



Famille Linux

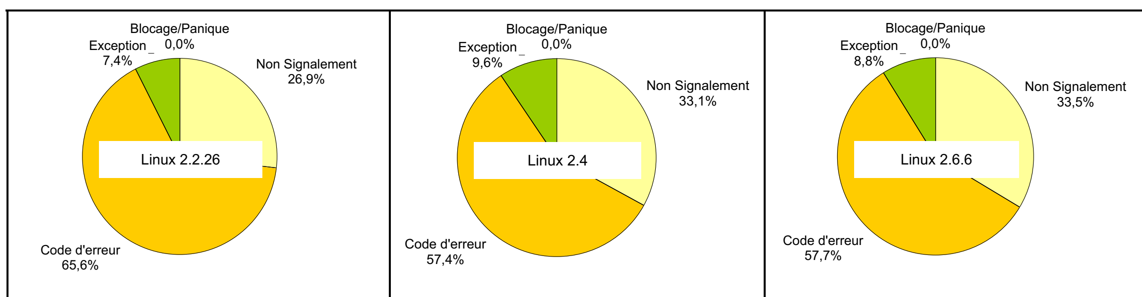


Figure 5-5 : Mesures de robustesse

Les temps de réaction des systèmes sont donnés dans le Tableau 5-5. Les temps de réaction des systèmes Windows montrent que Windows XP est le plus rapide en absence de fautes, alors qu'il est le plus lent en leur présence. Cependant, les *τexec* de Windows NT4 et de Windows 2000 ainsi que le Texec de Windows XP sont largement supérieurs à ceux obtenus dans les expériences réalisées avec Postmark. Une analyse plus détaillée de ces résultats montre que trois appels système sont à l'origine de cette énorme différence : *LoadLibraryA*, *LoadLibraryExA* et *LoadLibraryExW*. En effet, le *τexec* de ces trois appels système atteint les 1942944 µs, 249092 µs et 7381 respectivement pour Windows NT4, Windows 2000 et Windows XP. En excluant ces appels système du calcul, le Texec de Windwos XP diminue considérablement pour atteindre les 218 µs.

Le Tableau 5-5 montre que *τexec* a des valeurs du même ordre de grandeur pour les révisions 2.2.26, 2.4.5 et 2.4.26. Cette valeur est sensiblement inférieure dans le cas de la révision 2.6.6. En plus, les écarts type correspondant aux différents Texec sont assez importants, ce qui traduit une grande variation autour de la moyenne. Nous avons observé que certains appels système (*execve*, *getdents64* et *nanosleep*) ont des durées d'exécution assez grandes par rapport aux autres appels système testés.

Tableau 5-5 :Temps de réaction de l'OS

Famille Windows

	Windows NT4		*Windows 2000*		*Windows XP*	
	Moyenne	*Ecart Type*	*Moyenne*	*Ecart Type*	*Moyenne*	*Ecart Type*
τexec	*44034* µs		*9504* µs		*563* µs	
Texec	*42280* µs	*637986* µs	*306* µs	*1047* µs	*1376971* µs	*49175640* µs

Famille Linux

	Linux 2.2.26		*Linux 2.4.5*		*Linux 2.4.26*		*Linux 2.6.6*	
	Moyenne	*Ecart Type*	*Moyenne*	*Ecart Type*	*Moyenne*	*Ecart Type*	*Moyenne*	*Ecart Type*
τexec	*2951* µs		*2847* µs		*2255* µs		*865* µs	
Texec	*202* µs	*583* µs	*557* µs	*2257* µs	*544* µs	*2223* µs	*505* µs	*2198* µs

Les durées de redémarrage du système sont présentées dans le Tableau 5-6. Nous remarquons que, comme dans le cas de Postmark, le temps de redémarrage sans fautes (*τred*) est toujours inférieur de quelque secondes à la durée moyenne de redémmage en présence de fautes (*Tred).* Comme dans le cas de Postmark, les écarts type pour Windows sont très petits (5 secondes aux maximum) alors qu'ils sont assez élevés pour les systèmes Linux.

La comparaison des *τred* obtenus avec Postmark et JVM montre une certaine différence entre les temps correspondant aux systèmes Windows ; quant aux temps correspondant aux systèmes Linux, ils restent équivalents. En effet, pour les systèmes Windows, les *τred* correspondant aux expériences réalisées avec JVM sont plus petits que ceux obtenus avec Postmark. Cela suggère une dépendence entre le temps de redémarrage du système et l'activité utilisée. Ceci est expliqué par le fait que le temps de redémarrage de la machine à étalonner comprend la durée nécessaire au système pour enregistrer les variables d'environnement actuel du système et les paramètres personnels de l'utilisateur. Pour les deux activités utilisées, Windows XP possède toujours le *τred* le plus petit comme dans les cas du client TPC-C et du Postmark.

Pour les systèmes de la famille Linux, les temps de redémarrage augmentent au fur et à mesure de l'augmentation des versions et des révisions, à l'exception de Linux 2.6.6. Tous ces résultats permettent de confirmer les résultats obtenus avec Postmark.

Nous notons enfin que globalement, comme dans le cas du prototype avec Postmark, le temps nécessaire à Windows pour redémarrer est supérieur à celui nécessaire aux systèmes Linux.

Tableau 5-6 : Temps de redémarrage du système

Famille Windows

	Windows NT4		*Windows 2000*		*Windows XP*	
	Moyenne	*Ecart Type*	*Moyenne*	*Ecart Type*	*Moyenne*	*Ecart Type*
τred	*90 s*		*88 s*		*67 s*	
Tred	*91 s*	*3 s*	*89 s*	*5 s*	*71*	*5 s*

Famille Linux

	Linux 2.2.26		*Linux 2.4.5*		*Linux 2.4.26*		*Linux 2.6.6*	
	Moyenne	*Ecart Type*	*Moyenne*	*Ecart Type*	*Moyenne*	*Ecart Type*	*Moyenne*	*Ecart Type*
τred	*64 s*		*74 s*		*79 s*		*77 s*	
Tred	*71 s*	*37 s*	*79 s*	*23 s*	*83 s*	*24 s*	*82 s*	*27 s*

5.4.2 Raffinement des mesures

Les trois mesures de l'étalon sont raffinées en considérant les différents états de l'activité et les issues de l'OS.

Les mesures de robustesse

Les mesures de robustesse obtenues par combination des issues de l'OS avec les différents états de l'activité montrent que cette dernière finit son exécution dans 92% à 94% des cas de non signalement de Windows. Ce même pourcentage est observé sur les systèmes Linux. Nous avons également classé l'ensemble des expériences menées sur les systèmes d'exploitation uniquement selon l'état final de l'activité. Nous avons constaté que dans 92% des cas, l'activité finit son exécution sur les systèmes d'exploitation Windows. Une fois de plus, le même pourcentage a été enregistré pour les cas de terminaison de l'activité JVM sur les systèmes Linux.

Nous avons réalisé une étude de sensibilité des résultats obtenus pour la technique de corruption des paramètres des appels systèmes. Les résultats de cette étude sont présentés dans l'annexe 3.

Temps de réaction de l'OS

Afin de mieux comparer les temps de réaction des différents systèmes considérés, nous les avons analysés en éliminant les durées d'exécution correspondant aux appels système *LoadLibraryA*, *LoadLibraryExA* et *LoadLibraryExW* des systèmes Windows ainsi que les durées d'exécution des appels système *execve*, *getdents64* et *nanosleep* des systèmes Linux. Le temps de réaction de l'OS, raffiné par rapport aux différentes issues observées de l'OS est présenté dans le Tableau 5-7.

Concernant la famille Windows, nous rappelons que les appels système exclus du calcul de temps de réaction (*LoadLibraryA*, *LoadLibraryExA* et *LoadLibraryExW*) ont eu des comportements normaux lors de l'étalonnage avec DBench-OS-Postmark.

Tableau 5-7 : Temps détaillé de réaction de l'OS

Famille Windows

	Windows NT4		*Windows 2000*		*Windows XP*	
	Moyenne	*Ecart Type*	*Moyenne*	*Ecart Type*	*Moyenne*	*Ecart Type*
τexec	183 µs		278 µs		298 µs	
Texec	501 µs	3792 µs	203 µs	505 µs	218 µs	488 µs
TEr	417 µs	3660 µs	47 µs	110 µs	50 µs	102 µs
TEx	586 µs	4691 µs	84 µs	168 µs	98 µs	209µs
TNS	504 µs	3426 µs	306 µs	587 µs	351 µs	629 µs

Famille Linux

	Linux 2.2.26		*Linux 2.4.5*		*Linux 2.4.26*		*Linux 2.6.6*	
	Moyenne	*Ecart Type*	*Moyenne*	*Ecart Type*	*Moyenne*	*Ecart Type*	*Moyenne*	*Ecart Type*
τexec	1575 µs		1399 µs		765 µs		250 µs	
Texec	88 µs	85 µs	241 µs	1479 µs	227 µs	1438 µs	88 µs	26 µs
TEr	90 µs	101 µs	79 µs	6 µs	84 µs	30 µs	86 µs	8 µs
Tex	87 µs	7 µs	85 µs	8 µs	87 µs	8 µs	98 µs	15 µs
TNS	84 µs	6 µs	572 µs	2545 µs	523 µs	2545 µs	89 µs	43 µs

Dans le Tableau 5-7, nous remarquons que pour les deux systèmes Windows 2000 et Windows XP, TEr est inférieur à TXp qui, à son tour, est inférieur à TNS. Les écarts type correspondants à ces moyennes sont relativement faibles. En revanche, les moyennes des temps de réaction de Windows NT4 échappent à cette règle ; en plus, les écarts type correspondants sont assez importants. Pour expliquer cette exception à la règle générale, nous présentons dans la Figure 5-6 le temps de réaction des systèmes Windows dans les cas de retour d'exceptions. Ainsi, tous les 38 appels système de Windows ainsi que la moyenne d'exécution de chacun d'eux sont présentés sur cette figure. Nous pouvons constater que le temps de réaction de l'appel système `FindNextFileW` est à l'origine des importants écarts type trouvés pour le système Windows NT4. En effet, le temps de réaction de cet

appel système est supérieur à 13400 µs dans le cas de Windows NT4 alors qu'il ne dépasse pas les 75 µs pour les deux autres systèmes, d'où la grande valeur correspondant à TXp pour Windows NT4. Enfin, nous notons que ce même appel système est à l'origine de ces grandes valeurs de moyennes de Windows NT4 dans les cas de retour d'un code d'erreur ou les cas de non signalement (TEr et TNS respectivement).

Pour les systèmes Linux, nous trouvons que les écarts type sont significativement supérieurs à la moyenne pour la version 2.4 du noyau. De plus, il est facilement remarquable pour les deux révisions de 2.4 que les temps de réaction dans le cas de non signalement sont supérieurs à la moyenne générale en présence de fautes. Dans la Figure 5-7, nous présentons les durées d'exécution des appels système de Linux en cas de non signalement. Sur cette figure, nous constatons que, comme dans le prototype précédent utilisant Postmark, une anomalie apparaît au niveau de l'appel système *mkdir*. En effet, cet appel système a une durée d'exécution très grande quand son second paramètre est corrompu.

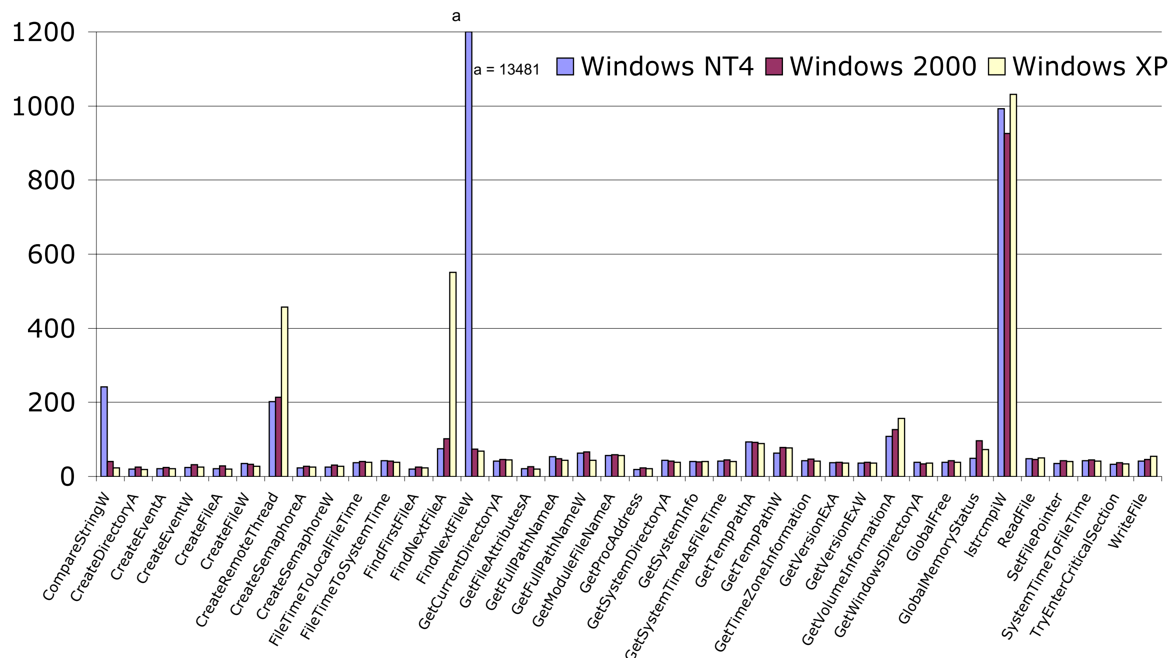


Figure 5-6 : Temps de réaction de Windows, cas de retour d'exception

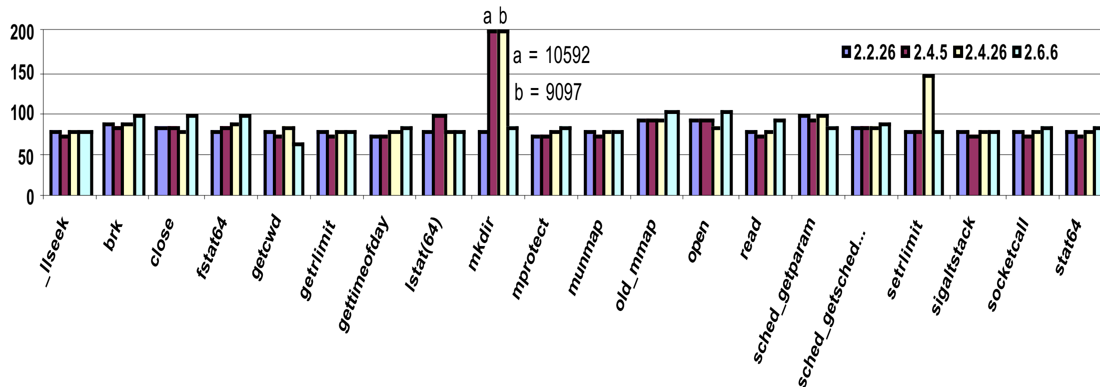


Figure 5-7 : Temps de réaction de Linux, cas de Non Signalement

Temps de redémarrage du système

Le raffinement des temps de redémarrage des systèmes ciblés confirme les résultats obtenus précédemment avec l'activité de Postmark :

- La même corrélation entre le temps de redémarrage et l'état final de l'activité a été observée sur les systèmes Windows : après un blocage/abandon de l'application, le système Windows met plus de temps pour redémarrer, alors qu'après une exécution complète de l'activité, ce temps de redémarrage est proche de la moyenne générale obtenue en absence de fautes.

- Cette corrélation n'existe pas pour les systèmes Linux, où les temps de redémarrage ne varient pas signaficativement à l'exception des cas particuliers correspondant aux activités de « check disk » effectuées par ces systèmes après 26 redémarrages.

5.5 Comparaison

Le Tableau 5-8 résume les **mesures de robustesse** obtenues à partir des expérimentations utilisant les activités *Postmark* et la *machine virtuelle Java* pour trois versions de Linux (2.2.26, 2.4 et 2.6.6) et trois versions de Windows (2000, NT4 et XP).

Tableau 5-8 : Mesures de robustesse pour Linux et Windows

	Dbench-OS-Postmark				***Dbench-OS-Java***			
	Code d'erreur	**Exception**	**Non signalement**	**Blocage/ Panique**	**Code d'erreur**	**Exception**	**Non signalement**	**Blocage/ Panique**
Linux 2.2.26	60,0%	10,9%	29,1%	0,0%	65,6%	7,4%	26,9%	0,0%
Linux 2.4.x	56,4%	10,9%	32,7%	0,0%	57,1%	9,8%	33,1%	0,0%
Linux 2.6.6	50,8%	11,1%	38,1%	0,0%	57,7%	8,8%	33,5%	0,0%
Windows NT4	27,0%	17,5%	55,5%	0,0%	24,5%	21,8%	53,6%	0,0%
Windows 2000	24,5%	20,3%	55,2%	0,0%	24,8%	22,8%	52,3%	0,0%
Windows XP	23,1%	20,7%	56,1%	0,0%	24,7%	22,8%	52,4%	0,0%

Il est important de souligner qu'aucun des systèmes étalonnés n'a enregistré des cas de panique ou de blocage. Ces états finaux sont considérés comme étant les plus graves, puisque dans de tels états, le système d'exploitation n'est plus opérationnel. La restauration d'un état normal de fonctionnement nécessite un redémarrage, ce qui traduit une indisponibilité du système. D'un autre côté, nous observons que les pourcentages de retour de code d'erreur sont environ deux fois supérieurs dans le cas de Linux. En revanche, les pourcentages de levée d'exception sont environ deux fois supérieurs dans le cas de Windows. Les deux études suggèrent que la mise en œuvre des appels système est plus robuste pour Linux que pour Windows. Nous précisons que, lors de raffinement des cas de non signalement de l'OS, nous avons trouvé que les deux activités finissent leurs exécutions dans 90% des cas.

Nous avons également effectué une analyse particulière des expériences relatives aux cas de retour de codes d'erreur avec des données incorrectes, car on pourrait penser que l'OS n'est pas censé détecter de telles situations. Cette analyse a révélé que dans 88 % des cas, les données incorrectes correspondent à des données hors limite dans le contexte particulier des expériences. Par conséquent, le retour de codes d'erreurs dans ces cas est justifié.

Les **mesures temporelles** globales ont montré que les temps de réaction des systèmes Linux sont supérieurs à ceux des systèmes Windows. En plus, les systèmes Linux mettent moins de temps que Windows pour redémarrer, même si des écarts type assez importants sont enregistrés pour les systèmes Linux. D'un autre côté, le raffinement de ces deux mesures temporelles a révélé les points suivants :

- Les temps de réaction des OSs correspondant à une minorité d'appels système peuvent avoir des conséquences considérables sur la moyenne générale calculée. Par exemple, nous

remarquons que pour les temps de réaction des systèmes Windows avec Postmark, les écarts type sont plus petits que pour Linux, ce qui traduit une variation moins importante autour de la moyenne. En effet, pour Windows il n'y a pas de valeur surprenante pour les durées d'exécution en présence de fautes comme c'est le cas pour Linux (par exemple l'appel système `mkdir`). D'un autre côté, dans le cas de l'étalonnage avec l'activité JVM, les systèmes d'exploitation des deux familles Windows et Linux ont des appels système dont les temps d'exécution étaient très supérieurs par rapport aux temps d'exécution du reste des appels système. La même observation est signalée dans le chapitre quatre avec l'activité TPC-C. Une fois que les durées d'exécution de ces appels système sont écartées du calcul de la moyenne générale, nous trouvons que les durées d'exécution du reste des appels système ont le même ordre de grandeur pour Linux et Windows (de quelques dizaines à quelques centaines de microsecondes).

En ce qui concerne les durées de redémarrage, la plus courte a été observée pour Windows XP dans la famille Windows, et pour la révision 2.2.26 dans la famille Linux. Les écarts type enregistrés pour les temps de redémarrage des systèmes Windows sont très faibles, alors qu'ils sont plus importants dans les cas des systèmes Linux. Ceci est dû au temps nécessaire au système Linux pour réaliser l'opération de vérification du disque (check-disk) après 26 redémarrages ; le temps de redémarrage dans ce cas est trois fois supérieur aux temps de redémarrage dans les autres cas. Nous avons vu que l'état final de l'activité n'a pas d'influence sur la durée de redémarage dans le cas de Linux. En revanche, une analyse détaillée des durées de redémarrage pour Windows a fait apparaître une corrélation entre les durées de redémarrage et les états finaux de l'activité. Lors du blocage de l'activité, les durées de redémarrage sont supérieures d'environ 10 % aux cas de terminaison de l'activité.

5.6 Conlcusion

Nous avons présenté dans ce chapitre les résultats de l'étalonnage de la sûreté de fonctionnement des systèmes d'exploitation appartenant à deux familles distinctes : Windows et Linux. Pour atteindre notre objectif, nous avons utilisé deux étalons de sûreté de fonctionnement basés respectivement sur les activités Postmark et la machine virtuelle Java pour solliciter les systèmes cibles.

Après avoir présenté la mise en œuvre de ces étalons, nous avons exposé dans un premier temps les résultats d'étalonnage obtenus par l'étalon DBench-OS-Postmark. Ces résultats ont indiqué une équivalence en termes de robustesse entre les différents systèmes d'exploitation de chacune des familles étudiées. Pour les deux familles, aucun cas de panique ou de blocage de l'OS n'a été observé. Les systèmes d'exploitation Linux ont levé plus de codes d'erreur que les systèmes Windows, alors que ces derniers ont levé plus d'exceptions.

Les temps de réaction des OSs correspondant à une minorité d'appels système sont la raison de la grande différence observée entre les différents systèmes d'exploitation ciblés. La grande majorité des appels système interceptés ont des temps de réaction du même ordre de grandeur. Les résultats obtenus ont permis l'identification de points qui nécessitent une

attention particulière de la part des développeurs des systèmes d'exploitation considérés. Par exemple, dans le cas de la version Linux 2.4, l'appel système `mkdir` a une durée d'exécution inacceptable après corruption de son second paramètre. Ceci est également confirmé pour les OSs de la famille Windows, où des points de faiblesse de cette nature ont été identifiés dans le chapitre 4.

Nous n'avons pas trouvé de grands écarts entre le temps de redémarrage des systèmes cibles en présence et en absence de fautes. Sur ce point, les systèmes Linux semblent être plus avantageux. Nous notons que dans le chapitre 4, une corrélation entre l'état final de l'activité et le temps de redémarrage du système à étalonner a été observée pour les systèmes Windows.

Nous avons ensuite présenté les résultats relatifs à l'étalon DBench-OS-JVM. Des résultats similaires ont été obtenus pour les différents systèmes d'exploitation ciblés. Comme ils sont semblables dans les deux cas, nous pouvons conclure que l'activité d'étalonnage n'a pas un impact significatif sur les mesures proposées.

Conclusion générale

Les travaux présentés dans ce mémoire s'inscrivent dans le cadre de l'étalonnage de la sûreté de fonctionnement des systèmes informatiques en général, et des systèmes d'exploitation (OS) en particulier. Le but de l'étalonnage de la sûreté de fonctionnement est de fournir des moyens génériques et reproductibles pour la caractérisation du comportement des systèmes informatiques en présence de fautes. Un étalon de sûreté de fonctionnement se distingue des techniques d'évaluation et de validation existantes par le consensus qu'il doit assurer entre les différentes communautés qui l'utilisent.

Dans le cadre du projet européen DBench, nous avons défini un cadre conceptuel qui permet de définir des étalons de sûreté de fonctionnement. Ce cadre concerne une catégorie de systèmes allant des logiciels *COTS* comme les systèmes d'exploitation aux systèmes complexes à base de *COTS*. Il s'inspire en grande partie des techniques expérimentales de caractérisation de la sûreté de fonctionnement et des techniques d'évaluation de performance explorées dans l'état de l'art. Dans ce cadre, nous avons identifié les principales dimensions nécessaires à la définition d'un étalon de sûreté de fonctionnement. Ces dimensions décrivent : 1) le système cible et le contexte d'étalonnage, 2) les mesures souhaitées de l'étalon, et enfin, 3) les expérimentations à réaliser. Pour assurer le consensus qui distingue l'étalonnage par rapport aux autres méthodes de caractérisation de sûreté de fonctionnement, nous avons identifié l'ensemble des propriétés qu'un étalon doit vérifier. Ces propriétés doivent être prises en considération depuis les premières phases de spécification jusqu'à la mise en œuvre du prototype de l'étalon.

Basés sur le cadre conceptuel, nous avons spécifié trois étalons de sûreté de fonctionnement destinés à comparer la sûreté de fonctionnement des systèmes d'exploitation de familles différentes. Cette spécification a été réalisée tout en prenant en considération l'ensemble des propriétés que l'étalon doit satisfaire.

Ces trois étalons (dénommés DBench-OS-TPCC, DBench-OS-Postmark et DBench-OS-JVM) se distinguent essentiellement par leur profil d'exécution. Ils utilisent respectivement le client *TPC-C*, *Postmark* et la *machine virtuelle Java* comme activités privilégiées. L'ensemble de fautes utilisées dans chacun de ces étalons dépend des appels système sollicités par l'activité, tandis que la technique de *substitution sélective* des paramètres de ces appels système reste invariable.

L'ensemble de mesures fournies par ces trois étalons est constitué essentiellement par des mesures de robustesse et des mesures temporelles. Les mesures de robustesse correspondent aux différentes issues de l'OS après la corruption de paramètres d'un appel

système. Les mesures temporelles contiennent le temps de réaction de l'OS (la moyenne des temps d'exécution des appels système corrompus) et le temps moyen de redémarrage du système après chaque expérience. Ensuite, nous avons proposé des affinements possibles pour ces mesures.

Nous avons développé un prototype de l'étalon DBench-OS-TPCC pour comparer la sûreté de fonctionnement des OSs de la famille Windows. Nous avons porté le prototype développé sur trois OSs de cette famille : Windows NT4, Windows XP et Windows 2000. Pour donner plus de confiance dans les résultats obtenus, nous avons utilisé la même plate-forme expérimentale pour exécuter les activités d'étalonnage des trois OSs ciblés. Les résultats obtenus ont révélé que les trois OSs étaient équivalents en termes de robustesse. En revanche, Windows XP détient des temps de réaction et de redémarrage plus courts que ceux de Windows NT4 et Windows 2000, en présence ainsi qu'en absence de fautes. Ces résultats ne contredisent pas les déclarations du concepteur de cette famille de systèmes d'exploitation (Microsoft).

Nous avons développé des prototypes des deux étalons DBench-OS-Postmark et DBench-OS-JVM dans le but de comparer la sûreté de fonctionnement des OSs des familles Windows et Linux. Cette comparaison est basée sur les éléments suivants : 1) la même activité a été utilisée pour solliciter les différents systèmes cibles, 2) la même technique d'injection de fautes (substitution sélective des paramètres de tous les appels systèmes utilisés) a été appliqué, et enfin 3) la même conduite d'expérimentation a été réalisée. De plus, la même plate-forme matérielle a été utilisée pour étalonner les OSs ciblés.

Aucun des systèmes étalonnés n'a enregistré un cas de blocage ou de panique. Les résultats obtenus ont révélé une équivalence, en termes de robustesse, entre les OSs de la même famille. Les pourcentages de retour de code d'erreur sont environ deux fois supérieurs pour Linux que pour Windows alors que, inversement, les pourcentages de levée d'exception sont environ deux fois supérieurs dans le cas de Windows par rapport à Linux. Globalement, les résultats des deux étalons montrent que les appels système Linux sont plus robustes que ceux de Windows.

Les mesures temporelles ont montré que les temps de réaction des systèmes Linux sont supérieurs à ceux des systèmes Windows. En revanche, les systèmes Linux nécessitent moins de temps que Windows pour redémarrer. L'affinement de ces mesures a révélé les points suivants :

- Les temps de réaction des OSs correspondant à une minorité des appels système peuvent augmenter considérablement la moyenne générale du temps de réaction de l'OS.
- Pour les systèmes Windows, une corrélation existe entre les durées de redémarrage et les états finaux de l'activité : après le blocage ou l'abandon de l'activité, ces systèmes mettent plus de temps pour redémarrer.

Nous avons vérifié que nos étalons satisfont les propriétés définies dans le cadre conceptuel. Certaines de ces propriétés sont directement vérifiées par construction en montrant que la spécification des étalons les respecte et les garantit. Nous avons également

vérifié par expérimentation les autres propriétés de l'étalon (comme la répétitivité, la portabilité, l'interférence…). Etant donné que l'ensemble de fautes retenu peut avoir un impact important sur les résultats obtenus, nous avons mis l'accent sur la sensibilité de ces résultats par rapport à l'ensemble de fautes utilisé. Tous les résultats obtenus avec des ensembles différents de fautes ont confirmé l'équivalence des systèmes d'exploitation d'une même famille en termes de robustesse.

Les principaux axes de recherche développés tout au long de ce mémoire sont susceptibles de donner lieu à de nombreuses extensions qui nous semblent fort intéressantes.

À court terme, nous retiendrons les voies suivantes :

- Les travaux expérimentaux présentés dans ce mémoire ont traité l'étalonnage de la sûreté de fonctionnement des OSs des deux familles Windows et Linux. Il serait intéressant de développer des étalons de sûreté de fonctionnement d'autres familles d'OSs, comme ceux de Mac OS.
- Pour activer nos systèmes, nous avons utilisé des activités des étalons de performance (Postmark, TPC-C) et une activité réelle (JVM). Il peut être intéressant de vérifier si le développement d'une activité spécifique aux étalons de sûreté de fonctionnement des OSs serait préférable. Cette activité devrait permettre de solliciter les composants fonctionnels communs aux différents OSs. Elle devrait être également portable sur les différents OSs.

À long terme, plusieurs axes de recherche peuvent être explorés :

- L'affinement des mesures temporelles a révélé la présence d'appels système qui modifient considérablement le temps de réaction du système. Il serait intéressant de chercher la cause de ces anomalies. Ceci suppose une analyse détaillée du code source de ces appels système. Cette information peut être intéressante pour les concepteurs de l'OS, qui pourraient éventuellement trouver un moyen, si nécessaire, pour réduire les temps d'exécution de telles fonctions. Cependant, il pourrait s'avérer que ces fonctions, par nature, nécessitent plus de temps que les autres, et leur temps d'exécution ne peut pas être réduit.
- D'autres mesures de sûreté de fonctionnement ou de performance en présence de fautes peuvent élargir l'ensemble de mesures fournies par nos étalons. L'exemple que nous citons concerne le risque de propagation d'erreurs entre processus à travers le système d'exploitation, qui ne communiquent pas d'une manière explicite mais qui s'exécutent sur le même OS. Ce type de mesures est intéressant dans le cas où deux processus de niveaux d'intégrité différents se partageraient les mêmes ressources (structures de données, tables, files d'attente, etc.) du système d'exploitation. Il s'agit d'évaluer l'impact d'un processus erroné de niveau d'intégrité bas sur le comportement d'un processus de haut niveau d'intégrité. Cette mesure constitue une forme de caractérisation de la capacité de l'OS à contenir de telles erreurs. Dans ce cadre, il

serait intéressant d'identifier le type de fautes qui sont susceptibles de se propager et évaluer la robustesse de l'application critique par rapport à ce type de fautes.

- Les travaux menés tout au long de ce mémoire traitent de l'étalonnage de la sûreté de fonctionnement des systèmes d'exploitation par rapport aux comportements défectueux du niveau applicatif. Il serait intéressant d'étalonner la sûreté de fonctionnement de ces systèmes par rapport aux comportements défectueux du matériel utilisé, ou par rapport aux fautes de l'opérateur interagissant avec le système pour des raisons opérationnelles ou de maintenance.
- Nos travaux peuvent être étendus à l'étalonnage vis-à-vis des malveillances. Il s'agirait de caractériser la vulnérabilité du système à travers l'analyse de son comportement suite à des malveillances, comme des tentatives d'intrusion ou des attaques en déni de service.
- Enfin, les étalons développés concernent un composant *COTS* logiciel, et sont basés essentiellement sur l'expérimentation. Ces travaux peuvent s'étendre à l'étalonnage d'un système à base de *COTS* matériels ou logiciels. Dans ce cas, l'étalonnage peut être basé à la fois sur une modélisation analytique du système, décrivant son comportement en présence de fautes, et complétée par des évaluations expérimentales de certains paramètres du modèle (qui sont en fait des mesures expérimentales).

Annexe 1 Exemple d'implémentation des mécanismes de substitution de paramètres et d'observation

Nous présentons dans cette annexe un exemple d'implémentation des mécanismes de substitution de paramètres et d'observation de la fonction Win32 `ResumeThread` dans l'outil *Detours*. Nous rappelons que cet outil est développé en langage C.

Dans cet exemple, le code original de *Detours* est présenté en ***gras italique***. Le code responsable de la substitution de paramètres est écrit en **gras**. Les mécanismes d'observation de l'issue de l'OS sont présentés en mode *italique*.

```
DWORD  __stdcall Mine_ResumeThread (HANDLE A0)
{
occurrence_ResumeThread ++ ;
_PrintEnter("ResumeThread (%lx)\n", a0) ;
if (occurrence_experience==occurrence_ResumeThread)
       {
       a0=*(Pointer(NumValue)) ;
       }
DWORD rv = 0 ;
DWORD LastError = 0, LastException = 0 ;
LARGE_INTEGER Start, End, Frequency ;
Start.QuadPart = 0 ;
End.QuadPart = 0 ;
Frequency.QuadPart=0 ;
__try
       {
       __try
              {
              QueryPerformanceCounter(&Start) ;
              rv = Real_ResumeThread(a0) ;
              QueryPerformanceCounter(&End) ;
              }
       __except(1)
              {
              QueryPerformanceCounter(&End) ;
              LastError = GetLastError() ;
```

```
                LastException = GetExceptionCode() ;
                }
        }

__finally
        {
        double Time = 0 ;
        char *TimeString ;
        int decimal, sign ;
        QueryPerformanceFrequency(&Frequency) ;
        Time = (End.QuadPart-Start.QuadPart)/Frequency.QuadPart ;
        TimeString = _fcvt (time, 6, &decimal, &sign) ;
        _PrintEnter(˝ResumeThread execution time : %s˝, TimeString) ;
        LastError = GetLastError() ;
        _PrintExit(˝ResumeThread()-> %lx˝, rv) ;
        }
return rv ;
}
```

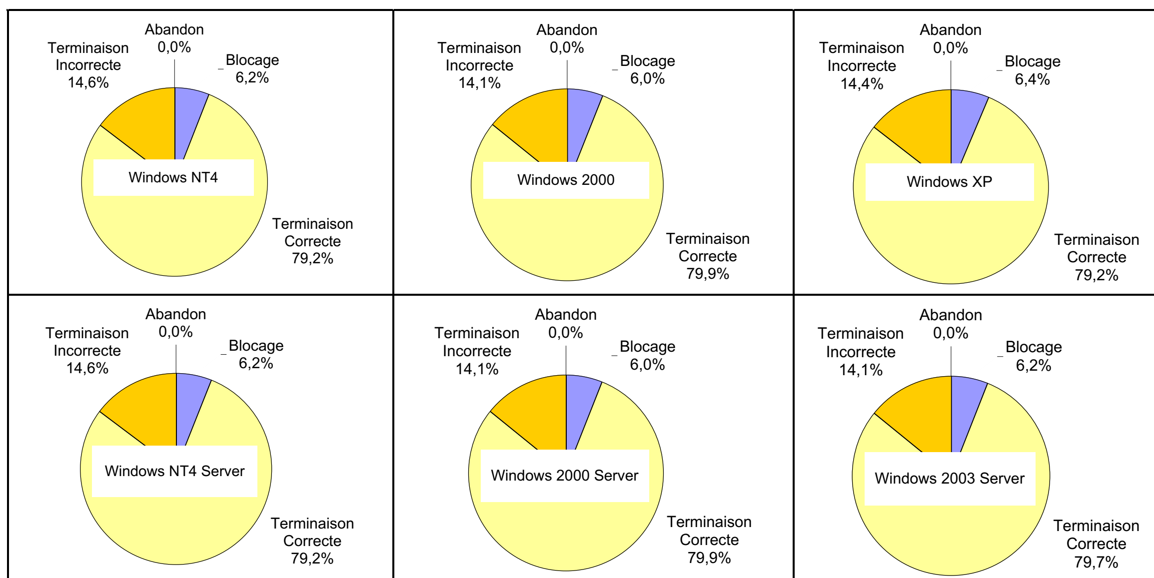
Annexe 2 Raffinement des mesures de robustesse par rapport à l'état final de *Postmark*

Dans le troisième chapitre, nous avons mentionné que les mesures de robustesse de l'étalon peuvent être raffinées en considérant l'état final de l'activité après chaque expérience. Ainsi, il est possible de connaître l'influence de la corruption d'un appel système sur l'exécution de l'activité.

Contrairement au client *TPC-C* et à la *JVM*, le déterminisme de l'activité *Postmark* permet le diagnostic de son état à la fin d'une expérience. Nous présentons dans cette annexe le raffinement des mesures de robustesse obtenues par DBench-OS-Postmark. Nous rappelons que nous avons déterminé quatre états pour caractériser l'activité à la fin de chaque expérience. Ces états correspondent à la terminaison correcte, terminaison incorrecte, abandon ou blocage de l'activité. Les résultats de cette étude sont présentés dans la Figure A2-1. Cette figure montre que, pour chacune des deux familles, les versions considérées ont des comportement similaires. Nous constatons que, pour les systèmes Windows, l'activité termine correctement son exécution dans plus de cas que les systèmes Linux. Aucun état d'abandon de l'activité n'a été constaté pour les systèmes de la famille Windows.

Dans le chapitre 3, nous avons mentionné que le vecteur de *robustesse de l'activité* (PNS) fournit des informations sur l'état de l'activité dans le cas de non signalement de l'OS (cf. section 3.2.2.2). Les vecteurs PNS des différentes versions ciblées de Windows et de Linux sont présentés dans le Tableau A2-1.

Famille *Windows*



Famille *Linux*

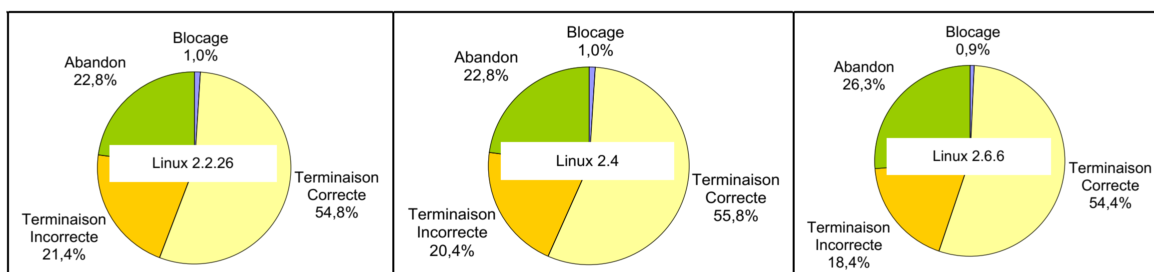


Figure A2-1 : Etats finaux de l'activité *Postmark*

	Terminaison Correcte	Terminaison Incorrecte	Abandon	Blocage
Windows NT4	82,4 %	8,6 %	0 %	9,1 %
Windows NT4 Server	82,4 %	8,6 %	0 %	9,1 %
Windows 2000	82,8 %	8,4 %	0 %	8,8 %
Windows 2000 Server	82,8 %	8,4 %	0 %	8,8 %
Windows XP	82,4 %	8,4 %	0 %	9,2 %
Windows 2003 Server	82,2 %	8,5 %	0 %	9,3 %
Linux 2.2.26	88,2 %	9,8 %	2 %	0 %
Linux 2.4	90,7 %	7,4 %	1,9 %	0 %
Linux 2.6.6	91,7 %	5,5 %	2,8 %	0 %

Tableau A2-1 : les vecteurs de robustesse de l'activité (PNS)

Annexe 3 Validation des résultats de DBench-OS-Postmark et DBench-OS-JVM par rapport à l'ensemble de fautes

Nous présentons dans cette annexe une étude de sensibilité des résultats des deux étalons DBench-OS-Postmark et DBench-OS-JVM par rapport à la nature de valeurs de substitution des paramètres des appels système. Pour cela, nous avons évalué la robustesse de la famille Windows et de la famille Linux en prenant l'ensemble *F0+F1+F2*. Les résultats obtenus ont montré que les différentes versions de la même famille sont équivalentes en termes de robustesse. Nous rappelons que dans *F0*, les valeurs des paramètres sont substituées par des données hors-limites. *F1* correspond aux cas où les valeurs des paramètres sont substituées par des adresses incorrectes, et *F2* aux cas où les valeurs sont substituées par des données incorrectes (cf. section 4.5.3.4 du chapitre 4).

Nous avons réalisé une première étude de sensibilité des résultats obtenus avec DBench-OS-Postmark, vis-à-vis de la technique de corruption de paramètres. Les résultats obtenus en appliquant les ensembles *F0* et *F0+F1* séparément montrent l'équivalence du point de vue de robustesse des versions considérés de Windows, et l'équivalence des versions de Linux.

Une étude similaire a été effectuée avec les résultats de DBench-OS-JVM. Elle montre que les trois versions de Windows sont équivalentes en termes de robustesse en utilisant *F0*, *F0+F1* et *F0+F1+F2*. De la même façon, les versions de Linux sont équivalentes en termes de robustesse avec ces ensembles.

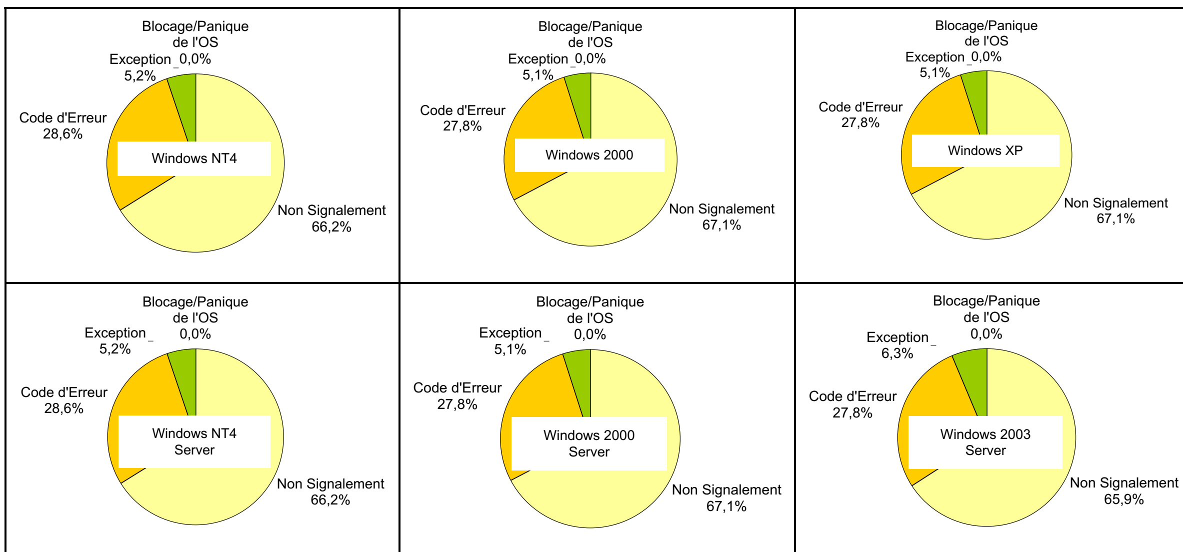
A titre d'exemple, nous présentons sur la Figure A3-1 les résultats de l'application des valeurs hors-limite sur les différents OSs (ensemble de fautes *F0*) pour Postmark. Le nombre des expériences est réduit à 77 expériences (au lieu de 418) pour Windows et à 55 expériences (au lieu de 206) pour Linux. Les résultats obtenus valident les résultats de la Figure 5.1, dans la mesure où ils montrent l'équivalence des systèmes de la même famille du point de vue de robustesse.

La Figure A3-2 illustre la robustesse des six systèmes d'exploitation des familles Windows et Linux, après l'applicatoin de l'ensemble de valeurs hors-limites (*F0*) pour JVM. Cette figure montre l'équivalence, en termes de robustesse, entre les différents systèmes de la même famille d'OSs. Nous notons que le nombre d'expériences réalisées sur Windows est

passé de 1295 (Figure 5.5) à 264 expériences, et que sur les 457 expériences réalisées sur les systèmes Linux, seules 119 expériences ont été retenues en utilisant *F0*.

Enfin, nous avons constaté que le comportement des systèmes d'exploitation d'une même famille (Windows ou Linux) reste invariable avec les deux activités Postmark et JVM. A titre d'exemple, nous constatons sur les deux Figures A3-1 et A3-2 qu'aucun cas de blocage ou de panique n'a été signalé pour les deux familles. Sur ces deux figures, le taux de non signalement a été identique pour les systèmes de chacune des familles ciblées. Le taux de retour de code d'erreur et le taux de levée d'exception n'ont pas varié significativement pour les OSs d'une même famille.

Famille *Windows*



Famille *Linux*

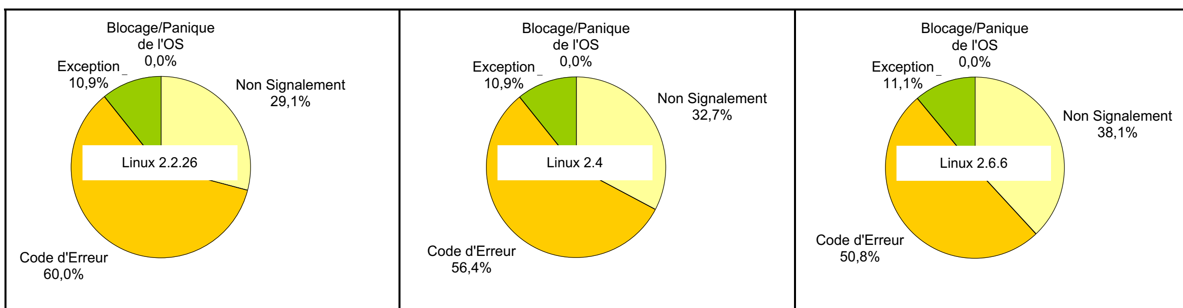
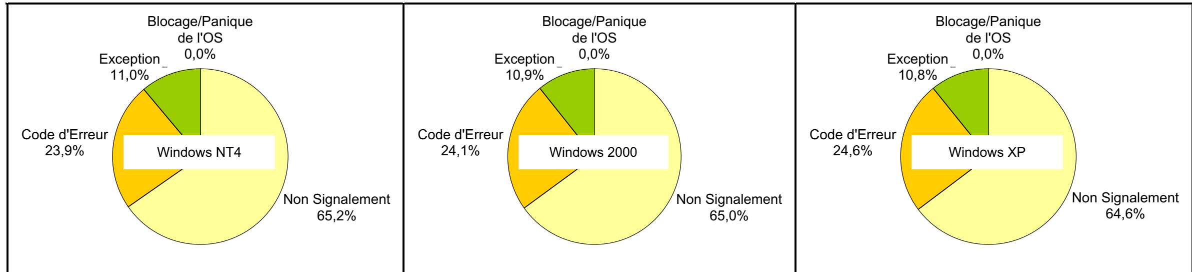


Figure A3-1 : Mesures de robustesse avec Postmark, valeurs hors-limites uniquement (F0)

Famille *Windows*



Famille *Linux*

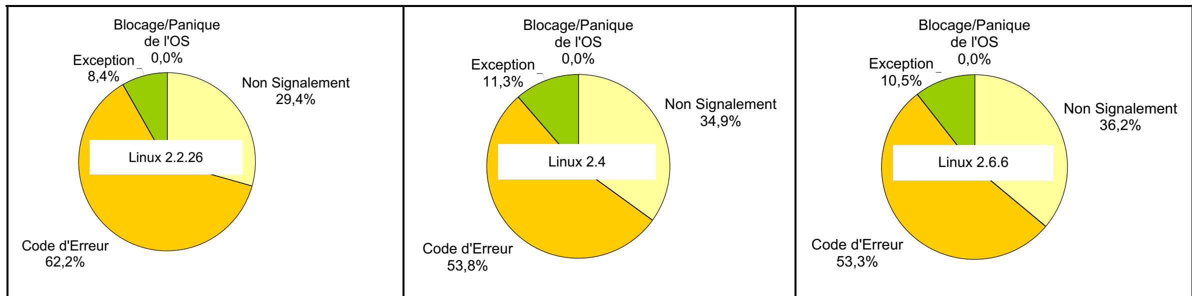
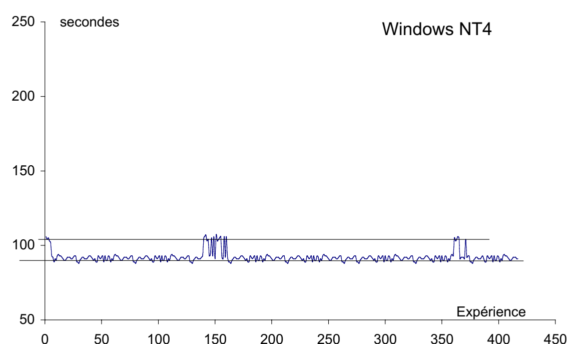
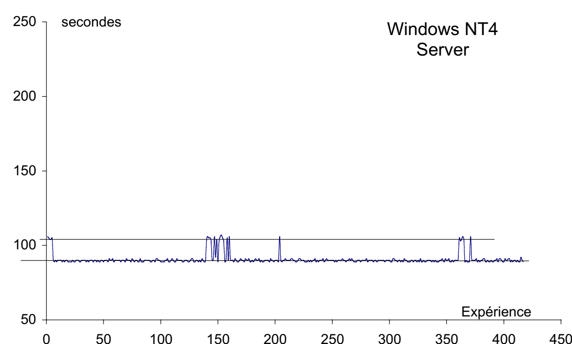


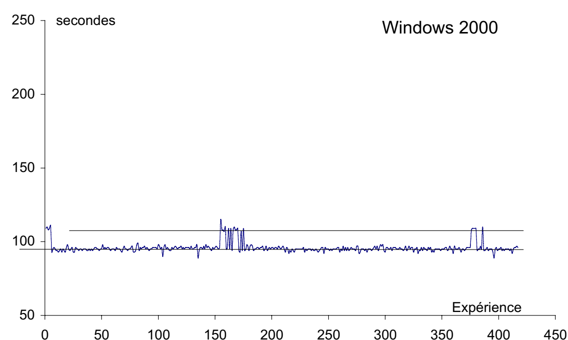
Figure A3-2 : Mesures de robustesse avec JVM, valeurs hors-limites uniquement (F0)

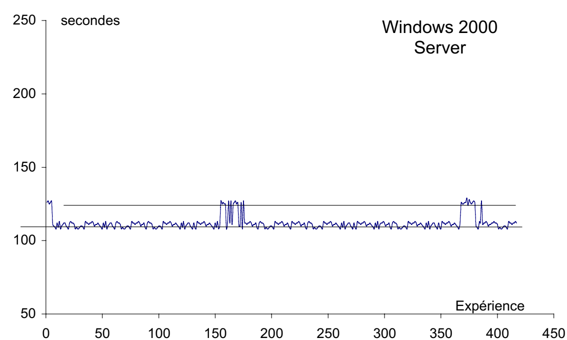
Annexe 4 Temps de redémarrage de Windows et de Linux avec DBench-OS-Postmark

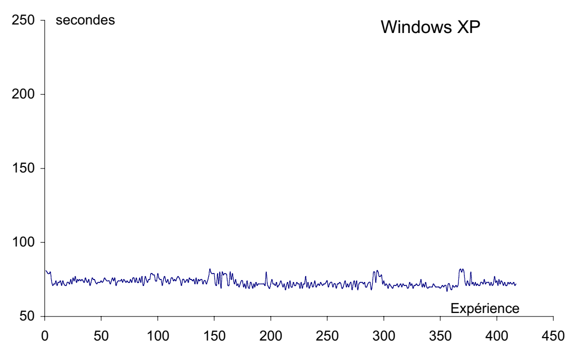
Famille *Windows*

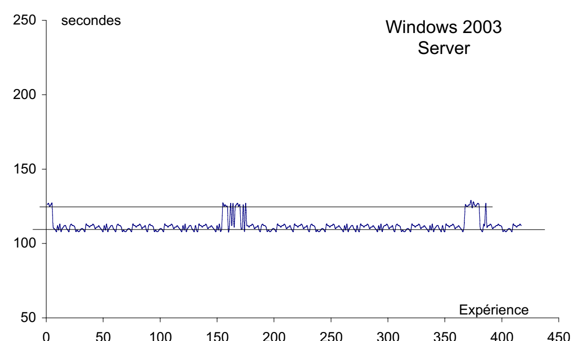




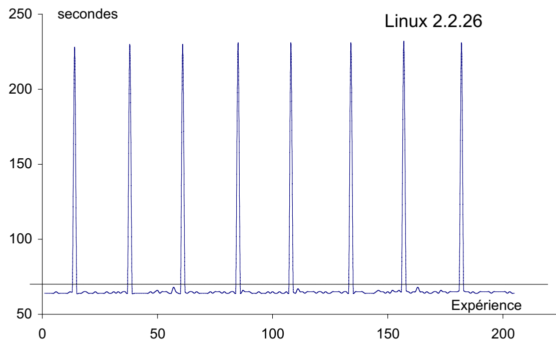


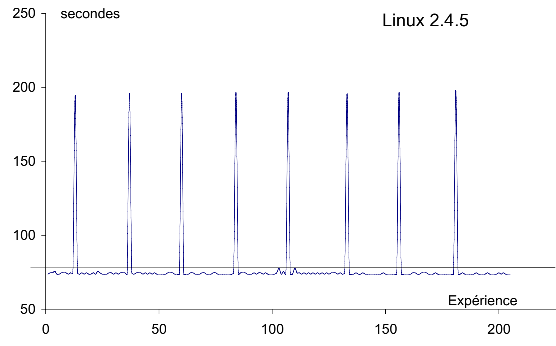


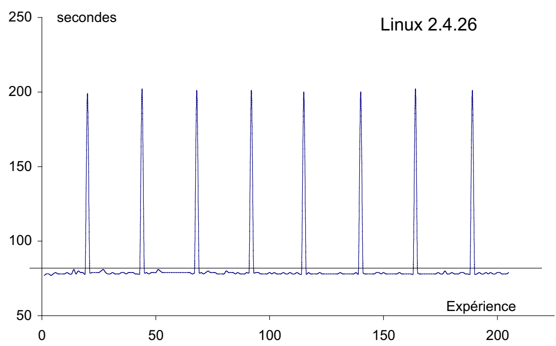


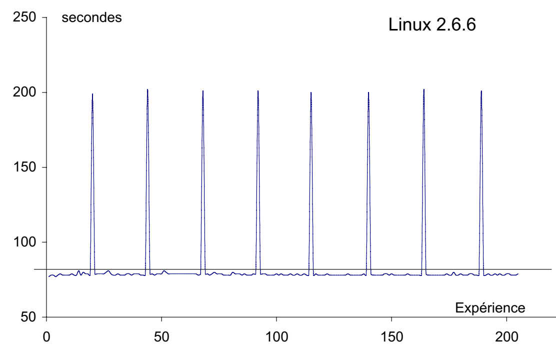


Famille *Linux*









Références bibliographiques

[Accetta 1986] M.J. Accetta, R.V. Baron, W. Bolosky, Golub D.B., R.R. Rashid, A. Tevanian et M.W. Young, "MACH: a new foundation for UNIX development", dans *Summer USENIX 1986 Technical Conference,* (Atlanta, Georgia), pp.93-112, Juillet 1986.

[Albinet 2004] A Albinet, J Arlat et J.-C. Fabre, "Characterization of the Impact of Faulty Drivers on the Robustness of the Linux Kernel", dans *Int. Conf. on Dependable Systems and Networks (DSN 2004),* (Florence, Italy), pp.867-876, IEEE Computer Society Press, Juin 2004.

[Arlat 1989] J. Arlat, Y. Crouzet et J.-C. Laprie, "Fault Injection for Dependability Validation of Fault-Tolerant Computing Systems", dans *Proc. 19th Int. Symp. on Fault-Tolerant Computing (FTCS-19),* (Chicago, IL, USA), pp.348-355, IEEE Computer Society Press, Juin 1989.

[Arlat 1990] J. Arlat, M. Aguera, L. Amat, Y. Crouzet, J.-C. Fabre, J.-C. laprie, E. Martins et D. powell, "Fault Injection for Dependability Validation — A Methodology and Some Applications", *IEEE Transactions on Software Engineering*, Volume 16 (2), pp.166-182, Février 1990.

[Arlat 1999] J. Arlat, J. Boué et Y. Crouzet, "Validation-based Development of Dependable Systems", *IEEE Micro*, 19 19 (4), pp.66-79, Juillet-Août 1999.

[Arlat 2002a] J. Arlat et Y. Crouzet, "Faultload Representativeness for Dependability Benchmarking", dans *Supplement Int. Conference on Dependable Systems and Networks (DSN-2002),* (Washington, DC, USA), Juin 2002.

[Arlat 2002b] J. Arlat, J.-C. Fabre, M. Rodríguez et F. Salles, "Dependability of COTS Microkernel-Based Systems", *IEEE Transactions on Computers*, Volume 51 (2), pp.138-163, Février 2002.

[Barton 1990] J.H. Barton, E.W. Czeck, Z.Z. Segall et D.P. Siewiorek, "Fault Injection Experiments Using FIAT", *IEEE Transactions on Computers*, Volume 39 (4), pp.575-582, Avril 1990.

[Brown 2002] A. Brown, *Availability Benchmarking of a Database System*, EECS Computer Science Division, University of California at Berkley, 2002.

[Brown 2000] A. Brown et D.A. Patterson, “Towards Availability Benchmarks: A Cases Study of Software RAID Systems”, dans *Proc. 2000 USENIX Annual Technical Conference,* (San Diego, CA, USA), USENIX Association, Juin 2000.

[Brown 1997] A. Brown et M. Seltzer, “Operating System Benchmarking in the wake of Lmbench: A Case Study of the Performance of NetBSD on the Intel x86 Architecture”, dans *Proc. of the 1997 ACM SIGMETRICS Conference on the Measurements and Modeling of Computer Systems,* (Seattle, WA, USA), Juin 1997.

[Bryant 2001] R. Bryant, D. Raddatz et R. Sunshine, “PenguinoMeter: A New File-I/O Benchmark for Linux”, dans *5th Annual Linux Showcase and Conference,* (Oakland, California), 5-10 Novembre 2001.

[Bwinstone] Bwinstone, VeriTest, http://www.veritest.com/benchmarks/bwinstone.

[Carreira 1998] J. Carreira, H. Madeira et J.G. Silva, “Xception: A Technique for the Experimental Evaluation of Dependability in Modern Computers”, *IEEE Transactions on Software Engineering*, Volume 24 (2), pp.125-136, Février 1998.

[Chevochot 2001] P. Chevochot et I. Puaut, “Experimental Evaluation of the Fail-Silent Behavior of a Distributed Real-Time Run-Time Support Built from COTS Components”, dans *Proc. Int. Conference on Dependable Systems and Networks (DSN-2001),* (Göteborg, Sweden), pp.304-313, IEEE CS Press, Juillet 2001.

[Chillarege 1992] R. Chillarege, I. S. Bhandari, J. K. Chaar, M. J. Halliday, D. S. Moebus, B. K. Ray et M . Y. Wong, “Orthogonal Defect Classification - A Concept for In-Process Measurements”, *IEEE Transactions of software engineering*, Volume 18 (11), pp.943-956, Novembre 1992.

[Choi 1992] G.S. Choi et R.K. Iyer, “FOCUS: An Experimental Environment for Fault Sensitivity Analysis”, *IEEE Transactions on Computers*, Volume 41 (12), pp.1515-1526, Décembre 1992.

[Chou 2001] A. Chou, J. Yang, B. Chelf, S. Hallem et D. Engler, “An Empirical Study of Operating Systems Errors”, dans *Proc. 18th ACM Symp. on Operating Systems Principles (SOSP-2001),* (Banff, AL, Canada), pp.73-88, ACM Press, 2001.

[Crouzet 2004] Y Crouzet, A. Kalakech, K Kanoun et A Arlat, “ Etalonnage de la sûreté de fonctionnement de systèmes d'exploitation”, dans *Congrès de Maîtrise des Risques et Sûreté de Fonctionnement,* (Bourges, France), pp.98-105, 12-14 Octobre 2004.

[Durães 2002] J. Durães et H. Madeira, “Characterization of Operating Systems Behavior in the Presence of Faulty Drivers through Software Fault Emulation”, dans *2002 Pacific Rim Int. Sym. on Dependable Computing,* (Tsukuba City, Ibaraki, Japan), pp.201-209, Décembre 2002.

[Durães 2004] J. Durães, M. Vieira et H. Madeira, “ Dependability Benchmarking of Web-Servers”, dans *23rd International Conference on Computer Safety, Reliability and Security (SAFECOMP 2004),* (Postdam, Germany), Septembre 2004.

[Fabre 1999] J.-C. Fabre, F. Salles, M. Rodríguez moreno et J. Arlat, “Assessment of COTS Microkernels by Fault Injection”, dans *Proc. 7th IFIP Working Conf. on Dependable Computing for Critical Applications,* (C. B. Weinstock and J. Rushby, Eds.), (CA, USA), pp.25-44, Janvier 1999.

[Forrester 2000] J.E. Forrester et B.P. Miller, “An Empirical Study of the Robustness of Windows NT Applications Using Random Testing”, dans *Proc. 4th USENIX Windows System Symposium,* (Seattle, WA, USA), Août 2000.

[Fuchs 1996] E. Fuchs, “An Evaluation of the error Detection Mechanisms in MARS Using Software Implemented Fault Injection”, dans *European Dependable Computing Conf. (EDCC),* (Toarmina, Italy), pp.73-90, 1996.

[Gray 1990] J. Gray, “A Census of Tandem System Availability Between 1985 and 1990”, *IEEE Transactions on Reliability*, Volume R-39 (4), pp.409-418, 1990.

[Gray 1993] J. Gray (Ed.), *"The Benchmark Handbook for Database and Transaction Processing Systems",* 592p., Morgan Kaufmann Publishers, San Francisco, CA, USA, 1993.

[Gunneflo 1989] U. Gunneflo, J. Karlsson et J. Torin, “Evaluation of Error Detection Schemes Using Fault Injection by Heavy-ion Radiation”, dans *19th Symposium on Fault-Tolerant Computing (FTCS-16),* (Chicago, USA), pp.138-143, Juin 1989.

[Hildebrand 1992] D. Hildebrand, “An Architectural Overview of QNX”, dans *Proc. of the USENIX Workshop on Micro-kernels and Other Kernel Architectures,* (Seattle, WA, USA), pp.113-126, Avril 1992.

[Hunt 1999] G. Hunt et D. Brubaher, “Detours: Binary Interception of Win32 Functions”, dans *3rd USENIX Windows NT Symposium,* (Seattle, Washington, USA), pp.135-144, 1999.

[Ibm 1990] Ibm, *AIX version 3.1 for RISC System / 6000: general concepts and procedures*, IBM, Technical Report, 1990.

[Iometer] Iometer, Intel, http://www.iometer.org.

[Jarboui 2002a] T. Jarboui, J. Arlat, Y. Crouzet et K Kanoun, “Experimental Analysis of the Errors Induced into Linux by Three Fault Injection Techniques”, dans *Proc. Int. Conf. on Dependable Systems and Networks (DSN-2002),* (Washington, DC, USA), IEEE CS Press, Juin 2002.

[Jarboui 2002b] T. Jarboui, J. Arlat, Y. Crouzet, K. Kanoun et T. Marteau, “Analysis of the Effects of Real and Injected Software Faults: Linux as a Case Study”, dans *Proc. 2002 Pacific Rim International Symposium on Dependable Computing (PRDC 2002),* (Tsukuba City, Ibaraki, Japan), pp.51-58, IEEE CS Press, Décembre 2002.

[Jarboui 2002c] T. Jarboui, A. Kalakech et O. Guitton, “Assessment of the robustness of Windows2000 via the injection of select bit-flips”, dans *4th European Dependable*

Computing Conference (EDCC-4). Fast abstracts track, (Toulouse, France), pp.9-10, Octobre 2002.

[Jarboui 2003] T Jarboui, “Impact of internal and external software faults on the Linux Kernel”, *IEICE Transactions on Information and Systems*, Volume E86-D (12), pp.2571-2578, Décembre 2003.

[Kalakech 2004a] A. Kalakech, T Jarboui, A Arlat, Y Crouzet et K Kanoun, “Benchmarking Operating Systems Dependability: Windows as a Case Study”, dans *2004 Pacific Rim International Symposium on Dependable Computing (PRDC 2004),* (Papeete, Polynesia), pp.262-271, IEEE CS Press, Mars 2004.

[Kalakech 2004b] A. Kalakech, K Kanoun, Y Crouzet et A Arlat, “Benchmarking the dependability of Windows NT4, 2000 and XP”, dans *International Conference on Dependable Systems and Networks (DSN'2004),* (Florence, Italie), pp.681-686, Juin 2004.

[Kanawati 1995] G.A. Kanawati, N.A. Kanawati et J.A. Abraham, “FERRARI: A Flexible Software-Based Fault and Error Injection System”, *IEEE Transactions on Computers*, Volume 44 (2), pp.248-260, Février 1995.

[Kanoun 2004] K. Kanoun, H. Madeira, Y. Crouzet, F. Moreira et J.C. Ruiz-Garcia, *Dbench: Dependability Benchmarks*, LAAS, Project IST 2000-25425, Mars 2004.

[Kao 1995] W.-L. Kao et R.K. Iyer, “DEFINE: A Distributed Fault Injection and Monitoring Environment”, in *Fault-Tolerant Parallel and Distributed Systems* (D. Pradhan and D. R. Avresky, Eds.), pp.252-259,IEEE CS Press, Los Alamitos, CA, USA, 1995.

[Kao 1993] W.-L. Kao, R.K. Iyer et D. Tang, “FINE: A Fault Injection and Monitoring Environment for Tracing the UNIX System Behavior under Faults”, *IEEE Transactions on Software Engineering*, Volume 19 (11), pp.1105-1118, Novembre 1993.

[Katcher 1997] J. Katcher, *Postmark: A New File System Benchmark*, Network Appliance, Technical report 3022, N°3022, 1997.

[Koopman 2002] P. Koopman, “What's Wrong With Fault Injetion As A Benchmarking Tool?” dans *Proc. International Conference in Dependable System and Networks, Workshop on Dependability Benchmarking,* (Washington, D.C., USA), pp.F- 31-36, 2002.

[Koopman 1999] P. Koopman et J. Devale, “Comparing the Robustness of POSIX Operating Systems”, dans *Proc. 29th Int. Symp. on Fault-Tolerant Computing (FTCS-29),* (Madison, WI, USA), pp.30-37, IEEE CS Press, 1999.

[Koopman 1997] P.J. Koopman, J. Sung, C. Dingman, D.P. Siewiorek et T. Marz, “Comparing Operating Systems using Robustness Benchmarks”, dans *Proc. 16th Int. Symp. on Reliable Distributed Systems (SRDS-16),* (Durham, NC, USA), pp.72-79, IEEE Computer Society Press, Octobre 1997.

[Laprie 1995] J. -C. Laprie, J. Arlat, J. P. Blanquart, A. Costes, Y Crouzet, Y. Deswarte, J. C. Fabre, H. Guillermain, M. Kaâniche, K. Kanoun, C. Mazet, D. Powell, C. Rabéjac et P.

Thévenod, *"Guide de la sûreté de fonctionnement",* 324p., Cépaduès-Editions, Toulouse, France, 1995.

[Lee 1995] I. Lee et R. K. Iyer, "Software Dependability in the Tandem GUARDIAN System", *IEEE Transactions of Software Engineering*, Volume 21 (5), pp.455-467, 1995.

[Li 2000] M. Li et C. Smidts, "Ranking Software Engineering Measures Related to Reliability Using Expert Opinion", dans *11th Int. Symposium on Software Reliability Engineering (ISSRE 2000),* (San Jose, California), pp.246-258, 2000.

[Lightstone 2003] S. Lightstone, J. Hellerstein, W. Tetzlaff, P. Janson, E. Lassettre, C. Norton, B. Rajaraman et L. Spainhower, "Towards Benchmarking Autonomic Computing Maturity", dans *Proc. First IEEE Conference on Industrial Automatics (INDIN),* (Banff, Canada), Août 2003.

[Loepere 1991] K. Loepere, *MACH 3 kernel principles*, Open Software Foundation and Carnegie Mellon University, Technical Report, 1991.

[Lucas 1971] H. C. Lucas, "Performance Evaluation and Monitoring", *ACM Computing Surveys*, Volume 3 (3), pp.79-91, 1971.

[Madeira 2000] H. Madeira, D. Costa et M. Vieira, "On the Emulation of Software Faults by Software Fault Injection", dans *Proc. Int. Conference on Dependable Systems and Networks (DSN-2000),* (New York, NY, USA), pp.417-426, IEEE CS Press, Juin 2000.

[Madeira 1994] H. Madeira, M. Rela, F. Moreira et J.G. Silva, "RIFLE: A General Purpose Pin-level Fault Injector", dans *Proc. 1st European Dependable Computing Conf. (EDCC-1),* (K. Echtle, D. Hammer and D. Powell, Eds.), (Berlin, Germany), Lecture Notes in Computer Science, 852, pp.199-216, Springer-Verlag, Octobre 1994.

[Marsden 2001] E. Marsden et J.-C. Fabre, "Failure Mode Analysis of CORBA Service Implementations", dans *Proc. IFIP/ACM International Conference on Distributed Systems Platforms (Middleware'2001),* (Heidelberg, Germany), Novembre 2001.

[Mauro 2004] J. Mauro, J. Zhu et I. Pramanick, "The System Recovery Benchmark", dans *Proc. of Pacific Rim Dependable Computing (PRDC 2004),* (Papeete, Polynesia), IEEE CS Press, 2004.

[Mcgrath 2004] *Source Forge Strace Project*, http://sourceforge.net/projects/strace/, 2004.

[Mcvoy 1996] L. W. Mcvoy et C. Staelin, "lmbench: Portable Tools for Performance Analysis", dans *USENIX Annual Technical Conference,* (San Diego, CA, USA), pp.279-295, 1996.

[Meyer 1993] J.F Meyer et W.H Sanders, "Specification and Construction of Performability Models", dans *Int. Workshop on Performability Modeling of Computer and Communication Systems,* (Mont Saint Michel, France), pp.1-32, 1993.

[Microsoft 2000] Microsoft, *Windows 2000 Performance Tuning,* *http://www.microsoft.com/windows2000*, White Paper, Mars 2000.

[Microsystems 1991] Sun Microsystems, *Solaris SunOS 5.0 multithreaded architecture*, sunsoft, http://www.sun.com/solaris, White Paper, 1991.

[Miller 1995] B.P. Miller, D. Koski, C. Pheow Lee, V. Maganty, R. Murthy, A. Natarajan et J. Steidl, *Fuzz Revisited: A Re-examination of the Reliability of UNIX Utilities and Services*, University of Wisconsin, USA, Research Report, N°CS-TR-95-1268, Avril 1995.

[Miller 1990] Barton P. Miller, L. Fredriksen et B. So, "An Emprical Study of Reliability of Unix Utilities", *Communications of the Association for Computing Machinery*, Volume 33 (12), pp.32-44, Décembre 1990.

[Moreira 2004] F. Moreira, D. Costa et M. Rodriguez, "Dependability Benchmarking of Real-Time Kernels for Onboard Space Systems", dans *15th Int. Symposium on Software Reliability Engineering (ISSRE 2004),* (Saint-Malo, France), Novembre 2004.

[Mukherjee 1997] A. Mukherjee et D.P. Siewiorek, "Measuring Software Dependability by Robustness Benchmarking", *IEEE Transactions of Software Engineering*, 23 23 (6), pp.366-376, 1997.

[Mutz 2003] D. Mutz, G. Vigna et R. Kemmerer, "An Experience Developing an IDS Simulator for the Black-Box Testing of Network Intrusion Detection Systems", dans *ACSAC,* 2003.

[Nessus 2004] Nessus, http://www.nessus.org, 2004.

[Nutt 2000] G. Nutt, *"Operating Systems: A modern Perspective",* 2000.

[Ousterhout 1990] J. Ousterhout, "Why aren't Operating Systems Getting Faster as Fast as Hardware", dans *Proc. of the 1990 Summer USENIX Technical Conference,* (Anaheim, CA, USA), pp.247-256, Juin 1990.

[Pan 2001] J. Pan, P. J. Koopman, D. P. Siewiorek, Y. Huang, R. Gruber et M. L. Jiang, "Robustness Testing and Hardening of CORBA ORB Implementations", dans *Proc. 2001 Int. Conference on Dependable Systems and Networks (DSN-2001),* (Göteborg, Sweden), pp.141-150, IEEE Computer Society Press, Juillet 2001.

[Rodríguez 2002a] M. Rodríguez, A. Albinet et J. Arlat, "MAFALDA-RT: A Tool for Dependability Assessment of Real Time Systems", dans *Proc. Int. Conf. on Dependable Systems and Networks (DSN-2002),* (Washington, DC, USA), pp.267-272, IEEE CS Press, Juin 2002.

[Rodríguez 2002b] M. Rodríguez, J.-C. Fabre et J. Arlat, "Assessment of Real-Time Systems by Fault-Injection", dans *Proc. European Safety and Reliability Conference (ESREL-2002),* (Lyon, France), pp.101-108, Mars 2002.

[Rodríguez 1999] M. Rodríguez, F. Salles, J.-C. Fabre et J. Arlat, "MAFALDA: Microkernel Assessment by Fault Injection and Design Aid", dans *Proc. 3rd European Dependable Computing Conf. (EDCC-3),* (E. M. J. Hlavicka, A. Pataricza, Ed.), (Prague, Czech Republic), LNCS, 1667, pp.143-160, Springer, Septembre 1999.

[Ruiz-Garcia 2004] J.C. Ruiz-Garcia, P. Yuste, P. Gil et L. Lemus, “ On Benchmarking the Dependability of Automotive Engine Control Applications”, dans *International Conference on Dependable Systems and Networks (DSN 2004),* (Florence, Italie), pp.857-866, 2004.

[Salles 1999] F. Salles, M. Rodríguez, J.-C. Fabre et J. Arlat, “Metakernels and Fault Containment Wrappers”, dans *Proc. 29th IEEE Int. Symposium on Fault-Tolerant Computing (FTCS-29),* (Madison, WI, USA), pp.22-29, IEEE Computer Society Press, Juin 1999.

[Shelton 2000] C. Shelton, P. Koopman et K. De Vale, “Robustness Testing of the Microsoft Win32 API”, dans *Proc. Int. Conference on Dependable Systems and Networks (DSN’2000),* (New York, NY, USA), pp.261-270, IEEE Computer Society Press, Juin 2000.

[Siewiorek 1993] D.P. Siewiorek, J.J. Hudak, B.-H. Suh et Z. Segall, “Development of a Benchmark to Measure System Robustness”, dans *Proc. 23rd Int. Symp. on Fault-Tolerant Computing (FTCS-23),* (Toulouse, France), pp.88-97, IEEE CS Press, 1993.

[Solomon 2000] D. A. Solomon et M. E. Russinovich, *"Inside Microsoft Windows 2000, Third Edition",* 2000.

[Spec] Spec, Standard Performance Evaluation Corporation, http://www.spec.org.

[Specweb99] Specweb99, *“SPECweb99 Release 1.02 (Design Document)”,* 2000 (Juillet), Standard Performance Evaluation Corporation, http://www.spec.org/web99/.

[Sullivan 1991] M. Sullivan et R. Chillarege, “Software Defects and their Impact on System Availability - A Study of Field Failures in Operating Systems”, dans *21st IEEE Int. Symp. on Fault-Tolerant Computing (FTCS-21),* (Montréal, Canada), pp.2-9, 1991.

[Sysmark] Sysmark, Business Applications Performance Corporation, http://www.bapco.com.

[Torvalds 1994] L. Torvalds, “Linux Kernel Implementation”, dans *AUUG94,* (Melbourne, Australia), pp.9-14, Septembre 1994.

[Tpc] Tpc, Transaction Processing Performance Council, http://www.tpc.org.

[Tsai 1995] T. Tsai et R. K. Iyer, “Measuring Fault tolerance with the FTAPE Fault Injection Tool”, dans *Proc Eighth International Conference on Modeling Techniques and Tools for Computer Performance Evaluation,* (Heidelberg, Germany), pp.26-40, 20-22 Septembre 1995.

[Tsai 1996] T. Tsai, R. K. Iyer et D. Jewitt, “An Approach Towards Benchmarking of Fault-Tolerant Commercial Systems”, dans *Proc. 26th Int. Symp. on Fault-Tolerant Computing (FTCS-26),* (Sendai, Japan), pp.314-323, IEEE Computer Society Press, Juin 1996.

[Uqtr 2004] Uqtr, *“Benchmarking, www.uqtr.ca/balise/benchmarking/”*, 2004.

[Vieira 2002] M. Vieira et H. Madeira, “Recovery and Performance Balance of a COTS DBMS in the Presence of Operator Faults”, dans *Int. Performance and Dependability Symposium IPDS2002,* (Bethesda, Maryland, USA), Juin 2002.

[Vieira 2003a] M. Vieira et H. Madeira, “Benchmarking the Dependability of Different OLTP Systems”, dans *Int. Conf. on Dependable Systems and Networks,* (San Francisco, CA, USA), pp.305-310, 2003.

[Vieira 2003b] M. Vieira et H. Madeira, “A Dependability Benchmark for OLTP Application Environments”, dans *29th International Conference on Very Large Data Bases (VLDB 2003),* (Berlin, Germany), pp.742-753, Septembre 2003.

[Webbench] Webbench, VeriTest, http://www.veritest.com/benchmarks/webbench.

[Zhu 2002] J. Zhu, J. Mauro et I. Pramanick, *R-Cubed (R3): Rate, Robustness, and Recovery - An Availability Benchmark Framework*, Sun Microsystems Laboratories, N°TR-2002-109, 2002.

[Zhu 2003] J. Zhu, J. Mauro et I. Pramanick, “Robustness Benchmarking for Hardware Maintenance Events”, dans *Proc. of Int. Conf. on Dependable Systems and Networks (DSN 2003),* (Sans Fransisco, CA, USA), pp.115-122, IEEE CS Press, 2003.

Table des matières

Sommaire 1

Introduction générale 5

Chapitre 1 Contexte des travaux 9

1.1 Introduction 9

1.2 Notions de sûreté de fonctionnement informatique 10

1.2.1 Concepts de sûreté de fonctionnement informatique *10*

1.2.2 La prévision de fautes *11*

1.2.3 L'injection de fautes *13*

1.2.3.1 Les attributs de l'injection de fautes 14

1.2.3.2 Injection de fautes par logiciel 16

1.3 Étalons de performance 17

1.4 Les systèmes d'exploitation 19

1.4.1 Fonctions des systèmes d'exploitation *19*

1.4.2 Classification des systèmes d'exploitation *21*

1.4.3 Exemples de systèmes d'exploitation *22*

1.5 Caractérisation des systèmes d'exploitation 24

1.5.1 Etalons de performance des systèmes d'exploitation *24*

1.5.2 Caractérisation de la sûreté de fonctionnement des systèmes d'exploitation par injection de fautes *26*

1.5.2.1 MAFALDA 26

1.5.2.2 BALLISTA 28

1.5.2.3 DBench 29

1.5.2.4 FINE 31

1.6 Conclusion........32

Chapitre 2 Étalonnage de la sûreté de fonctionnement........33

2.1 Introduction........33

2.2 Etalons de sûreté de fonctionnement existants........33

2.3 Cadre conceptuel pour l'étalonnage de la sûreté de fonctionnement........37

2.3.1 Dimensions de catégorisation........38

2.3.2 Dimension de mesures........39

2.3.3 Dimensions d'expérimentation........41

2.4 Propriétés d'un étalon de sûreté de fonctionnement........45

2.4.1 La répétitivité........45

2.4.2 La reproductibilité........45

2.4.3 La représentativité........46

2.4.4 La portabilité........48

2.4.5 La non-intrusivité........48

2.4.6 L'interférence........49

2.4.7 La mise à l'échelle........49

2.4.8 L'automatisation........49

2.4.9 Le coût de l'étalonnage........49

2.5 Conclusion........50

Chapitre 3 Spécification d'étalons de systèmes d'exploitation........51

3.1 Introduction........51

3.2 Spécification des étalons........52

3.2.1 Système cible et contexte d'étalonnage........52

3.2.2 Mesures fournies par l'étalon........53

3.2.2.1 Mesures de base........54

3.2.2.2 Mesures complémentaires........56

3.2.2.3 Récapitulatif........57

3.2.3 L'expérimentation........59

3.2.3.1 Techniques de corruption des paramètres des appels système........59

3.2.3.2 Profils d'exécution 60

3.2.4 Environnement d'étalonnage et conduite d'expériences....62

3.3 Validation des propriétés par construction....66

3.3.1 La reproductibilité 66

3.3.2 La représentativité 66

3.3.3 La portabilité....67

3.3.4 La non-intrusivité....68

3.3.5 La mise à l'échelle 68

3.4 Conclusion....68

Chapitre 4 Étalonnage de la sûreté de fonctionnement des systèmes Windows......71

4.1 Introduction 71

4.2 Mise en œuvre 72

4.3 Mesures de l'étalon 73

4.3.1 Robustesse de l'OS....73

4.3.2 Temps de réaction de l'OS....74

4.3.3 Temps de redémarrage du système....75

4.3.4 Récapitulatif....75

4.4 Mesures complémentaires et affinement....75

4.4.1 Robustesse de l'OS....75

4.4.2 Temps de réaction de l'OS....77

4.4.3 Temps de redémarrage 82

4.4.4 Temps d'exécution de l'activité....83

4.4.5 Récapitulatif....84

4.5 Validation des propriétés par expérimentation 84

4.5.1 La reproductibilité 84

4.5.2 La répétitivité 85

4.5.3 La représentativité 85

4.5.3.1 Impact de la technique de corruption des paramètres....86

4.5.3.2 Impact des appels système considérés 87

4.5.3.3 Comparaison entre substitution sélective et substitution systématique 88

4.5.3.4 Récapitulatif 89

4.5.4 La portabilité..90

4.5.5 L'interférence..90

4.5.6 L'automatisation..90

4.5.7 Le coût de l'étalonnage 91

4.6 Conclusion...92

Chapitre 5 Étalonnage de Windows et de Linux 93

5.1 Introduction 93

5.2 Mise en œuvre..93

5.3 Résultats de DBench-OS-Postmark..96

5.3.1 Mesures de l'étalon...96

5.3.2 Raffinement des mesures de l'étalon...100

5.4 Résultats de DBench-OS-JVM..104

5.4.1 Mesures de l'étalon...104

5.4.2 Raffinement des mesures...107

5.5 Comparaison...110

5.6 Conlcusion..112

Conclusion générale 115

Annexe 1 Exemple d'implémentation des mécanismes de substitution de paramètres et d'observation 119

Annexe 2 Raffinement des mesures de robustesse par rapport à l'état final de *Postmark*..121

Annexe 3 Validation des résultats de DBench-OS-Postmark et DBench-OS-JVM par rapport à l'ensemble de fautes...123

Annexe 4 Temps de redémarrage de Windows et de Linux avec DBench-OS-Postmark...127

Références bibliographiques...129

Table des matières 137

« Étalonnage de la sûreté de fonctionnement des systèmes d’exploitation – Spécifications et mise en œuvre »

RESUME

Les développeurs des systèmes informatiques, y compris critiques, font souvent appel à des systèmes d’exploitation sur étagère. Cependant, un mauvais fonctionnement d’un système d’exploitation peut avoir un fort impact sur la sûreté de fonctionnement du système global, d’où la nécessité de trouver des moyens efficaces pour caractériser sa sûreté de fonctionnement.

Dans cette thèse, nous étudions l’étalonnage de la sûreté de fonctionnement des systèmes d’exploitation par rapport aux comportements défectueux de l’application. Nous spécifions les propriétés qu’un étalon de sûreté de fonctionnement doit satisfaire. Après, nous spécifions les mesures et la mise en oeuvre des trois étalons destinés à comparer la sûreté de fonctionnement de différents systèmes d’exploitation.

Ensuite, nous développons les prototypes des trois étalons. Ces prototypes servent à comparer les différents systèmes d’exploitation des familles Windows et Linux, et pour montrer la satisfaction des propriétés identifiées.

MOTS-CLÉS : Systèmes d’exploitation, Etalonnage de sûreté de fonctionnement, Injection de fautes, Analyse de performance, Robustesse.

" Benchmarking Operating Systems Dependability – Specifications and Implementation "

ABSTRACT

System developers are increasingly resorting to off-the-shelf operating systems, even in critical application domains. Any malfunction of the operating system may have a strong impact on the dependability of the global system. Therefore, it is important to make available information about the operating systems dependability.

In our work, we aim to specify dependability benchmarks to characterize the operating systems with respect to the faulty behavior of the application. We specify three benchmarks intended for comparing the dependability of operating systems belonging to different families. We specify the set of measures and the procedures to be followed after defining the set of properties that a dependability benchmark should satisfy.

After, we present implemented prototypes of these benchmarks. They are used to compare the dependability of operating systems belonging to Windows and Linux, and to show that our benchmarks satisfy the identified properties.

KEY-WORDS : Operating Systems, Dependability Benchmarking, Fault Injection, Performance analysis, Robustness.