Institut **S**upérieur
d'**I**nformatique de
Modélisation et de
leurs **A**pplications

Complexe des Cézeaux
BP 125
63173 AUBIERE Cedex

European **O**rganization
for
Nuclear **R**esearch

CH 1211
GENEVE 23
Switzerland

Rapport de stage de 2ème année
Option Architectures Matérielles et Conception de Circuits

Amélioration de la gestion des périphériques d'un système embarqué

Présenté par Jean-Christophe Garnier

Tuteur entreprise : Niko Neufeld
Tuteur ISIMA : Emmanuel Mesnard

Mardi 11 Septembre 2007

Remerciements

Je tiens à remercier mon tuteur de stage Niko Neufeld, qui a su me faire confiance et qui a eu la patience de me fournir des explications à propos des technologies impliquées dans mon projet, ou encore sur les technologies de l'information en général.

Je remercie également Emmanuel Mesnard pour son implication dans le rôle de tuteur de stage.

Je remercie tout particulièrement Sai Suman Cherukuwada pour toutes les connaissances qu'il a partagé avec moi, à propos de la programmation et de l'utilisation des systèmes UNIX, des technologies réseaux ou de stockage, et pour tous les conseils qu'il a su m'apporter.

Enfin je remercie toute l'équipe LHCb Online pour m'avoir accueilli et pour avoir porté un intérêt à mon travail, et d'une manière générale toutes les personnes avec qui mes rapports furent enrichissants.

Résumé

L'expérience LHCb, menée dans le cadre du CERN, recueille un nombre extraordinaire de données. Le système d'acquisition de ces données est donc démesuré, entièrement dédié à cette tâche. Des centaines de **cartes électroniques TELL1** assurent la sélection des données pertinentes et leur transfert sur le réseau avant qu'elles soient finalement enregistrées.

Un **système embarqué**, la **Creditcard PC** permet aux physiciens de contrôler et de modifier le comportement de la carte TELL1. Ce système est équipé d'une **distribution Linux** propre au CERN, et de **bibliothèques** spécifiques pour gérer la communication avec la carte TELL1. De nombreux **utilisateurs** peuvent travailler sur la même carte TELL1 à un moment donné, il faut donc qu'une **synchronisation** soit assurée au niveau de l'accès aux divers périphériques par les **processus**.

Après avoir réalisé des tests sur la paire de cartes qui m'était dédiée, j'ai déployé la solution sur un ensemble de cartes plus important, avant de finalement appliquer la solution à l'ensemble des cartes TELL1 de l'expérience LHCb et à d'autres systèmes utilisant la Creditcard PC.

Mots clés : cartes électroniques TELL1, Credit Card PC, distribution Linux, système embarqué, bibliothèques, utilisateurs, synchronisation, processus.

Abstract

At CERN, experiments must gather huge amounts of data. The Data Acquisition System of the LHCb is therefore very large, and dedicated to this task. Hundreds of **TELL1 boards** provide the selection of relevant data and their transfer over a local area network in order to be saved.

An **embedded system**, the **Creditcard PC**, allows physicists to monitor and program the TELL1 boards. This system runs a CERN release of the **Linux operating system**, and provides several **libraries** to handle the communication with the TELL1 board. Several **users** can work on the same TELL1 board simultaneously. This calls for **process synchronization** to control the access to the devices.

Tests were made progressively, from some dedicated boards to a larger assembly, and then the solution has been distributed to the all TELL1 boards and to other boards using the Creditcard PC.

Key words: TELL1 board, embedded system, Creditcard PC, Linux operating system, libraries, users, process synchronization.

Glossaire

Accélérateur : Un accélérateur consiste en un tube sous vide dans lequel des particules sont accélérées par des champs électriques et dont la trajectoire est guidée par des champs magnétiques.

Antimatière : La matière est constituée de particules, l'antimatière d'antiparticules. Une antiparticule possède une charge électrique opposée à celle d'une particule. L'antimatière est maintenant rare dans l'univers alors que sa quantité était égale à la quantité de matière lors du Big Bang. La rencontre de la matière avec l'antimatière est sensée les annihiler toutes deux.

API (Application Programming Interface) : Une interface de code source fournie par une bibliothèque ou un système informatique pour répondre aux requêtes d'un programme.

BIOS (Basic Input Output System) : Une mémoire morte (ROM) qui permet d'effectuer les opérations de bases sur un ordinateur (écrire à l'écran, lire un secteur du disque, écrire en mémoire, …).

Broadcast : Dans les technologies réseaux, ce terme signifie le fait d'émettre depuis une source unique vers plusieurs destinataires.

Bus : Ensemble de liaisons physiques qui peuvent être exploitées en commun par plusieurs éléments pour communiquer.

CCPC (Creditcard PC) : Système embarqué de contrôle des cartes TELL1.

DHCP (Dynamic Host Configuration Protocol) : Protocole réseau qui assure la configuration automatique des machines connectées.

EEPROM (Electrically-Erasable Programmable Read-Only Memory) : mémoire morte reprogrammable électriquement. Une mémoire morte contient des données qui ne sont pas effacées lorsqu'elle n'est plus alimentée en électricité.

Etherboot : Secteur d'amorçage ROM qui permet de démarrer à partir d'un réseau en s'appuyant sur divers protocoles.

FPGA (Field-Programmable Gate Array) : Circuit intégré reprogrammable, capable de réaliser des traitements complexes.

GPIO (General Purpose Input/Output) : Périphérique qui propose d'écrire dans des registres servant à la configuration du matériel connecté, ou bien de lire l'état de ce matériel.

I^2C (Inter Integrated Circuit Bus) : Bus série synchrone. Les informations sont envoyées à la suite sur le même fil, une donnée par coup d'horloge.

IPC (Inter Process Communication) : Techniques permettant de partager des données entre plusieurs processus.

JTAG (Joint Test Action Group) : Utilisé au lieu du terme générique Boundary-Scan, c'est une norme pour faciliter et automatiser les tests sur les circuits intégrés. Il a désormais d'autres applications comme la programmation des composants logiques programmables (FPGA).

Méson : Particule non élémentaire composée d'un nombre pair de quarks et d'antiquarks.

Microcontrôleur : Circuit intégré composé d'un microprocesseur, de mémoires et de périphériques de communication.

NFS (Network File System) : Protocole de partage de systèmes de fichiers à travers un réseau. Il permet à un ordinateur de travailler sur un disque dur distant.

NPTL (Native POSIX Thread Library) : Bibliothèque permettant au noyau Linux d'exécuter des threads POSIX de manière efficace.

PCI (Peripheral Component Interface) : Bus synchrone qui supporte un multiplexage des signaux d'adressages et de données. Il est prévu pour le fonctionnement maître esclave sous le contrôle d'un arbitre.

POSIX (Portable Operating System Interface) : Standards définis par l'IEEE pour uniformiser les APIs des logiciels destinés à fonctionner sur des variantes du système d'exploitation UNIX.

Quark : Le constituant élémentaire de la matière. Il existe plusieurs quarks ayant pour propriétés une fraction de charge électrique élémentaire. Ainsi deux quarks de charges +2/3 et un quark de charge -1/3 forment un proton.

Signaux : Informe un processus à propos d'un événement de manière asynchrone.

System V : Une des principales branches de la famille des systèmes UNIX. System V a servi de base à l'élaboration de la norme POSIX.

TFTP (Trivial File Transfer Protocol) : Protocole simplifié de transfert de fichier, surtout efficace sur les réseaux locaux.

Watchdog : Compteur matériel qui réinitialise le système informatique si jamais il n'a pas été réinitialisé dans un temps imparti.

Table des Matières

Table des Figures et des Illustrations

Introduction

Le CERN (Organisation Européenne pour la Recherche Nucléaire) fournit aux scientifiques les moyens de réaliser des expériences très poussées dans le domaine de la physique des particules. Quatre expériences s'articulent autour de l'accélérateur de particules LHC et étudient ce qui se passe lors de collisions de particules. Chaque expérience repose en premier lieu sur un détecteur qui collecte une grande quantité de données qu'il faut analyser et enregistrer. Une partie de l'analyse est effectuée alors que les données transitent sur le système d'acquisition pour réduire le volume à enregistrer.

Le système d'acquisition du LHCb repose sur les cartes TELL1, configurées et contrôlées par un système embarqué, la Creditcard PC. Ce système permet aux scientifiques de programmer divers périphériques sur la carte d'acquisition. Cependant, l'interface logicielle avant les travaux effectués dans le cadre de ce projet était incomplète et ne prenait pas en compte le fait que plusieurs utilisateurs pouvaient accéder au même périphérique au même moment et ainsi créer des conflits.

Une fonctionnalité matérielle du système embarqué est un « watchdog » utile pour contrôler à distance le redémarrage de la carte en cas de problème. Cependant, son comportement n'est pas optimal dans certaines conditions d'utilisation et une meilleure gestion de cette ressource est nécessaire.

La première partie du travail consistera à mettre en place des mécanismes de synchronisation pour contrôler les accès aux périphériques de la carte TELL1. La seconde partie sera d'améliorer la gestion du watchdog pour qu'il réponde au mieux aux besoins du LHCb.

1 Présentation générale

1.1 Le CERN

Le CERN est l'Organisation Européenne pour la Recherche Nucléaire. Le plus grand centre de recherche en physique des particules du monde se situe sur la frontière franco-suisse. Il est le résultat d'un des premiers projets de coopération européenne, et compte de nos jours 20 états membres européens.

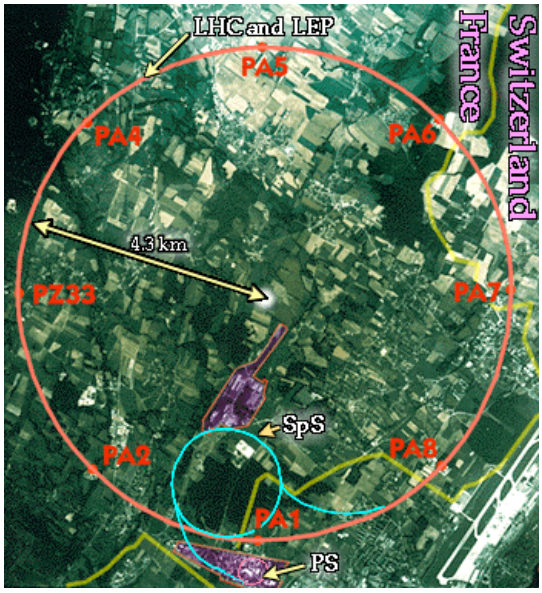

Figure 1 Vue satellite des complexes du CERN

Il permet aussi à de nombreux Etats de participer à ses expériences. On retrouve ainsi 6500 scientifiques, de 80 nationalités différentes, représentant environ 500 universités et instituts de par le monde. Ces scientifiques sont appuyés dans leurs tâches par environ 2500 personnes, physiciens, ingénieurs, techniciens, ouvriers administrateurs ou secrétaires, qui sont entre autres chargés de concevoir et de construire les appareils sophistiqués mis en œuvre dans chaque expérience : le collisionneur LHC et des détecteurs.

1.2 L'expérience LHCb

L'expérience LHCb, qui signifie Large Hadron Collider beauty experiment, est une expérience spécialisée, avec un but clairement défini. Ce but est la compréhension de l'asymétrie matière-antimatière. Une condition qui explique ce phénomène est la violation de la symétrie CP, que LHCb va observer lors de la désintégration de mésons B.

1.2.1 L'objectif

L'univers est en apparence formé uniquement de matière, de particules. Les physiciens pensent que lors du Big Bang, la matière et l'antimatière ont été créées dans des proportions égales mais ils ne savent pas expliquer ce que serait devenu l'antimatière.

La rencontre entre une particule et une antiparticule, entre de la matière et de l'antimatière, annihile les deux substances pour ne produire que de l'énergie. Comment expliquer alors la présence plus importante de matière que d'antimatière dans l'univers ?

La collision entre deux protons permet de produire de nombreuses particules. Les scientifiques du LHCb espèrent détecter des Mésons B et observer leur désintégration. Un méson est composé d'un nombre égal de quark et d'antiquark, donc de matière et d'antimatière.

1.2.2 Le détecteur

Plus on souhaite étudier l'infiniment petit, plus les détecteurs sont imposants. Contrairement aux détecteurs polyvalents que sont ATLAS et CMS, LHCb est destiné à réaliser la meilleure détection possible des mésons B. Ces mésons sont susceptibles d'être produits dans des directions voisines du faisceau. Le détecteur n'est donc pas circulaire mais orienté pour couvrir au mieux la direction dans laquelle se disperseront les particules les plus intéressantes.

Le détecteur est composé de nombreux sous-détecteurs, chacun est destiné à déterminer un type de particule en fonction de leurs propriétés physiques :

- VELO (VErtex LOcator) : situé dans l'axe du faisceau, ce détecteur permet de fournir des mesures précises de la trajectoire des particules à proximité de la zone de collision. Il est également utilisé pour reproduire la création et l'affaiblissement des mésons Beauté et Charme afin de fournir des données précises de leur durée de vie.
- RICH 1&2 : les deux « Ring Image CHerenkov counters » permettent d'identifier les particules chargées passant dans le détecteur, ainsi que leur quantité de mouvement.
- Spectromètre : dipôle magnétique situé à proximité de la zone de la collision pour en minimiser la taille.
- Tracking System : ce détecteur détermine la trajectoire des particules entre le VELO et les calorimètres, ainsi que la quantité de mouvement des particules.
- Calorimètres : en absorbant l'énergie des particules, les calorimètres permettent d'identifier les hadrons, les électrons et les photons, qui sont parfois le résultat de l'affaiblissement des mésons B. Ils fournissent aussi des informations sur leur énergie.
- Muon : ce détecteur utilise la capacité de pénétration des muons pour les détecter précisément. Les muons sont aussi un résultat de l'affaiblissement des mésons B.

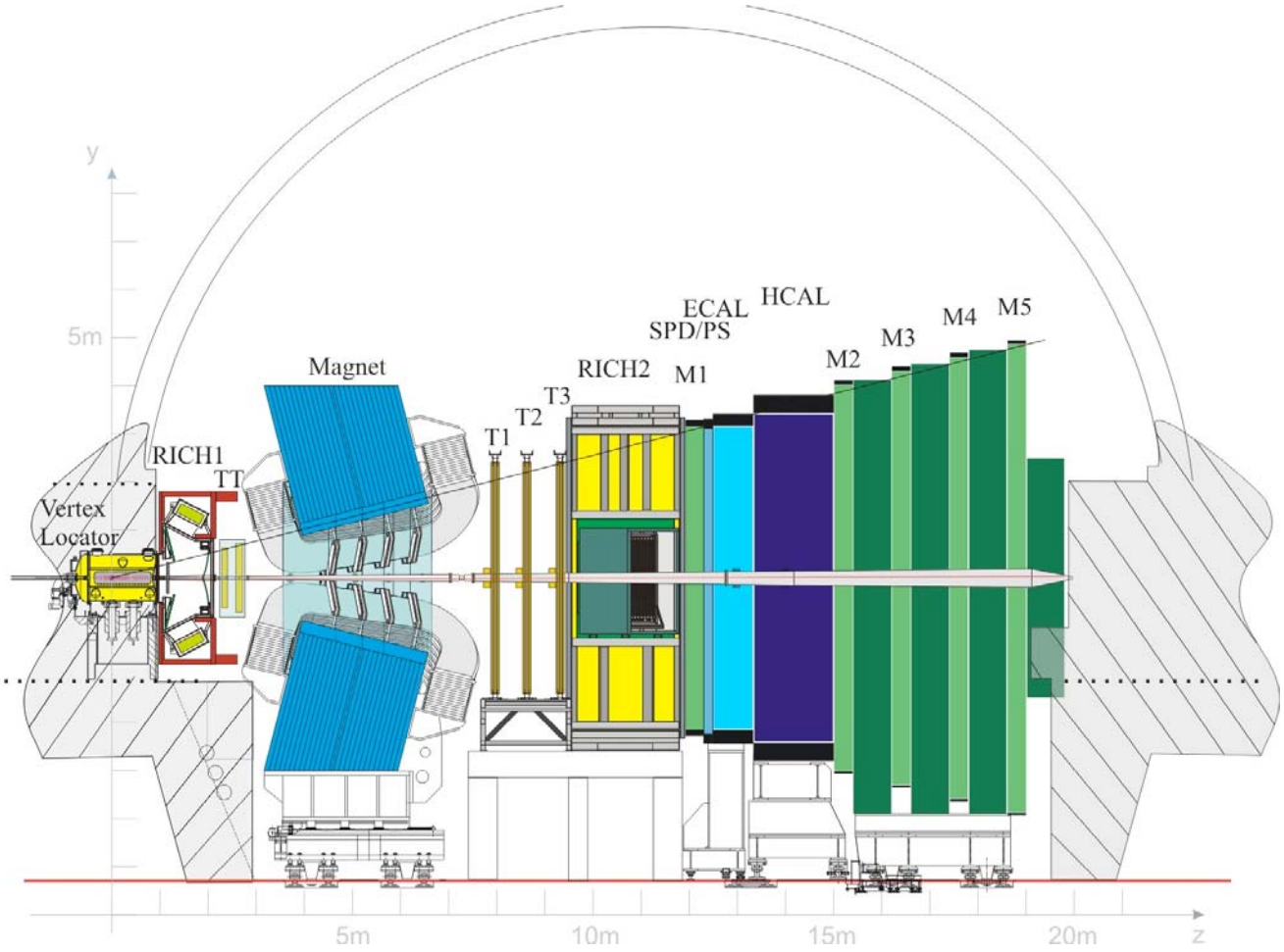

Figure 2 Schéma en coupe du détecteur de l'expérience LHCb

Le détecteur global permet donc d'identifier les particules et d'étudier leur énergie et leur quantité de mouvement.

1.3 Moyens mis en œuvre

Le système DAQ (Data AcQuisition) s'étend de l'électronique qui interface les sous-détecteurs de l'expérience LHCb jusqu'au système de stockage des données potentiellement intéressantes. Tout au long de ce système, les données sont sélectionnées par différents mécanismes pour ne conserver que ce qui est potentiellement pertinent et intéressant pour l'expérience. On différencie ainsi LHCb Online, qui

regroupe tout ce qui concerne le traitement effectué au cours de l'acquisition, de LHCb Offline, qui regroupe donc le traitement effectué sur les données stockées.

1.3.1 Le système d'acquisition de données

Le LHC produira des collisions, ou événements, à une fréquence de 40 MHz, mais peu de collisions sont susceptibles de fournir un résultat intéressant pour les physiciens du LHCb. Les collisions intéressantes ne seront que de l'ordre de 10 MHz, mais seulement 1% de ces collisions produiront des paires de quarks et d'anti-quark beauté. De plus, seulement 15% de ces collisions ne produiront des mésons B, dont la désintégration ne sera intéressante que dans un cas pour mille, la fréquence de collisions intéressantes sera au final de 15 Hz.

Comme il n'existe aucune technologie permettant de transférer et de stocker les informations relatives à toutes les collisions, qui représentent tout de même 40 To/s, le LHCb a mis au point, conjointement au système DAQ, un système de sélection des données, le Trigger System.

Ce système de sélection est aussi bien basé sur de l'électronique que sur du logiciel. Ses différents niveaux de sélection permettent de déterminer quels sont les événements à conserver.

La première sélection s'effectue au niveau du L0 Trigger. Les calorimètres et le détecteur MUON sont les plus rapides et permettent de déterminer si la collision est potentiellement intéressante. La fréquence est événements acquis devient donc de 1 MHz à la sortie du L0 Trigger.

Après L0, les données sont envoyées vers les cartes TELL1, le composant principal de ce sélecteur. Supervisée, elle effectue une sélection, elle met en mémoire tampon les données, puis elle les transfère via un réseau IP assez complexe vers les ordinateurs qui appliqueront ensuite des algorithmes de sélections pour encore réduire la fréquence d'événements intéressants et donc réduire la taille des données à mémoriser. Cette ferme de calcul correspond au HLT (High Level Trigger).

1.3.2 La carte TELL1

La carte TELL1 (Trigger ELectronics and L1 board) a été développée par le CERN. Les 300 cartes TELL1 qui composent l'électronique d'entrée du système DAQ sont connectées à des entrées différentes suivant le sous-détecteur auquel elles correspondent, mais leur sortie est standardisée, sous la forme de connexions Gigabit Ethernet.

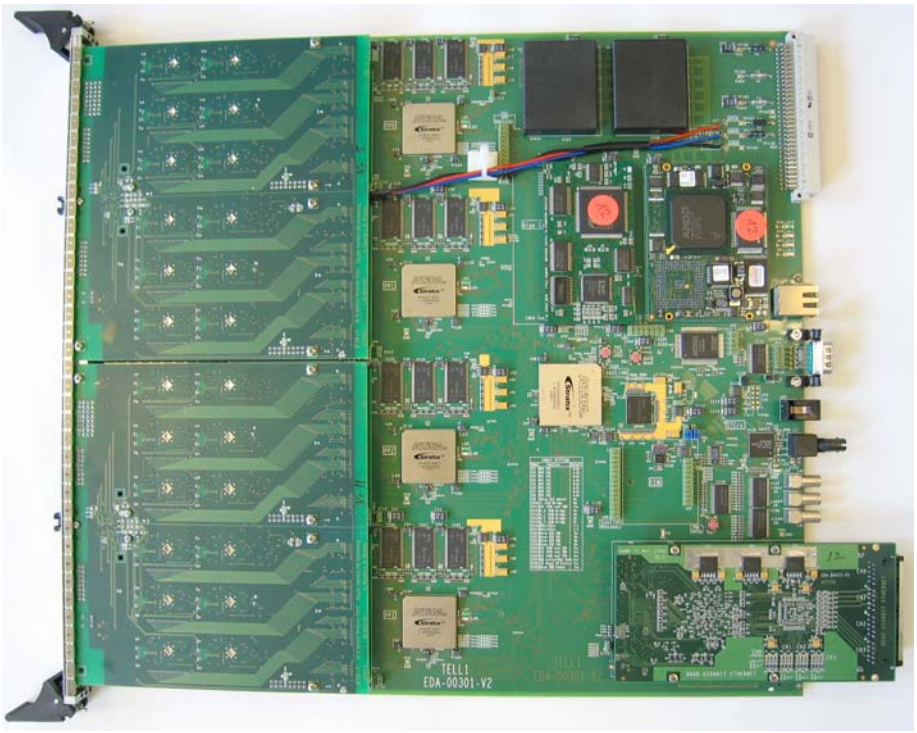

Figure 3 Photographie de la carte TELL1

La carte TELL1 possède 4 FPGA de prétraitement. Ces prétraitements dépendent du sous-détecteur auquel est connecté le FPGA. Chaque FPGA est ensuite connecté à une mémoire tampon dans laquelle il stocke les données. Après quelques traitements comme la suppression des zéros inutiles, les données sont transmises à un cinquième FPGA, le SyncLink. Ce dernier est, entre autres, chargé de les formater en un Multiple Event Packet (MPE), ce paquet contiendra les données de plusieurs événements et sera émis via une carte Gigabit Ethernet sur le réseau IP, à destination de la ferme de calcul.

D'une manière générale, il faut pouvoir configurer et contrôler tous les périphériques de la carte TELL1. Pour cela elle possède trois interfaces, qui sont trois bus JTAG, quatre bus I^2C et un bus parallèle. Ces bus ont plusieurs objectifs.

Chacun des bus JTAG a un rôle unique. Une chaîne JTAG est dédiée à la programmation de l'unique EEPROM de la carte TELL1, qui contient le programme des FPGA. Une autre permet de configurer directement les FPGA. La dernière correspond à l'usage principal du JTAG, et permet de réaliser des tests au niveau des entrées sorties des FPGA.

Les bus I^2C sont partagés entre plusieurs objectifs, notamment le déboguage des FPGA et le contrôle des cartes d'émissions et de réceptions de la TELL1.

Enfin, le bus parallèle permet d'accéder à toutes les mémoires RAM, aux mémoires tampons MEP ainsi qu'aux registres de configuration des FPGA [HAEFALI et Coll.].

La carte TELL1 est équipée d'un petit module pour contrôler et programmer tous ses périphériques, la Creditcard PC.

1.3.3 La Creditcard PC et la Glue-Card

La Creditcard PC est un ordinateur embarqué, donc de petite taille. C'est un Smart Module SM520 développé par Digital Logic. Il est équipé d'un microcontrôleur AMD Elan 520, ainsi qu'entre autres d'une mémoire RAM, d'un bus PCI et d'un contrôleur Ethernet. Ce microcontrôleur fournit notamment un watchdog, qui permet de redémarrer la carte si jamais il n'a pas été réinitialisé avant une certaine échéance.

La carte démarre par le réseau grâce à un module du BIOS. Elle fait tourner une distribution Linux Red Hat 4 avec un noyau 2.6.9 spécialement modifié pour la carte. Ce noyau possède des modules propres et a été alléger au maximum pour permettre à la carte de démarrer le plus rapidement possible.

Le système de fichier provient d'un serveur NFS (Network File System), ainsi tout est centralisé, ce qui permet de facilement gérer les nombreuses Creditcard PC.

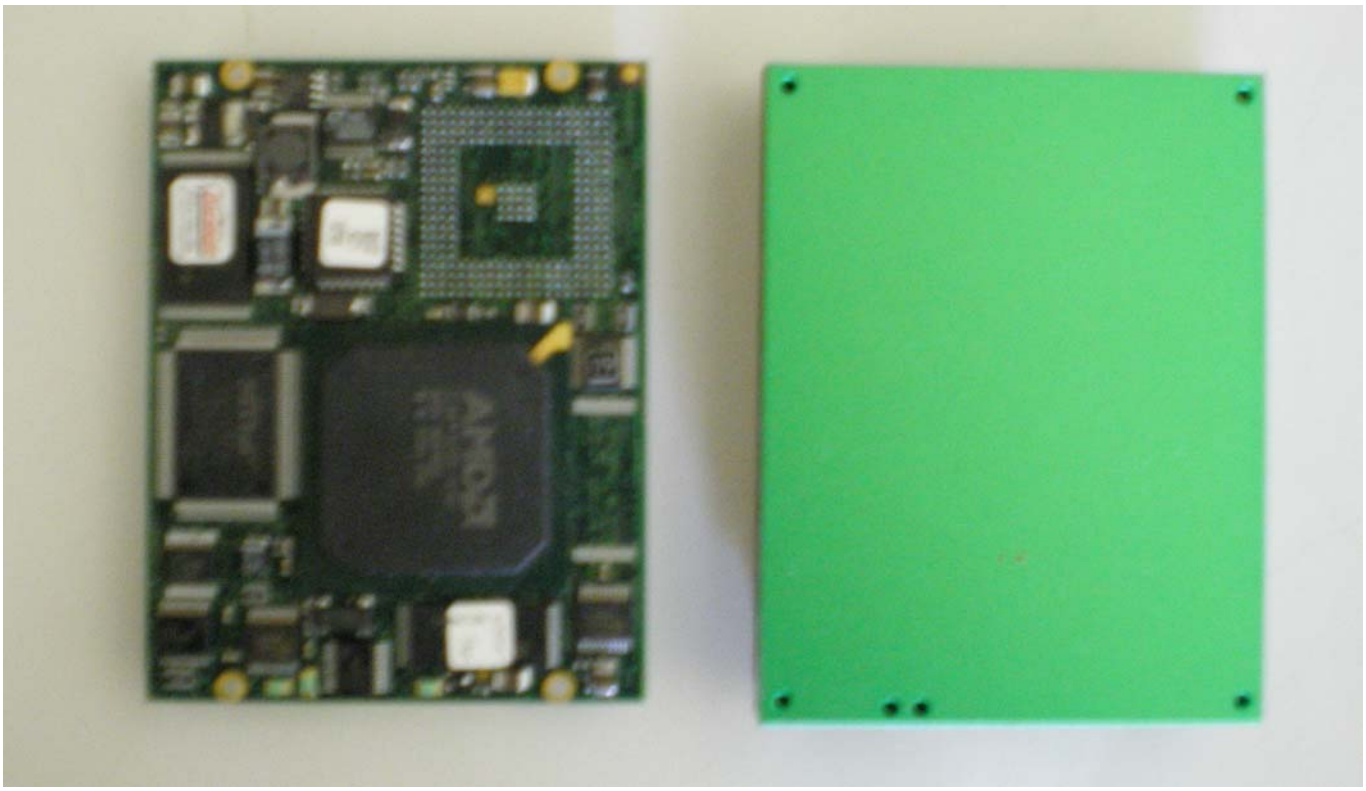

Figure 4 Photographie de la Creditcard PC

Cette carte ne possède qu'une interface PCI pour que ses composants puissent communiquer avec l'extérieur. La communication avec les différents bus de la carte TELL1 se fait par l'intermédiaire d'une autre carte développée par le CERN, la Glue-Card.

Connectée au bus PCI, elle l'interface en un bus parallèle grâce au composant PLX 9030, qui est lui-même en partie interfacé en bus JTAG et I^2C par un FPGA. Le composant PLX fournit aussi des lignes GPIO permettant de configurer et contrôler divers périphériques. Ainsi cette carte assure la complète communication entre la CCPC et la carte TELL1.

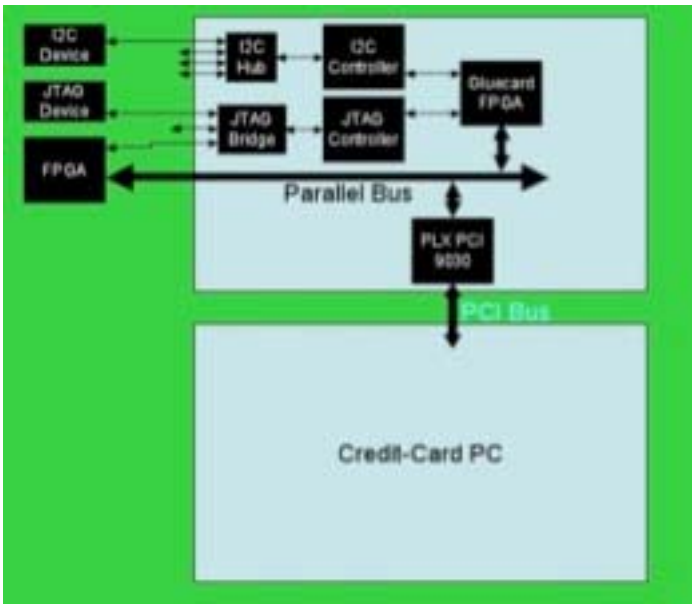

Figure 5 Schéma de la connexion CCPC – Glue-Card

C'est dans ce contexte d'accès aux ressources de la TELL1 depuis la CCPC à travers la Glue-Card que va se dérouler mon projet principal. Le second projet que je vous présente se déroule plus dans le cadre des interactions entre la CCPC et le système d'exploitation Linux.

2 La synchronisation des processus

2.1 Le logiciel existant

Au LHCb, des physiciens travaillent sur les cartes TELL1 pour les programmer et les contrôler. Pour cela ils se connectent sur la Creditcard PC et utilisent des outils fournis pour l'utilisation de la Glue-Card, ou bien développent leurs propres applications. Ils n'ont pas à se soucier de la communication entre la CCPC et la Glue-Card. Celle-ci est assurée par des bibliothèques qui fournissent l'API nécessaire pour communiquer avec les modules du noyau qui gèrent la Glue-Card [NEUFELD 2004].

Il y a actuellement deux modules, le premier gère l'interface entre le bus PCI et le composant PLX 9030, le second interface les bus de la Glue-Card. Ces deux modules sont donc complémentaires.

Techniquement, ces seuls modules suffisent à assurer la communication, à bas niveau, entre la CCPC et la Glue-Card. Mais il est possible que la configuration matérielle soit modifiée, ce qui impliquerait une modification des modules. Une API est donc assurée par des bibliothèques partagées. Elle assure que les modifications apportées aux modules ou au matériel n'entraîneront pas des modifications du comportement des applications développées par les utilisateurs. L'API permet aussi de simplifier l'accès à la Glue-Card car ses fonctions regroupent toutes les opérations nécessaires aux modules pour accomplir les objectifs des utilisateurs.

On trouve ainsi plusieurs bibliothèques, chacune dédiée à la gestion d'un périphérique. Elles s'appliquent par couches. La bibliothèque de base gère le bus parallèle, c'est la *lblib* (Local Bus Library). Elle communique directement avec le pilote de périphérique qui gère le composant PLX et permet de programmer ses registres.

Directement liée à la *lblib*, la *gluelib* (Glue-Card Library) fournit de quoi initialiser la Glue-Card et lire et écrire sur les lignes GPIO. L'initialisation passe par la fermeture, ou connexion, d'un *bus switch*.

Le *bus switch* est un registre représentant l'état de connectivité entre la CCPC et la Glue-Card. Cette bibliothèque doit aussi charger la bibliothèque de la « Carrier Board ».

Cette dernière est dédiée à la carte qui utilisera la CCPC. On trouve ainsi une bibliothèque *cb_tell1* pour configurer la Glue-Card en fonction des besoins de la carte TELL1, ainsi que d'autres bibliothèques liées à d'autres cartes d'acquisition basées sur le même couple CCPC – Glue-Card.

On trouve ensuite deux bibliothèques pour interfacer les bus I^2C et JTAG à partir des trois bibliothèques citées précédemment, *i2clib* et *JTAGlib*. Si la bibliothèque *i2clib* ne propose principalement que des fonctions de base pour lire et écrire, la bibliothèque *JTAGlib* est plus complexe du fait des rôles du JTAG dans la carte TELL1. Cette bibliothèque fournit les moyens de programmer les FPGA selon les moyens souhaités en fonction du type du code du FPGA et de la méthode employée pour l'écrire.

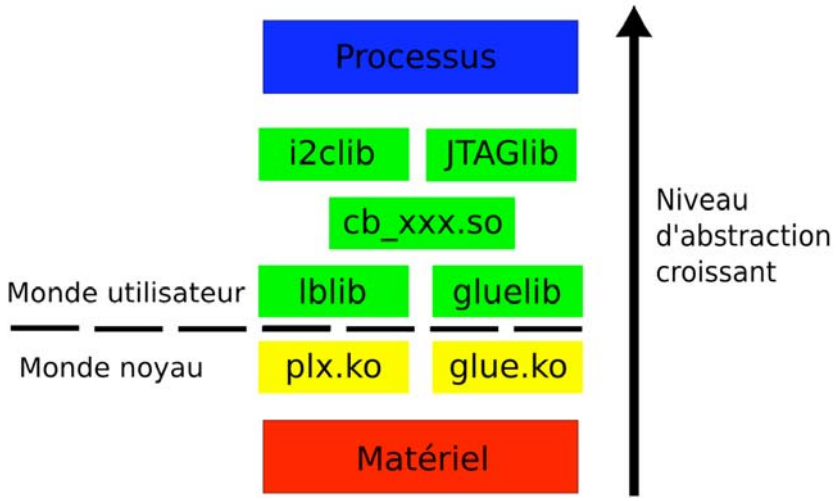

Figure 6 Architecture logicielle de la CCPC

Le système Linux de la Creditcard PC fournit donc une couche logicielle de haut niveau pour permettre aux utilisateurs de programmer simplement les accès au matériel. Chaque couche repose sur une couche inférieure qui se rapproche du matériel et devient plus complexe à utiliser.

Des outils sont livrés avec toutes ces bibliothèques pour fournir aux utilisateurs des accès aux fonctionnalités de base de la CCPC, par l'intermédiaire de lignes de commandes. Alors les utilisateurs ne

sont pas obligés de maîtriser le langage C, mais peuvent développer des scripts pour réaliser leurs applications.

La Creditcard PC fait aussi tourner un serveur, le *ccserv*. Ce serveur permet aux utilisateurs de communiquer avec la CCPC via un client distant, leur proposant ainsi une interface graphique. Le serveur leur permet d'exécuter les mêmes actions que par les lignes de commandes.

Pour résumer, il y a plusieurs moyens de travailler sur la CCPC, on peut se connecter via l'application client serveur, on peut utiliser les outils dans des scripts ou directement en ligne de commande, enfin on peut programmer soit même à partir des bibliothèques.

On remarque aussi que le logiciel s'articule de la même manière que le matériel, I^2C et JTAG sont des couches logicielles et matérielles basées sur l'utilisation du bus parallèle.

2.2 L'objectif et ses contraintes

Linux est un système d'exploitation multitâche et multi-utilisateur. Ceci implique que plusieurs utilisateurs peuvent exécuter plusieurs processus au même moment sur une même CCPC.

Dans l'état des bibliothèques à mon arrivée au CERN, rien ne garantissait qu'un processus possède l'exclusivité sur les ressources de la Glue-Card. Les bibliothèques implémentaient l'accès à la Glue-Card, mais aucun contrôle n'était effectué quant à savoir si une ressource était libre d'accès ou non. Ainsi deux processus peuvent programmer en parallèle le même FPGA, un processus peut déconnecter la Glue-Card de la CCPC alors qu'un autre processus lis des informations sur les bus I^2C, …

Ces processus peuvent être d'utilisateurs distincts. Ils peuvent être tout simplement les outils exécutés directement, un script qui créé des processus fils pour chaque appel d'outil, un programme développé par un utilisateur à partir des bibliothèques, une routine du serveur, …

Les processus utilisant les bibliothèques doivent donc être synchronisés. S'il est possible de modifier directement les outils sans pour autant troubler les utilisateurs, il est beaucoup plus difficile de modifier leurs applications. Les scientifiques qui développent sur les cartes TELL1 sont nombreux et

éparpillés dans différents laboratoires sur le globe. La solution doit donc être développée soit au niveau des modules du noyau soit au niveau des bibliothèques. Ainsi elle sera effective à tous les processus, pour le moindre effort de la part des utilisateurs.

Plus on se rapproche du matériel, plus le niveau d'abstraction est faible et plus il est difficile d'identifier l'objectif de l'utilisateur. Le module *plx* ne propose que d'écrire sur le bus local ou dans des registres de configuration. S'il est possible de savoir dans quel but a été appelé le module, en fonction des paramètres, il est difficile de déterminer la section critique de l'application de l'utilisateur à ce niveau. La section critique est la région du code où le processus accède à des ressources et où aucun autre processus ne doit y accéder.

L'intervention au niveau des bibliothèques assure déjà une meilleure identification de la section critique. Intervenir au niveau des bibliothèques implique de manipuler des ressources du système dans le monde utilisateur, et de les partager entre différents processus de différents utilisateurs. De plus, une telle solution peut être directement utilisée dans les programmes si les utilisateurs souhaitent une exclusivité absolue lors du déroulement de leurs applications.

Par conséquent, j'ai décidé d'implémenter une solution au niveau des bibliothèques, pour avoir la section critique la plus large possible et pour fournir une solution plus paramétrable pour les utilisateurs.

Travailler dans le monde utilisateur implique cependant de nouvelles contraintes, il faut que l'implémentation soit la plus propre possible au niveau du système, malgré les nombreuses interactions possibles entre les processus, les utilisateurs, et les ressources et les mécanismes utilisés.

2.3 Choix de conception

L'objectif est clair, il faut partager des mécanismes de synchronisation entre les processus. Pour cela UNIX fournit des mécanismes nommés IPC, comme Interprocess Communication. Le langage pour développer l'interface entre les utilisateurs et le matériel, donc le système d'exploitation, ses modules et les bibliothèques, est le langage C, j'ai donc continué de l'exploiter pour développer ma solution.

Les implémentations d'IPC disponibles sous les systèmes UNIX sont, entre autres, System V et POSIX. Ces deux implémentations sont différentes au niveau de leurs performances et de leurs utilisations. Si l'implémentation POSIX est d'une manière générale plus légère et facile d'utilisation, elle fournit aussi moins d'options et moins de contrôle que l'implémentation System V [ROCHKIND 2004].

J'ai donc porté mon choix sur les IPC System V pour certaines fonctionnalités que je vous présenterai en même temps que les mécanismes utilisés.

2.3.1 Synchronisation

Les mécanismes de synchronisation sont principalement les sémaphores. Ce sont en fait des compteurs que l'on utilise pour protéger des sections critiques. Le principe est que chaque processus qui souhaite entrer dans la section critique décrémente le compteur, si la valeur devient négative alors il doit s'endormir jusqu'à ce qu'elle redevienne positive ou nulle. Alors le processus peut entrer et travailler dans la section critique. Ensuite lorsqu'il la quitte il incrémente le compteur et si un processus était en attente, il est réveillé.

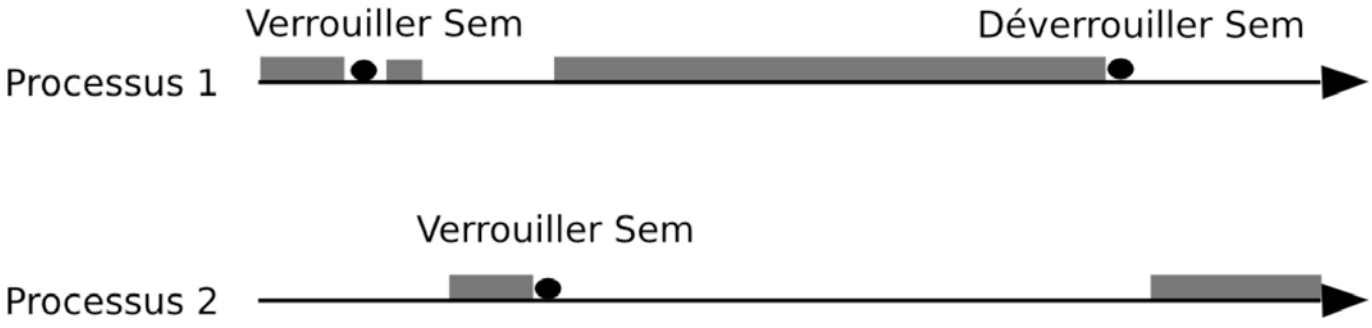

Figure 7 Schéma de synchronisation de processus

Les implémentations des sémaphores proposent de nombreuses options dérivées du mécanisme de base. Elles sont plus ou moins utiles et alourdissent le mécanisme.

Il existe une solution plus légère que les sémaphores, nommée Mutex, comme Mutual Exclusion. Ce sont des sémaphores binaires, la valeur du compteur est soit 1, ou libre, soit 0, ou verrouillé, un seul processus a accès à la ressource critique. La contrepartie est que ce mécanisme est implémenté pour des threads, ou processus légers. Un Mutex n'est donc pas partagé entre les processus nativement comme les sémaphores. L'implémentation courante des Mutex sous Linux est fournie par la bibliothèque NPTL (Native POSIX Thread Library), elle est basée sur un mécanisme introduit avec les noyaux 2.6 qui est le Futex (Fast Userspace muTEX).

Ce mécanisme a été introduit car les anciens Mutex et les sémaphores gaspillent du temps. En effet, pour garantir l'atomicité du code, la gestion de ces ressources est effectuée dans le monde noyau, ce qui implique des changements de contextes assez lourds.

Le processus s'exécute dans le monde utilisateur et demande au noyau de réaliser une opération sur ces ressources lors d'un appel système. Cet appel se traduit par un changement de contexte pour passer du monde utilisateur au monde noyau, puis par un second changement pour retourner au monde utilisateur. Ceci implique des milliers d'instructions primitives qui dépensent les ressources du CPU.

Or le noyau n'est nécessaire que pour arbitrer les conflits, qui arrivent statistiquement peu souvent mais qui sont fatals. Il est alors possible de ne faire appel au noyau que lorsqu'il doit arbitrer un conflit sur le mécanisme de synchronisation.

Le Futex exécute donc toutes les opérations, tests, incrémentations et décrémentations, dans le monde utilisateur, et ne fait appel au noyau que pour gérer la mise en sommeil et l'éveil des processus.

J'ai finalement porté mon choix sur les Mutex NPTL à cause de leur légèreté, et aussi pour respecter la contrainte de la propreté du système. En effet, les sémaphores ont une entrée dans le noyau et réservent un identifiant, elles utilisent des ressources et peuvent générer des conflits. Mais le vrai problème est qu'il n'est pas possible de déterminer combien de processus ont accès au sémaphore et si l'un d'eux va le verrouiller. Il serait donc impossible dans le projet de déterminer à quel moment on peut libérer le sémaphore.

J'utilise tout de même un sémaphore pour compter le nombre de processus qui accèdent à la Glue-Card. Ainsi les processus sont prévenus qu'ils ne doivent pas déconnecter la Glue-Card de la CCPC. Ceci est réalisé avec une fonctionnalité des sémaphores System V qui modifie leur comportement pour que les processus attendent que le compteur soit nul avant de rentrer dans la section critique.

Une autre fonctionnalité très intéressante des sémaphores System V est l'ajustement. Le système conserve pour chaque processus une valeur qui correspond au négatif des modifications qu'a effectué le processus sur le sémaphore. Ainsi si le processus s'achève de manière anormale le système ajoute cette valeur à la valeur du sémaphore pour annuler toutes les actions du processus terminé. L'état du sémaphore est maintenu correct même en cas de problèmes inattendus.

Le choix des Mutex m'oblige à utiliser un moyen de les partager entre les processus. Ce moyen est le segment de mémoire partagée. Les segments de mémoire partagée System V me permettent entre autres de régler le problème posé par la libération des sémaphores.

2.3.2 Partage

Les segments de mémoire partagée consistent à permettre aux processus de partager des pages de mémoire physiques à partir de leur espace d'adressage. La mémoire partagée est donc accessible depuis un pointeur. Typiquement, on définit une structure qui représente le contenu de la mémoire partagée, ainsi tous les processus ayant accès à la mémoire maîtrisent son contenu.

Un segment de mémoire partagée peut être une source de conflit si plusieurs processus accèdent à la même variable simultanément, son contenu peu devenir erroné. L'utilisation de segment de mémoire partagée nécessite l'utilisation conjointe de mécanismes de synchronisation. Ici nous veillerons à ce qu'un processus ne modifie pas les variables contenues dans le segment avant qu'il ne possède l'exclusivité sur la ressource matérielle souhaitée et donc sur la mémoire partagée.

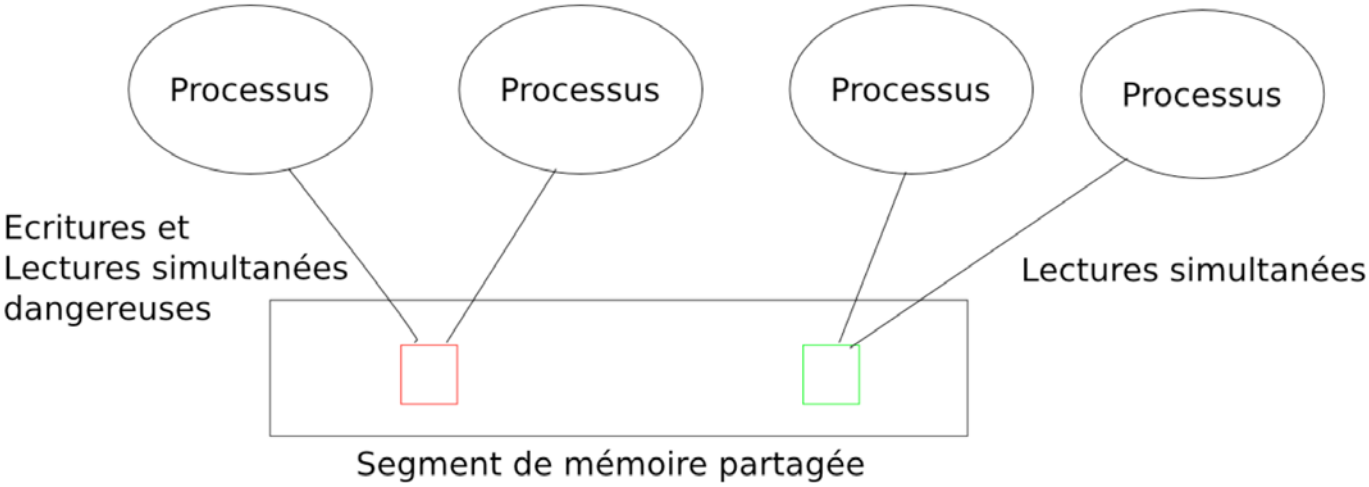

Figure 8 Schéma de l'utilisation d'un segment de mémoire partagée

L'implémentation System V permet de contrôler efficacement les attributs du segment de mémoire partagée, et de savoir combien de processus l'utilisent. Ceci au niveau du code mais aussi au niveau du système, ce qui assure une meilleure gestion des ressources.

L'un des avantages de l'utilisation du couple Mutex/mémoires partagées par rapport aux sémaphores est que l'on sait si un processus est connecté à la mémoire partagée. On peut donc déterminer quand libérer les ressources du système.

Un problème vient du fait que seul un processus ayant l'UID (User IDentifier) du processus créateur du segment de mémoire partagée peut le marquer comme libérable. Ceci permet habituellement de terminer le processus créateur et de laisser les autres processus ayant accès à la mémoire partagée finir leur travail et se terminer avant de la libérer. Mais une mémoire marquée comme libérable n'accepte plus de connexion de la part de nouveaux processus donc si le processus créateur se termine en annonçant la libération de la mémoire, le prochain processus qui souhaite accéder à la Glue-Card créera un nouveau segment avec de nouvelles ressources, et la synchronisation de processus ne sera pas effective.

Ce problème est résolu en ne libérant la mémoire qu'une fois que le dernier processus se termine. Si le créateur n'est pas le dernier processus à se terminer, il transfère les droits sur le segment de mémoire partagée au dernier processus qui a effectué une modification de son état, ainsi la propriété du segment tourne entre les processus jusqu'à sa libération.

La combinaison des Mutex avec les mémoires partagées permet d'obtenir un code plus léger qu'avec uniquement des sémaphores, et participe au respect des contraintes concernant la libération des

ressources du système. La libération du sémaphore utilisé s'effectue en même temps que la libération du segment de mémoire partagé qui lui est associé.

2.3.3 Bibliothèques

J'ai donc développé la solution au niveau des bibliothèques partagées, en utilisant des mécanismes de synchronisation légers stockés dans des segments de mémoire partagée. J'ai choisi de développer cette solution au sein d'une nouvelle bibliothèque, *exclulib*, pour ne pas avoir à trop intervenir dans les bibliothèques existantes et pour éviter la redondance de code.

Le principe d'intervenir dans les bibliothèques assure aux utilisateurs de ne pas avoir à modifier leurs projets pour prendre en compte les nouvelles fonctionnalités. En effet, les bibliothèques partagées sont chargées dynamiquement à l'exécution du programme.

Leurs versions reposent sur une série de trois numéros, le majeur, le mineur et la correction mineure. Seule l'incrémentation du numéro majeur, qui définie une mise à jour importante avec des possibilités d'incompatibilités, nécessite une nouvelle édition de lien des applications avec la bibliothèque, et donc une intervention des utilisateurs.

Par conséquent j'incrémenterai le numéro mineur pour ne pas obliger les utilisateurs à recompiler leurs applications. Sous Linux, une bibliothèque partagée installée correctement assure même que les processus en cours d'exécution se poursuivront en appelant les fonctions de la nouvelle bibliothèque.

Les bibliothèques fournissent aussi une fonctionnalité utile pour respecter toujours plus nos contraintes. Ce sont les constructeurs et les destructeurs. Ces deux procédures sont appelées respectivement au chargement et au déchargement de la bibliothèque. L'appel de la procédure de destruction permet d'assurer le nettoyage des ressources en fin de processus.

Cependant, cette procédure n'est appelée que lorsque le programme se termine normalement. Si une erreur se produit, il en résulte que le processus reçoit un signal qui met fin à son exécution. Il suffit alors de connecter les signaux les plus courants à la procédure de terminaison normale d'un processus, *exit()*.

Les avantages des bibliothèques sont nombreux, ils interviennent aussi bien dans l'implémentation de la solution que dans sa distribution.

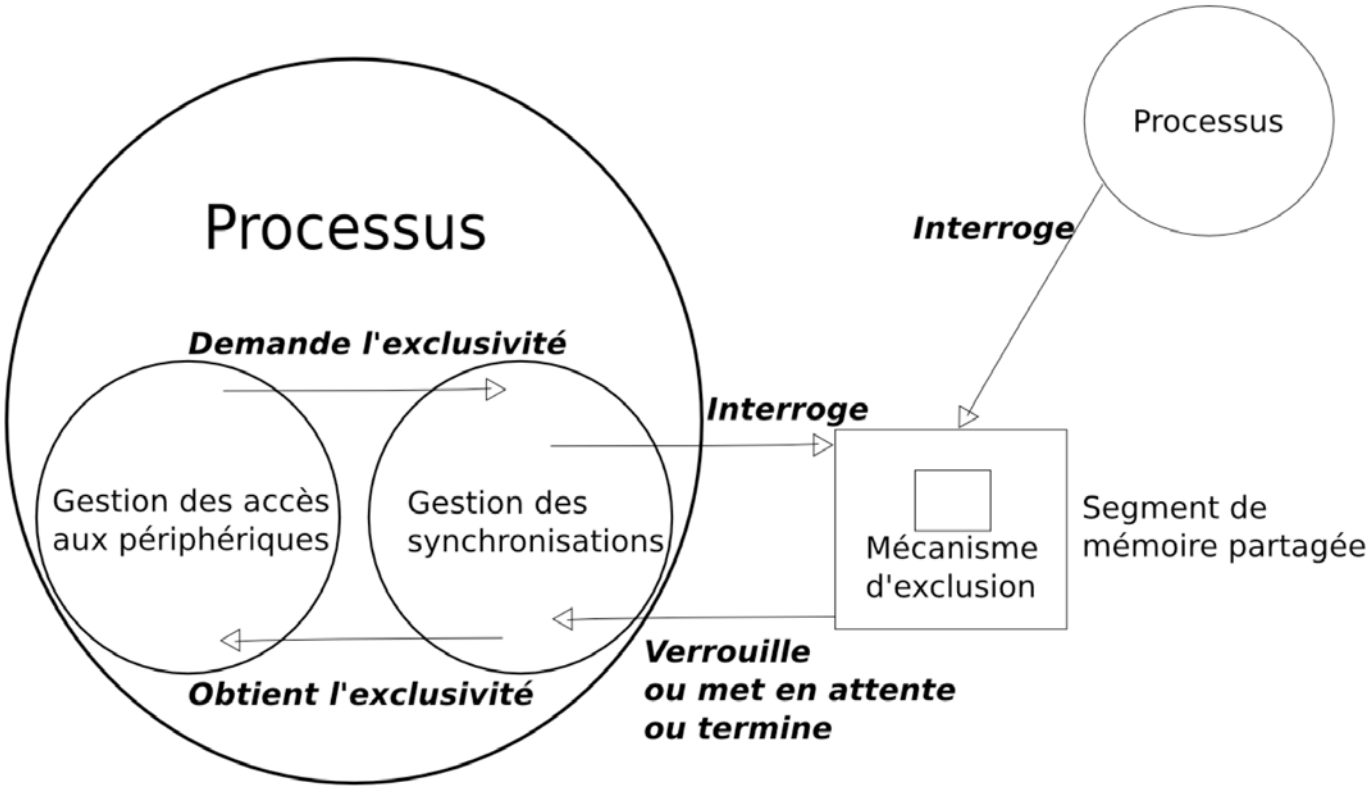

Figure 9 Schéma simplifié des interactions entre les mécanismes impliqués

2.4 Stratégie de développement

Mon projet intervient au sein d'un environnement déjà fonctionnel mais imparfait. Cet environnement est en cours d'utilisation par de nombreux utilisateurs. La solution la plus sage pour commencer le développement était de me dédier une carte TELL1 pour que je puisse ébaucher une première solution et que j'effectue mes premiers tests sans nuire à personne.

Les tests allaient de comparaisons de performances entre plusieurs solutions aux stress tests destinés à éprouver la solution dans des conditions d'utilisation peu communes pour étudier ses limites.

Une fois les tests validés, je déployais les nouvelles bibliothèques sur un premier jeu de cartes TELL1 utilisées pour faire des tests en interne au LHCb. Puis je les déployais au sein de l'ensemble dit

« production » du LHCb, qui est l'endroit où tout ce qui est développé est testé, pour enfin les déployer sur les cartes TELL1 du détecteur.

Toute cette procédure est bien entendue itérative et chaque problème soulevé lors des tests nécessitait une restauration des systèmes antérieurs si jamais le déploiement avait été effectué, puis une étude approfondie pour réussir à reproduire les erreurs afin de les identifier et de les résoudre, avant de recommencer la procédure de tests et de déploiement.

2.5 Réalisation

Le développement et l'intégration de la solution dans l'existant se sont faits successivement. Une fois la bibliothèque de synchronisation implémentée, il était possible de l'intégrée dans les bibliothèques existantes pour ensuite déployer la solution complète sur les systèmes du LHCb.

2.5.1 Implémentation

Le mécanisme de verrouillage consiste donc en une mémoire partagée contenant un Mutex et quelques autres informations, comme un PID (Process Identifier) pour savoir si la mémoire a été initialisée, et par quel processus, ainsi que différents identifiants des ressources utilisées.

La bibliothèque *exclulib* fournit de quoi travailler sur ce mécanisme de verrouillage. Des fonctions permettent de créer, verrouiller, déverrouiller, supprimer et contrôler ces mécanismes, à plus ou moins haut niveau.

C'est-à-dire que, dans un soucis de polyvalence et de réutilisation de cette bibliothèque, elle fournit les fonctions de bas niveau pour manipuler simplement les mécanismes de synchronisation avec les opérations atomiques habituelles, et elle fournit aussi de quoi manipuler les mécanismes de synchronisation en accord avec les algorithmes nécessaires pour l'utilisation sur la Creditcard PC. Le principe est l'encapsulation des fonctions de bas niveau dans des fonctions de plus haut niveau,

permettant à l'utilisateur de la bibliothèque de ne pas avoir à se concentrer sur les mécanismes de synchronisations qu'il va utiliser mais plutôt sur son travail.

Ainsi l'utilisateur ne fait que demander un accès à la section critique du bus qu'il souhaite utiliser. La bibliothèque se charge alors de créer les segments de mémoire partagée ou de s'y attacher et de verrouiller les mécanismes. Le tout en respectant l'architecture logicielle et matérielle de la Glue-Card.

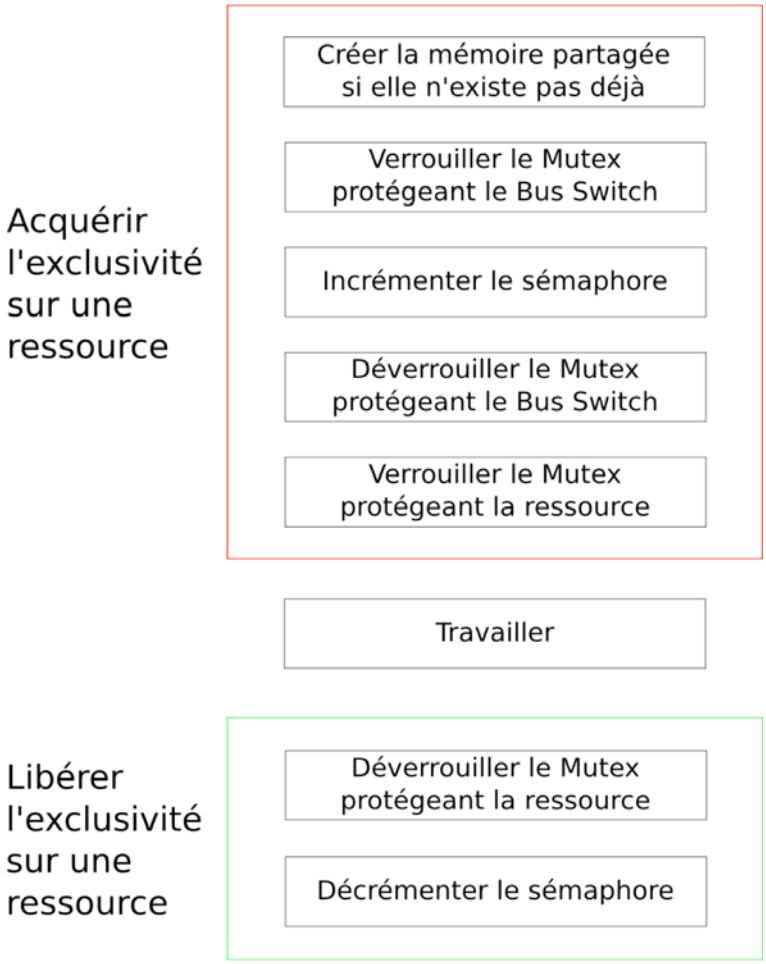

Figure 10 Principe de fonctionnement général

Ce principe subit de légères modifications en fonction du but du travail. En effet, certains accès aux périphériques de la Glue-Card sont temporaires alors des processus en attente peuvent ensuite accéder à ces périphériques et modifier leurs configurations. Mais d'autres actions sont destinées à configurer la

Glue-Card ou la carte TELL1 de manière durable. Dans ce cas, il faut éliminer les processus qui attendent pour détruire ces travaux et avertir leurs utilisateurs du problème qui peut être causé par leur démarche.

C'est le cas par exemple de la programmation des FPGA. En effet, si un utilisateur programme un FPGA, il ne souhaite pas être interrompu mais il ne souhaite pas non plus qu'un autre processus qui attend d'avoir accès au bus JTAG reprogramme le FPGA ensuite.

La protection de l'ouverture du *bus switch* nécessite elle aussi un traitement particulier.

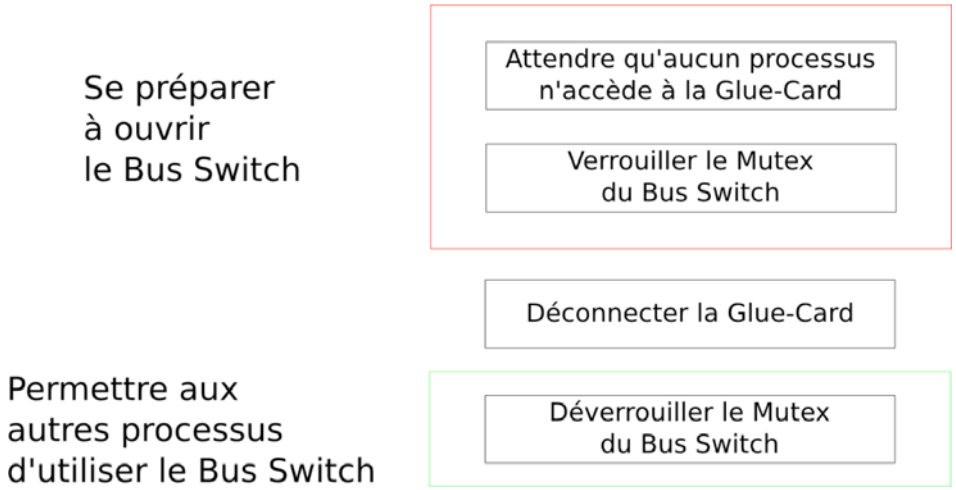

Figure 11 Principe de protection de la déconnexion du Bus Switch

On voit clairement, en comparant les figures 10 et 11, que la priorité revient aux processus travaillant sur la Glue-Card plutôt qu'à ceux qui la déconnectent de la Creditcard PC.

Toutes ces procédures de synchronisation sont imbriquées dans une procédure plus globale, relative au cycle de vie du processus. Le principe de fonctionnement de la bibliothèque tout au long de ce cycle de vie est le suivant :

- Un processus souhaite accéder à une ressource matérielle, il demande le mécanisme de synchronisation associé à la ressource, grâce à un identifiant.
  - Un segment de mémoire partagé est créé si aucun n'existe déjà.
  - Le segment de mémoire partagé existant est attaché au processus.
- Une fois le segment de mémoire attaché, il le reste jusqu'à la fin du processus.

- À la fin du processus, s'il est le dernier à être attaché au segment alors il le libère, sinon il se détache simplement.

Comme la plupart des programmeurs de la Creditcard PC sont avant tout des physiciens et que le système Linux n'est pas leur spécialité, j'ai longuement étudié comment gérer au mieux la bonne libération des ressources en fin de programme pour ne pas générer de conflits sur des résidus de ressources mal libérés et éviter une intervention de l'utilisateur à ce niveau-là.

Il y avait selon moi trois solutions possibles. La première consistait à se baser sur le daemon *ccserv*, à admettre qu'il était présent sur toutes les Creditcard PC et qu'il accédait lui aussi aux périphériques de la Glue-Card. Dans ce cas, comme il s'exécutait infiniment, il pouvait créer et conserver les droits sur les ressources du système, en les partageant avec d'autres processus.

La seconde solution aurait été de développer un autre daemon qui serait chargé uniquement du contrôle des mécanismes de synchronisation et des segments de mémoire partagée. Ainsi ce daemon libérerai les ressources du systèmes quand il les verrait non utilisées.

J'ai éliminé ces deux idées car elles dépendaient d'un mécanisme externe à la bibliothèque de synchronisation, le *ccserv* n'étant pas obligatoirement présent ou utilisé.

La solution que j'ai retenue a été de faire en sorte que la propriété des ressources du système se modifie avec les processus qui les utilisaient. Un processus en fin de vie va exécuter la routine de destruction de la bibliothèque.

Son principe est le suivant :

- Déverrouiller le mutex au cas le processus ne se termine pas normalement, si jamais il n'était pas verrouillé ou si jamais un autre processus l'avait verrouillé rien ne se passe.
- Si on libère la ressource associée au Bus Switch, libérer le sémaphore.
- Gérer le segment de mémoire partagée.
    - Si on est le dernier processus attaché au segment de mémoire partagée, marquer le segment comme libérable.
    - Sinon, modifier les droits du segment pour que l'UID du dernier processus à avoir fait une modification soit le propriétaire du segment.
- Se détacher.

Ainsi à tout moment un des processus a les droits de suppression des ressources, et le mécanisme reste interne à la bibliothèque.

Il arrive que les droits reviennent au *ccserv* s'il a utilisé les bibliothèques à un moment donné, par conséquent ce sera ce processus qui sera en charge de la libération des ressources. Ceci ne devrait jamais se produire. Mais comme tous les processus, il partagera les ressources avec les autres, alors tous les programmes se dérouleront comme prévus.

La bibliothèque est ainsi fonctionnelle. Elle est adaptée à chaque cas d'utilisation du couple CCPC – Glue-Card sur la carte TELL1. Elle maintient le système dans un bon état de fonctionnement et n'implique pas l'utilisateur. Il ne reste plus qu'à l'intégrer dans l'existant pour éprouver son efficacité et enfin distribuer la solution globale sur les systèmes du LHCb.

2.5.2 Intégration

Au niveau des bibliothèques

L'intégration de la bibliothèque dans l'architecture logicielle existante à d'abord consisté en l'analyse du code pour identifier correctement toutes les sections critiques. Dans la plupart des cas, cela a consisté à localiser l'endroit où se trouvait un appel aux fonctions de la bibliothèque *lblib* et à entourer cet appel d'un verrouillage et d'un déverrouillage du Mutex associé à la ressource.

Les bibliothèques possèdent donc une section critique protégée, adaptée à chaque ressource de la Glue-Card. De plus, un utilisateur expérimenté peut assurer lui-même la synchronisation de son application.

Recompiler les bibliothèques en n'incrémentant que le numéro mineur de leurs versions ne nécessite pas une nouvelle édition de lien avec les applications des utilisateurs. Alors tout a été installé subrepticement sur les nombreuses Creditcard PC. Et les utilisateurs ont directement utilisé la nouvelle solution proposée par les bibliothèques les plus récentes.

Au niveau du système

L'installation s'est faite au moyen de RPM (Red Hat Packet Manager) et de Yum (Yellow dog Updater, Modified). RPM est le mécanisme standard d'installation de paquets sous les distributions Red Hat de Linux, il consiste en un paquet contenant des fichiers et des instructions pour les installer correctement sur le système [BAILEY et Coll. 2000]. Yum est un logiciel d'installation, de suppression et de mise à jour automatique pour les systèmes basés sur RPM.

Ainsi l'installation a été effectuée en créant de nouveaux paquets pour les bibliothèques mises à jour et pour la nouvelle bibliothèque de synchronisation. En plaçant ces paquets dans le dépôt de Yum, celui-ci met à jour tous les systèmes ayant installé les anciennes versions des bibliothèques. Le paquet gérant la bibliothèque de synchronisation est automatiquement inclus car il figure sur la liste de dépendances des nouvelles versions. Alors l'installation s'effectue automatiquement sur les 300 cartes TELL1 de l'expérience LHCb et sur toutes les autres cartes basées sur la Creditcard PC.

3 La gestion du Watchdog

Ce projet consiste principalement en la réalisation de défis techniques. Peu de monde intervient à ce niveau dans l'implémentation d'un système. De plus, les problèmes à résoudre sont très spécifiques aux conditions de l'environnement de l'expérience LHCb et de la Creditcard PC.

3.1 Présentation de l'existant

La Creditcard PC est redémarrée en cas de problèmes par un watchdog matériel [DIGITAL 2003]. Le principe du watchdog est de compter jusqu'à une valeur, configurée par registres. Si le compteur a atteint la valeur limite, alors le watchdog envoie un signal matériel au processeur qui redémarre la CCPC. Il suffit de réinitialiser le compteur avant cette échéance pour que la CCPC continue de fonctionner.

Le watchdog de la Creditcard PC est utile pour redémarrer les cartes sans intervention humaine, dans le détecteur, en cas de problèmes au niveau du système d'exploitation. Cependant, le délai maximal est de 30 secondes, ce qui représente un délai assez restreint compte tenu, principalement, de l'environnement dans lequel les Creditcard PC du LHCb sont utilisées. Les cartes démarrent à partir d'un réseau. Si elles démarrent toutes au même moment, il peut arriver que le réseau soit surchargé et ne puisse plus assurer son service correctement.

Sous Linux, le watchdog du microcontrôleur AMD Elan 520 est géré par un module du noyau et par un daemon. Le noyau est la couche logicielle qui permet de faire abstraction de la couche matérielle. Le module est le pilote de périphérique.

Le module du noyau est le *SC520*, en fait ici ce n'est pas vraiment un module, il n'est pas chargé comme tel une fois le démarrage du noyau Linux fini, mais il est directement incrusté dans le noyau à sa compilation, pour être chargé et initialisé au même moment.

Ceci s'explique car ce module ne doit pas être optionnel sur la Creditcard PC, et il doit être chargé le plus rapidement possible, ce qui implique au même moment que le noyau.

Le pilote de périphérique SC520 est chargé de faire le lien entre le monde matériel et le monde logiciel. Il interface les registres propres au watchdog et permet aux utilisateurs de communiquer avec lui. Seul il ne peut pas être efficace pour activer le watchdog car il n'exécute rien à proprement parlé, il ne fournit qu'un moyen d'accéder au watchdog. C'est pour cela qu'il travaille en duo avec un daemon.

Un daemon est, sous UNIX comme sous n'importe quel système d'exploitation, un processus exécuté en arrière-plan et non pas sous le contrôle direct d'un utilisateur. Sous UNIX, ces processus sont en principe exécutés au démarrage du système. Ils fournissent en général des services réseau ou des contrôles sur des périphériques matériels, comme dans le cas présent avec le watchdog.

Notre daemon, *wd_daemon*, a pour rôle de communiquer avec le watchdog, par l'intermédiaire du pilote de périphérique, au moins une fois toutes les 30 secondes pour éviter le redémarrage du système.

3.2 Le démarrage de la Creditcard PC

3.2.1 Le problème

Le démarrage des Creditcard PC est effectué à partir du réseau grâce à un module du BIOS. Ce module, Etherboot, émet des requêtes de broadcast pour atteindre un serveur DHCP (Dynamic Host Configuration Protocol) qui lui indiquera alors l'adresse d'un serveur TFTP (Trivial File Transfer Protocol). Ce dernier serveur lui enverra alors une Network Boot Image, un fichier contenant le noyau Linux et un Initial RamDisk (initrd). L'initrd est un disque RAM contenant un système de fichier minimaliste pour permettre au noyau de s'exécuter dans un premier temps.

Une fois l'image de démarrage téléchargée, le noyau est décompressé et exécuté sur le système de fichier de l'initrd. Ainsi le noyau peut utiliser le contenu de ce périphérique pour démarrer et ensuite

charger les modules de gestion des périphériques de systèmes de fichiers, comme un disque dur, ou dans notre cas pour monter un système de fichier distant via NFS (Network File System).

Le noyau Linux fournit au *wd_daemon* de quoi communiquer avec le watchdog pour prévenir d'un redémarrage au début de son exécution. Ceci est effectué le plus tôt possible, dès le chargement du noyau. Le daemon est exécuté une première fois dans le système de fichier de l'initrd pour que la procédure de démarrage ne soit plus interrompue par un redémarrage du watchdog à partir de ce niveau.

Une fois le système de fichier distant accessible, le noyau change de système de fichier racine et exécute le processus d'initialisation qui va relancer le *wd_daemon*, une fois pour toutes.

Le noyau spécialement conçu pour la Creditcard PC a été allégé au maximum pour accélérer la procédure de téléchargement, de décompression et de démarrage. De plus, le watchdog est réinitialisé une première fois alors que le noyau initialise le pilote de périphérique du watchdog. Cependant, sur un réseau saturé en requêtes, le temps entre le démarrage du BIOS d'une CCPC et l'exécution du pilote de périphérique qui assure le premier gain de temps peut largement excéder 30 secondes.

Il est impossible d'intervenir avant que l'image ne soit récupérée depuis le réseau, il faut donc souhaiter que le téléchargement du matériel nécessaire au démarrage se fasse en un temps inférieur à 30 secondes. Pour intervenir le plus rapidement possible, il ne reste plus qu'à étudier et comprendre le code source gérant le démarrage du noyau Linux, et à intervenir à ce niveau en ajoutant les quelques lignes nécessaires pour écrire dans les registres du watchdog et ainsi le réinitialiser.

3.2.2 Le code du démarrage de Linux

Le noyau nécessite d'être chargé en mémoire, pour cela le BIOS fait appel à un chargeur d'amorçage. Ce dernier se place en RAM et effectue quelques opérations comme la configuration de la pile en mode réel ou l'affichage d'informations à l'écran, avant de charger le code du démarrage du noyau en mémoire et de l'exécuter.

Lorsqu'on étudie l'arborescence du code source du noyau, on trouve des fichiers propres à chaque architecture sur lesquelles le noyau Linux est susceptible de s'exécuter. Ses codes spécifiques sont notamment en charge du démarrage du noyau. On trouve ainsi un petit chargeur d'amorçage, utilisé lorsque le noyau doit être démarré à partir d'une disquette. Ensuite vient un fichier, *setup.S*, dédié à l'initialisation des périphériques matériels et à la configuration de l'environnement (RAM, …), pour la bonne exécution du noyau. Enfin, le dernier fichier décompresse le noyau si nécessaire et l'exécute [WANG 2004].

Donc, le premier code du noyau Linux qui est exécuté suite au secteur d'amorçage est la fonction *Setup()*. C'est donc l'endroit où nous pouvons intervenir pour répondre à notre besoin.

Ce code est de très bas niveau. Il est écrit en assembleur et adresse la mémoire en mode réel. C'est-à-dire qu'aucune protection de la mémoire et aucune gestion de la mémoire virtuelle n'est effectuée. On peut donc, à partir de ce mode, écrire dans les registres du watchdog pour le réinitialiser.

Les quelques instructions nécessaires à la correction du problème sont une sauvegarde de contexte et une restitution de ce contexte, qui entourent la détermination de l'adresse du registre du watchdog et l'écriture de quelques valeurs dans ce registre, selon le protocole pour réinitialiser le watchdog.

Ainsi nous devrions gagner 30 secondes supplémentaires pour décompresser et placer le noyau en mémoire. Ce qui serait amplement suffisant.

3.3 Le daemon du watchdog

3.3.1 Le problème

La Creditcard PC utilise un système de fichier distant partagé à travers un réseau via NFS. Il peut arriver que le serveur NFS soit inaccessible pour diverses raisons, comme une panne de réseau ou bien le daemon gérant NFS côté serveur s'est terminé de façon incorrecte, … Dans ces cas-là, la Creditcard PC ne redémarre pas, mais devient inutilisable.

En effet, le daemon chargé de réinitialiser le watchdog est un processus, il est donc situé en RAM et ne remarque pas l'absence du système de fichier. Il continue donc de remplir son rôle.

Il peut être souhaitable de redémarrer la carte dans le cas d'un problème matériel qui obligerait le changement de serveur NFS. L'une des contraintes à respecter pour cette fonctionnalité est un redémarrage après un temps paramétrable.

Une autre contrainte est que l'opération ne doit pas retarder le daemon dans sa tâche principale pour le cas où il n'y a aucun problème. Il faut que le test que nous effectuerons soit le plus rapide possible pour ne pas essuyer un redémarrage intempestif si jamais l'ordonnanceur du système rend la main trop tardivement au daemon par rapport à son échéance.

3.3.2 La solution

Il n'est pas possible de vérifier l'état du serveur NFS depuis un processus client. La méthode que j'ai décidée d'employer consiste à effectuer une opération de lecture sur un fichier.

Le système de fichier NFS est monté[1] par défaut avec l'option « hard », qui signifie qu'une opération sur un fichier alors que le serveur NFS est inaccessible tournera en boucle jusqu'à ce que la situation se débloque. Si aucune contrainte ne pesait sur ce projet, la solution était toute trouvée. En effet si l'opération sur le fichier boucle infiniment, le *wd_daemon* ne réinitialisera pas le watchdog et la CCPC redémarrera. Cependant, la solution est dans les options NFS.

L'option « soft » signifie qu'une erreur sera retournée si une opération sur un fichier ne peut pas aboutir à cause d'un problème au niveau du serveur NFS. Cette erreur est gérée de façon à faire entrer le daemon dans un nouvel état. Cet état est sensiblement le même que l'état normal du daemon, si ce n'est que le principe de celui-ci disparaît, à savoir qu'il ne bouclera pas infiniment mais pendant 15 minutes, le temps estimé nécessaire pour régler le problème du serveur NFS si cela est possible. Si le système de fichier devient accessible alors que le daemon est dans cet état de fin de vie, il retourne dans son état original.

[1] Monter : Attacher un système de fichier provenant d'un périphérique quelconque à l'arborescence principale du système.

Il n'était pas souhaitable de modifier toutes les options de tous les systèmes de fichiers de tous les serveurs de Creditcard PC, c'est pourquoi j'ai créé un nouveau fichier, monté en parallèle du système de fichier principal, pour permettre au daemon d'effectuer le test qui le fera changer d'état. Ce test est effectué le plus rapidement possible, en jouant avec les différentes options de l'ouverture de fichiers en langage C sous UNIX.

Ainsi, le daemon est maintenant capable de vérifier l'état du serveur NFS. Il effectue aussi d'autres tests sur son environnement pour vérifier son intégrité dans les conditions souhaitées par le LHCb. Par conséquent il est capable d'adapter son comportement pour redémarrer la Creditcard PC sur des environnements adéquats.

4 Résultats et discussion

4.1 Résultats de la synchronisation des processus

Les nombreux tests réalisés sur la bibliothèque l'ont éprouvée, au niveau de son objectif premier qui est la synchronisation de processus, comme au niveau des contraintes à respecter sur le système, la partie la plus difficile à mettre en oeuvre.

Les tests ont d'abord consisté à prouver qu'une synchronisation était bien effective. Ceci a consisté à exécuter des processus d'utilisateurs différents, de droits différents et de priorités différentes, et de les faire entrer conflit sur différentes ressources. Chaque test validé avec succès me permettait d'augmenter l'instabilité du système en incrémentant le nombre de processus et de ressources. Le résultat m'a satisfait quand les conditions d'utilisation normale d'une Creditcard PC étaient très largement dépassées et que le système était tout aussi propre à la fin de l'exécution de nombreux processus qu'à son démarrage.

Une fois la bibliothèque de synchronisation parfaitement éprouvée, je l'ai intégrée dans les bibliothèques de gestion des accès aux ressources matérielles et réalisé de nouveaux tests à partir des outils fournis par ces bibliothèques et à partir du serveur de contrôle *ccserv*. Ces tests avaient encore une fois pour objectifs de vérifier que les conflits étaient gérés correctement et qu'aucun résidu de subsistait dans le système une fois les applications terminées.

Les premiers tests étaient effectués sous mon contrôle direct sur une carte qui m'était dédiée, j'ai ensuite déployé mon travail d'abord sur une, puis sur l'ensemble des cartes TELL1 utilisées par un collègue au LHCb pour qu'il continue son travail en utilisant ma solution en me faisant part des problèmes qu'il rencontrerait.

Une fois que tous les tests précédents étaient validés, j'ai déployé l'ensemble des bibliothèques sur le serveur de production du LHCb. Pour cela j'ai utilisé le système de distribution de paquets de Red Hat.

De nombreux scientifiques travaillent de par le monde sur ce serveur. Il était possible de revenir à tout moment à la version logicielle précédente en installant les versions antérieures des paquets RPM.

L'installation sur ces serveurs étant concluante, j'ai déployé les bibliothèques sur le serveur des cartes du détecteur LHCb.

Cette synchronisation permet de conclure le développement des interfaces logicielles de la Creditcard PC. La bibliothèque de synchronisation est principalement dédiée aux cartes TELL1, mais elle est aussi fonctionnelle sur toutes les cartes à bases de CCPC de l'expérience LHCb.

4.2 Résultats des améliorations de la gestion du watchdog

Les tests des nouvelles fonctionnalités du daemon consistaient à mettre ce dernier devant chaque cas d'erreur et vérifier qu'elle était bien gérée selon nos souhaits. Une fois les tests unitaires réalisés sur mon set de développement validés, j'ai déployé la solution sur un nombre de cartes plus étendues pour valider de nouveau le comportement du daemon, pour ensuite le distribuer à l'ensemble des Creditcard PC.

Il est en revanche plus compliqué de vérifier le bon fonctionnement de la réinitialisation du watchdog pendant la séquence de démarrage du noyau. Au moment où j'écris ces lignes, je ne peux pas prouver que le watchdog est bien réinitialisé au début de l'exécution du noyau.

Les améliorations de la gestion du watchdog ont impliqué la création de nouveaux paquets RPM, certains sont propres aux environnements des CCPC, mais cette fois ci d'autres sont nécessaires pour les serveurs TFTP et NFS.

4.3 Avancement des projets

4.3.1 Synchronisation des processus

Pour réaliser ce projet, il me fallait connaître de façon avancée le mécanisme qui permettait à la Creditcard PC d'accéder aux ressources de la TELL1, au niveau matériel et au niveau logiciel. La

documentation de la Glue-Card est cependant très légère, mais le code source des bibliothèques est bien commenté et documenté [NEUFELD 2007] et j'ai donc pu analyser les méthodes de communication avec le pilote de périphérique chargé de gérer la Glue-Card et son composant d'interface PCI - Bus parallèle PLX 9030.

J'ai aussi étudié les mécanismes de synchronisation disponibles sous UNIX. Si mon choix s'est rapidement porté sur les Mutex partagés pour répondre aux contraintes du projet, j'ai d'abord souhaité réaliser ma propre implémentation à l'aide des Futex.

J'avais pour objectif d'alléger encore les Mutex en me débarrassant des options que je jugeais inutiles au projet, afin d'obtenir un mécanisme de synchronisation très basique mais suffisamment complet pour son application, proposant, entre autres, la récursivité et la propriété. Les algorithmes pour gérer les Futex sont très simples, mais il faut posséder de bonnes connaissances en assembleur, en C et en ordonnancement [DREPPER 2005], pour les implémenter et les déboguer.

La première bibliothèque réalisée utilisait donc les Futex. Les premiers tests étaient satisfaisants, mais les conditions extrêmes des stress tests ne validaient pas ma bibliothèque. Une erreur critique se produisait aléatoirement au cours des quelques millions d'itérations exécutées.

J'ai alors commencé un assez long processus de déboguage. À ce niveau, les opérations sont les plus atomiques possibles, le moindre affichage, la moindre ligne de code inutile en trop modifie le comportement des processus. Il n'existe vraiment aucune méthode pour déboguer ce niveau d'abstraction car tout modifie le comportement des processus. De même les débogueurs sont à proscrire.

Le déboguage a donc consisté à étudier tous les cas de figures d'interactions entre deux processus, opération atomique après opération atomique, pour identifier le problème.

C'est ici que j'ai commencé à douter du bien fondé de ma démarche dans l'implémentation des Futex et j'ai poursuivi le développement de la bibliothèque à partir des Mutex, tout en consacrant du temps au déboguage de ma première implémentation.

Le problème de mon implémentation des Futex venait d'une erreur d'algorithmique qui faisait que l'ordonnanceur pouvait donner la main à un autre processus alors que le processus courant entrait dans l'appel système destiné à le faire patienter. Il en résultait un verrouillage interminable des deux processus. Une fois le problème identifié la correction devenait évidente, un simple test manquait pour que les processus d'exécutent correctement.

Je me retrouvais donc avec deux implémentations de la bibliothèque de synchronisation. Il ne me restait plus qu'à comparer leurs performances.

Pour cela j'ai réalisé différents tests, m'appuyant sur des variations du nombre d'itérations de chaque processus, de l'augmentation du nombre de tâches. Statistiquement, l'implémentation basée sur les Mutex est plus performante que celle qui est basée sur les Futex.

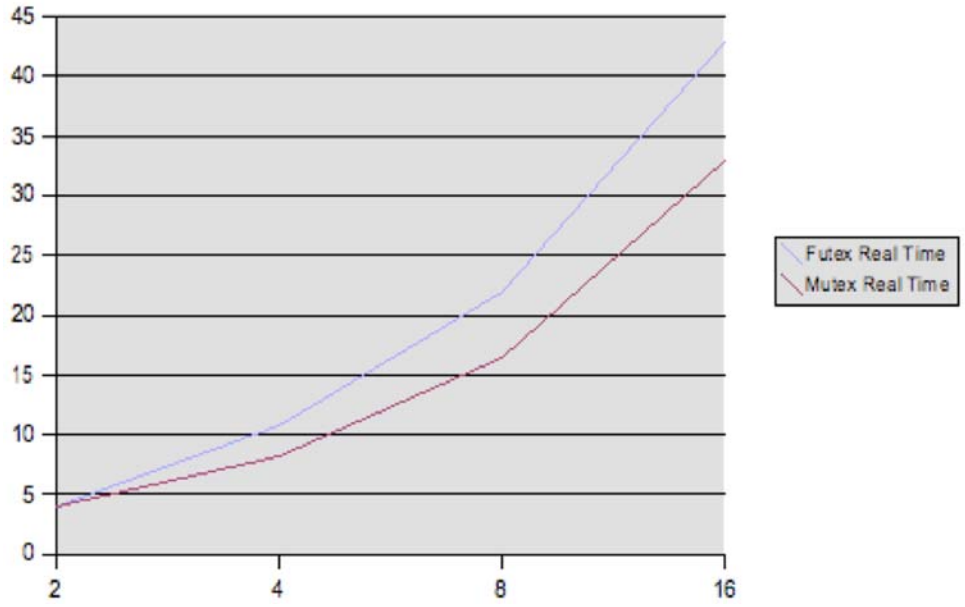

Figure 12 Courbes de la durée de vie moyenne des processus en secondes.

En abscisse le nombre de processus, en ordonnée le temps en secondes. En bleu la solution développée avec les Futex, en rouge les Mutex. Les tests ont été réalisés sur différents processeurs, en exécutant de longs processus, de centaines de milliers d'itérations.

Le problème est que statistiquement le temps passé dans les appels systèmes et dans le monde utilisateur sont équivalents, mais le temps total d'exécution du processus, qui prend en compte le temps où il est endormi, est bien plus élevé dans l'implémentation basée sur les Futex.

J'ai alors décidé de ne plus continuer qu'avec les Mutex de la NPTL, qui manifestement seraient toujours plus performants que ma propre solution, pour ne plus me consacrer qu'à la gestion correcte des ressources et à l'intégration de la synchronisation dans les bibliothèques existantes, pour finalement proposer la solution présentée plus haut.

Dans un soucis d'information, la documentation du code source et des interactions de la bibliothèque avec l'existant a été écrite en parallèle avec le code et mise en ligne en parallèle avec la distribution des bibliothèques.

4.3.2 Gestion du watchdog

Ce travail a demandé une importante recherche de documentation à propos du noyau UNIX, plus particulièrement à propos des mécanismes du démarrage d'un système Linux à partir du réseau et sur les possibilités de contrôle du système UNIX à partir d'un daemon sans faire intervenir le système de fichier.

Intervenir au niveau du démarrage du noyau UNIX n'est pas chose courante. Peu de documentation fiable existe et l'on ne peut guère que compter sur les commentaires du code source.

Cependant, en regroupant toutes les informations que j'ai pu trouver j'ai pu concevoir une solution sensée fonctionner, mais je suis toujours en train de réaliser des tests.

En ce qui concerne la surveillance du statut du serveur NFS, c'est un problème très spécifique auquel très peu de monde s'est déjà intéressé dans la communauté des développeurs UNIX. Trouver de la documentation concernant directement ce sujet est impossible.

Ma première approche a consisté à étudier si un processus s'exécutant en mémoire pouvait tout de même atteindre le système de fichier *proc*, qui est en fait une interface du noyau dans le système de fichier. Mais il n'était pas possible de l'atteindre par la voie conventionnelle, c'est-à-dire en l'utilisant comme des fichiers normaux. Toujours dans l'optique d'utiliser une interface avec le noyau, on trouve la fonction *sysctl* qui permet de lire et d'écrire des paramètres du système. Cependant, cette fonction ne permet que d'accéder à des options de configuration du noyau et non pas de contrôler l'état du système de fichier.

L'approche suivante a consisté à utiliser le système de fichier. En m'intéressant aux fonctionnalités de NFS, et en discutant avec les experts UNIX de l'équipe LHCb Online j'ai pu proposer la solution présentée auparavant dans ce rapport.

Ce projet englobant la gestion du watchdog tout au long du fonctionnement de la Creditcard PC a un résultat pour le moment plus mitigé. Mais le travail est toujours en progrès en ce qui concerne le démarrage.

Conclusion

Mon projet au sein de l'équipe LHCb a consisté à améliorer la gestion de l'accès aux ressources matérielles de la Creditcard PC et de la TELL1. La partie la plus importante qui consistait à assurer la synchronisation des processus qui accèdent aux périphériques de la carte TELL1 a été réalisée avec succès et mise en service. Le second projet est très bien avancé, et si la partie concernant l'amorçage du noyau Linux est susceptible de changer dans les jours qui suivent, la solution actuelle est prête à être distribuée.

Professionnellement parlant, le fait d'avoir développé deux implémentations de la bibliothèque de synchronisation est regrettable, il en a résulté une dépense de temps qui aurait pu être employé à d'autres tâches. Cependant cette expérience m'a ouvert les yeux sur la difficulté d'implémenter des mécanismes de synchronisation et m'a permis d'acquérir des techniques avancées dans l'optimisation de code et le déboguage multi processus.

Mon expérience au CERN ne se résume pas à ces seuls projets, je me suis impliqué dans l'équipe Online, au niveau de l'étude de l'investissement dans un nouveau module de contrôle plus récent pour remplacer l'actuelle Creditcard PC, et je me suis intéressé aux technologies matérielles et logicielles impliquées dans l'immense réseau du système DAQ pour participer à sa configuration.

La solution distribuée pour la synchronisation des processus a été éprouvée depuis maintenant quelques temps par les utilisateurs travaillant sur les cartes TELL1. Il reste toujours des améliorations possibles, notamment dans la prise en compte des différents objectifs du JTAG. Pour ce qui est de la gestion du watchdog, j'espère valider une solution pendant la période qui me reste à travailler au CERN.

Références Bibliographiques

[BAILEY et Coll. 2000] BAILEY Edward C., NASRAT Paul, SAOU Matthias, SKYTTÄ Ville, Maximum RPM, http://www.rpm.org/max-rpm-snapshot/, 2000.

[DIGITAL 2003] Digital Logic, Technical User's Manual for : SmartModule SM520PC, Nordstrasse 11/F, CH-4542 Luterbach, 2003.

[DREPPER 2005] DREPPER Ulrich, Futexes Are Tricky, http://people.redhat.com/drepper/futex.pdf, 2005.

[HAEFALI et Coll. 2005] HAEFALI G., BAY A., LEGGER F., LOCATELLI L., CHRISTIANSEN J., WIEDNER D., TELL1 : Specification for a common read out board for LHCb, LHCb, 2005.

[NEUFELD et Coll. 2007] NEUFELD Niko, GARNIER Jean-Christophe, Creditcard PC Online Software Documentation, http://lhcb-daq.web.cern.ch/lhcb-daq/ccpc/doc/index.html, 2007.

[NEUFELD 2004] NEUFELD Niko, Creditcard PC softwareguide, CERN Internal Note, 2004.

[ROCHKIND 2004] ROCHKIND Marc J., Advanced UNIX Programming Second Edition, Addison Wesley Professional Computing Series, 2004.

[WANG 2004] WANG Feiyun, Linux i 386 Boot Code HOWTO, http://tldp.org/HOWTO/Linux-i386-Boot-Code-HOWTO, 2004.

ANNEXES

Annexe A

Multiuser Safe Library and Watchdog Management

Internal Note

Issue:	1
Revision:	0
Reference:	LHCb-63-2007
Created:	August 27, 2007
Last modified:	August 27, 2007
Prepared by:	Jean-Christophe Garnier[a]
	[a]ISIMA, France

Abstract

This note introduces you to the primary project I worked on during my internship at CERN. This project involved the CCPC and consisted adding synchronization to the existing software, in order for the CCPC to be multi-user safe. This has been done using mutexes, semaphores and shared memories, working in libraries.

I also worked on the management of the watchdog. The daemon was not managing some troubles and it could lead us into troubles. Basically, the daemon had to be aware of the NFS server status. Another problem is that the watchdog maximum value is 30 seconds. This means that the CCPC has to boot from the network in a shorter time and it is not always possible. In order to earn 30 seconds as soon as possible, the watchdog timer has to be reseted after the kernel has been downloaded and before it is uncompressed.

Document Status Sheet

1. Document Title: Multiuser Safe Library and Watchdog Management			
2. Document Reference Number: LHCb-63-2007			
3. Issue	4. Revision	5. Date	6. Reason for change
Draft	1	August 6, 2007	First version.

Contents

1 Introduction

The TELL1 boards of the DAQ system are monitored by the CCPC. The CCPC is running a Linux Operating System, which is multi-user. Therefore, several users can run processes that would conflict on the access of the TELL1 devices. So processes have to be synchronized to use devices, which are critical sections. I will present and explain my choices in the implementation, and the way the solution works.

I will also speak about the modifications that I have done in the CCPC system to handle some possible errors of networking and file system, and to improve the boot procedure.

2 The CCPC in a few words

The CCPC is an embedded system developped by Digital Logic. Its aim is to configure and control the TELL1 board. It is booted from the network using etherboot, and its file system is mounted with NFS. An interesting feature is the hardware watchdog which permits to reboot the CCPC in case of problem. The watchdog is a counter which reboots the CCPC if it reaches a certain value, it has to be reseted every 30 seconds.

The CCPC provides tools to control and monitor the TELL1 board [CSG]. The interface between the CCPC and the TELL1 is done by the glue card, which interfaces the PCI bus of the CCPC to a parallel bus via a plx bridge, and then to I^2C and JTAG buses via a FPGA. The CCPC runs a Linux Operating system which allows several users to log in and work simultenously with the glue card. It consists in programming FPGA, write values on buses, etc. The problem is that nothing controls what the users are doing on the buses. Users can work using command line tools, through a PVSS interface connected to a CCPC daemon, cserv, or writing their own programs or scripts. So they can work on the same device and cause conflicts in their work.

Another feature of the glue card is that it can be disconnected from the CCPC via a bus switch. It consists of a registrer. Every process can disconnect the CCPC from the glue card, hence it has to be secured with a synchronization mechanism.

3 Multiuser safe library

3.1 Choice of IPC and implementation

The aim of this project is to provide synchronization on the CCPC, for the processes which are accessing its devices. It was possible to act in the kernel modules or in the libraries that manage the communication with the Glue Card.

Working in the kernel drivers would not have allowed a good identification of the critical section of each process. Indeed, a FPGA programmation through JTAG buses and a read on the I^2C devices has to be managed differently. Therefore I chose to work in the userspace.

In the userspace, the synchronization mechanisms themselves are basically mutexes and semaphores.

Semaphores are the most commonly known way to synchronize processes. It is basically a counter. A process which wants to access a resource tries to get a semaphore. To get a semaphore means to test its value, if it is less or equal to 0 it means that the semaphore is already locked by enough process, and then the process will wait until this value comes positive to continue its execution. If it is a positive value, then the process decrements the semaphore and can continue its execution.

Another synchronization mechanism is the mutex, which is a kind of binary semaphore, i.e. the maximum value of the counter is 1. So only one process can access a resource protected by mutexes. Semaphores are a heavy mechanism because they provide a lot of options (some are useful). Another reason is that semaphores imply one system call for every operation, contented or not. Futexes and Mutexes imply one system call for every operation only if they are contented. If they are not contented, there is no system call at all. Mutexes are much lighter, but they are not shared between real processes so they have to be stored in shared memories.

I chose to use mutex anyway, because of their weight, and also because the use of shared memories was a good way to implement a clean library. So far it is not possible to know if a process will lock a semaphore, it is possible to know if it is attached to a shared memory. Therefore it is easy to know if the shared memories should be cleaned or not in the end of a process.

I also use one semaphore to count how many processes are currently working through the plx bridge. I put this semaphore in a shared memory also, even if it is not really a matter of sharing, but to know when to destroy it, the same way as above. This needed a semaphore because it consists in counting how many processes are currently working on the glue card, if they are some, a process cannot close the bus switch and has to wait in order to perform that. This semaphore is a SYSTEM V semaphore. I

think there are two useful features in the use of the SYSTEM V implementation, although it is heavier in its use than the POSIX implementation [AUP].

The first useful feature is to be able to wait until the value becomes 0. Basically it is exactly what is needed to implement the bus switch counter. Each process accessing the glue card increment the semaphore, and each process which want to close the bus switch wait until the semaphore's value is 0.

The second useful feature is the adjustment. This is a mechanism provided by the SYSTEM V IPC and it is related to the process. It consists in, if the value of the semaphore is incremented, then the adjustment is decremented, if the value of the semaphore is decremented, then the adjustment is incremented. When a process exits, its adjustment value is added to the semaphore's value in order to cancel whatever it has done on the semaphore. It is useful in case of abnormal exiting.

As we are not the user programming the application to access to the glue card, it is very difficult to lock and unlock at the beginning and at the end of a critical section, so sometimes it would be possible that a lock was not released correctly. But this feature ensure us to always have a semaphore in a good state.

The shared memories are made with the SYSTEM V IPC, the unique keys are obtained with the file /tmp (in Unix, everything is a file) and the identities 0x10, 0x20 and 0x30. If you want to get a unique key with ftok, you should avoid these values else you would conflict with the running library. System V shared memories are heavier in their management than POSIX, but it provides the feature to control how many processes are linked to it.

A matter on the shared memories is that only a process of the same uid as the creator process uid has rights on it, e.g. can delete it. I manage these rights like a token, a dying process check if it was the latest to use the shared memory, if so it is the creator and it can destroy it, else, if it is the creator it gives the rights to another uid, the uid of the last attached process. Therefore, the running process will own the rights on the shared memory and it should be cleaned correctly in the end of the last process.

To work in the userland without touching the work of the users meant that I had to work in the libraries which provide the interfaces between kernels modules and users applications. Therefore I implemented a library which manages synchronization mechanisms.

3.2 Library implementation

From my library, I could implement the synchronization in the other libraries, which manage each devices. And then executables that are dynamically linked do not need to be compiled once again .

Moreover, library development provides interesting features. You can write 2 "methods", a constructor and a destructor, which are executed respectively when the library is loaded and unloaded. The destructor method is used to manage the destruction of the shared memories.

Basically, the destructor is called on the normal exit of a process. Here is performed everything to unlock mutexes and semaphores and free shared memories. This is executed only on a normal exit, it means that if the process ends abnormally (SIGSEGV, SIGTERM, ...) the destructor would not be called.

Most of the errors (core dump, segmentation faults) corresponds to a signal, I trapped the most well-known signals and I connected them to an exit call, therefore the process exits normally on most of the cases. The SIGKILL (9) which is not trappable, and some other signals that I decided to do not trap, will make an abnormal termination, and then the users would have to clean the system themselves.

3.3 How to use

3.3.1 Provided tools

I provided two tools to use to look at the system and to clean it (be careful).

The first tool is named lsshm, as its name says, it lists the shared memories. The shell command *s*"ipcs-m" performs the same function. However, on the CCPC, using this command results in a kernel panic. Hence this tool had to be written. Basically it displays the file /proc/sysvipc/shm.

The second tool is cleanshm. To call "ipcrm -m" on each shared memories was a lost of time when I was programming, hence this tool had to be made. It deletes every shared memories that are linked to 0 process.

3.3.2 *Use the library by yourself*

If you do not find the library enough efficient for your work, you can have a look at the sources documentation or at the header file, and call by yourself the functions get, lock, unlock and destroy on the IPC mechanisms identifiers. This will ensure that everything that is between a lock and an unlock will be executed safely. However, think about the other users.

You can also use them as process shared mutexes with any other identifier, for any other application than working on the glue card.

4 Watchdog timer

4.1 Watchdog daemon - NFS status

The Linux Kernel provides a module to manage the watchdog. This module works in pair with a daemon wich pings it at least once every 30 seconds. At LHCb, the watchdog is used to reboot the CCPC if a problem happens on it.

If the problem is that the nfs server is down, or unreachable, the CCPC loses its file system, but the CCPC will never reboot because the daemon works in RAM and it does not use the file system. So the daemon has to be aware of the nfs server status. If the daemon tries to access the plain file system, it would be stuck during the system call so it would not reset the watchdog anymore and the CCPC would reboot. However we want to wait 15 minutes before rebooting the CCPC in case the NFS server could be fixed.

The daemon would be stuck because the filesystem is mounted with the option "hard". This option makes the process retry indefinetly. The solution is therefore to have a file mounted with the option "soft" in the "hard" file system. This option makes the filesystem to return major timeout. This error can be handled, so it is done changing the state of the daemon. From the state in which the daemon pings the watchdog and checks the nfs status forevever, to a similar state limited in time for some minutes, defined as an argument.

So if the nfs server becomes up during this time, the daemon comes back to its orginal state, else it reboots the CCPC.

4.2 Watchdog daemon - File system integrity

The root filesystem should be read only mounted. It can happen that an administrator makes a mistake and mounts it writable. If so, each CCPC would write in a shared file system and it could be a source of conflicts. As this should not happen, the daemon checks the rights on the file system by creating a file, if it could create it, the CCPC is rebooted immediately.

4.3 Boot time

The CCPC watchdog timer maximum value is 30 seconds. It is really a short time, and if the CCPC needs more to download and uncompress the network boot image, the boot can be interrupted unexpectedly too early, before the first ping of the watchdog during the kernel module initialization. The only thing we can do is to reset the watchdog timer after the kernel has been downloaded and before it is uncompressed.

This is "managed" by pinging the watchdog at boot time, in the file setup.S [LBC]. Basically, this file contains the source code which initializes the computer hardware devices and configures the environement for the kernel to run. The ping should be possible because at this point of the boot we are still in real memory mode.

My source code is executed each boot, but it doesn't seems to reset the watchdog yet.

5 Conclusion

Most of the problems on the Creditcard PC are now solved or on their way to be solved. Process synchronization has been released on LHCb CCPC servers and seems efficient. The CCPC is aware of the NFS server and can restart in case of problems. The boot implementation is on a good way.

6 References

[AUP] Marc J. ROCHKIND, Advanced UNIX Programming, Second Edition, Addison Wesley Professional Computing Series

[COSD] Niko NEUFELD, Creditcard PC Online Software Documentation , http://lhcb-daq.cern.ch/lhcb-daq/ccpc/doc/index.html

[CSG] Niko NEUFELD, Creditcard PC softwareguide, CERN Internal Note

[LBC] Feiyun Wang, Linux i386 Boot Code HOWTO, http://tldp.org/HOWTO/Linux-i386-Boot-Code-HOWTO/index.html