MASTER EN SCIENCES INFORMATIQUES

Implémentation d'un système de courrier électronique sur UNIX

Radlet, Hugo

Award date:
1981

Awarding institution:
Universite de Namur

FACULTES UNIVERSITAIRES N. D. DE LA PAIX A NAMUR

INSTITUT D'INFORMATIQUE

Implémentation d'un système de courrier électronique sur UNIX

Mémoire présenté par
Hugo RADLET
pour l'obtention du grade
de Licencié et Maître
en Sciences Informatiques

Année Académique
1980 - 1981

Je tiens à exprimer mes remerciements à ceux qui m'ont aidé dans l'élaboration de ce mémoire :

* *A Ph. Van Batselaer qui a accepté de le diriger et dont les suggestions ont toujours été utiles.*
* *A J.P. Adans qui m'a aidé lors de l'implémentation du projet.*
* *A mon entourage qui m'a supporté dans les moments de découragement.*

Table des Matières. page

Introduction générale.

Ce mémoire est divisé en trois parties :

* Présentation générale du courrier électronique.
* Description du courrier électronique proposé.
* Implémentation sur UNIX.

La première partie décrit le concept initial du système d'échange de messages et les extensions de ce concept qui ont permis d'arriver à l'idée de courrier électronique. Cette partie se termine sur une critique du courrier électronique par rapport aux autres moyens de communication de messages.

La deuxième partie consiste en une description fonctionnelle du courrier électronique proposé dans le mémoire. Cette partie comprend quatre chapîtres. Le premier présente les restrictions du courrier électronique proposé par rapport à celui décrit en première partie. Le deuxième chapître décrit les concepts manipulés, tandis que le troisième décrit les fonctions maniant ces concepts. Le dernier chapître consiste en un exposé de différentes améliorations fonctionnelles réalisables à brève ou longue échéance.

La dernière partie explique comment ont été implémentées les fonctions décrites dans la deuxième partie, que ce soit du point de vue de l'organisation des informations ou du point de vue du traitement des informations. Les deux derniers chapîtres de cette partie ont trait au choix du langage de programmation et à la méthode utilisée pour tester le logiciel.

INTRODUCTION : LA BUREAUTIQUE.

Notre société est entrée dans l'ère de l'information. Une part de plus en plus grande de la population active travaille dans ce secteur. Aux Etats-Unis, cinquante pourcents de la population active est constituée de travailleurs de l'information (personnes qui utilisent l'information pour matière première et dont le résultat du travail est aussi de l'information). Dans le cadre de leur travail, ils passent cinquante à nonante pourcents de leur temps à communiquer entre eux.

Mais alors qu'une part de plus en plus grande de personnes travaillent dans un bureau - l'atelier du travailleur de l'information - paradoxalement, ce domaine reste sous-équipé par rapport aux autres secteurs d'activité. Aux Etat-Unis, en 1976, une personne travaillant dans un bureau, avait à sa disposition en moyenne 1300 dollars d'équipement alors qu'un ouvrier avait accès à 31000 dollars d'outils, soit plus de 13 fois plus ([1]).

Conséquemment, la croissance de la productivité moyenne d'un employé de bureau américain au cours de ces dix dernières années n'a été que de six pourcents alors que celle d'un ouvrier a atteint nonante-six pourcents.

Pourtant les technologies de traitement de l'information ont fait des progrès énormes ces dernières années. Des ressources telles que la mémoire ou le calcul sont devenues peu coûteuses et on peut encore s'attendre à de fortes diminutions pour la prochaine décade : baisses d'un facteur 20 pour le calcul et d'un facteur 100 pour la mémoire ([2]).

Jusqu'il y a peu, ces progrès ont surtout été exploités dans le traitement de l'information chiffrée grâce au développement de l'informatique de gestion. Or les chiffres ne sont pas les seules informations circulant dans un bureau. Il y a également le texte, la voix et l'image. Et ces trois types d'informations tiennent une place très importante dans le

volume global d'informations manipulées dans un bureau.

Donc pour rentabiliser le travail de ces personnes dont la majorité du temps est consacrée à communiquer avec les travailleurs de leur entreprise ou d'autres entreprises, il faut fournir des outils de gestion des informations non chiffrées. Tel est le projet de la bureautique. Voici par ailleurs une définition de la bureautique proposée par L. NAUGES ([1]) :

> *"La bureautique est concernée par l'ensemble des techniques de saisie, mémorisation, traitement, transport et diffusion de l'information et leur emploi pour la gestion des messages formels dans les organisations."*

Le courrier électronique est un des outils développés dans le cadre de la bureautique pour rentabiliser le travail de bureau. Cet outil est défini ci-après.

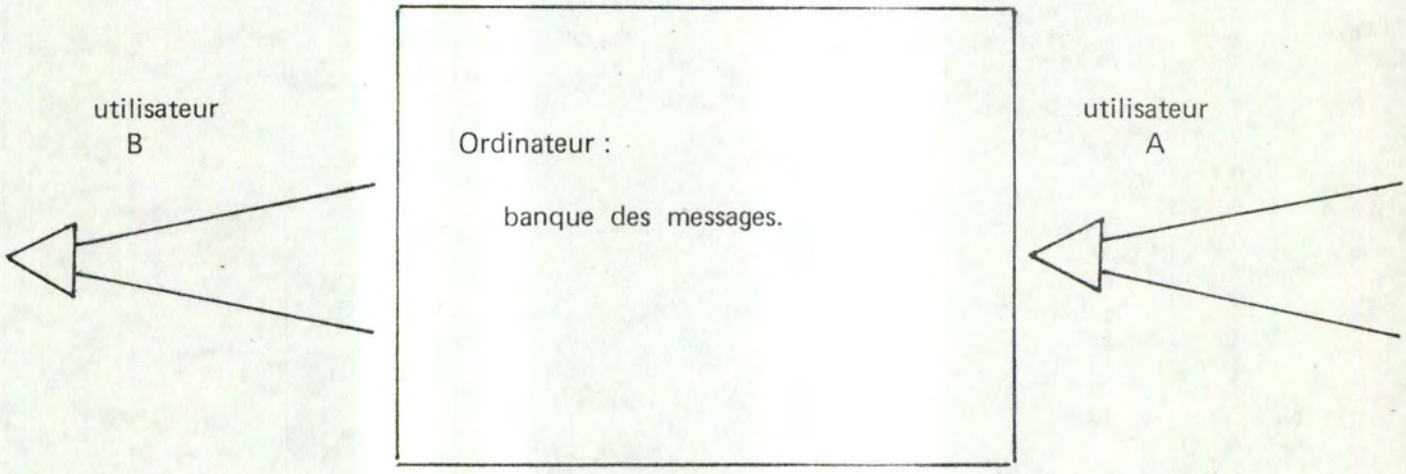
Ière Partie. Présentation générale du courrier électronique.

Dans le cadre de la bureautique, le courrier électronique est un outil tentant d'améliorer la communication des messages entre personnes travaillant dans un même site ou dans des sites différents.

Tentons de définir exactement le concept de courrier électronique.

1. Concept initial.

Voici schématiquement le concept initial d'un système de courrier électronique ([2]) :



Quel est le fonctionnement de ce système ?

* L'utilisateur A se connecte à un ordinateur via un terminal éloigné.
* Il tape un texte qui sera mémorisé dans un ordinateur et donne des instructions pour que le message soit envoyé à l'utilisateur B.
* Dès que l'utilisateur A a donné l'ordre d'envoi du message, celui-ci est disponible pour l'utilisateur B ; si ce dernier est connecté à l'ordinateur, il reçoit une note indiquant qu'un message est en attente pour lui, sinon il recevra la note la prochaine fois qu'il se connectera à l'ordinateur.

Ce système existait bien avant l'apparition de la bureautique, mais personne n'avait encore saisi le service que pourrait rendre cet outil dans le cadre du travail de bureau. Nous allons voir quel est l'intérêt d'un tel système.

2. Propriété remarquable : "Asynchronisme".

Une propriété remarquable d'un système de courrier électronique est que c'est un moyen de communication asynchrone rapide.

Une communication est asynchrone si la présence simultanée de toutes les personnes concernées n'est pas nécessaire pour que la communication puisse avoir lieu. Dans le système vu plus haut, il n'était pas nécessaire que l'utilisateur B soit connecté à l'ordinateur pour que l'utilisateur A puisse lui envoyer un message. Le courrier électronique est un moyen de communication rapide si on le compare à un autre moyen de communication asynchrone : la lettre postale. Même par rapport à un moyen de communication synchrone – exigeant la présence simultanée des personnes concernées pour qu'il y ait communication – rapide tel que le téléphone, le courrier électronique supporte facilement la comparaison.

Nous verrons plus loin les avantages d'un système de communication asynchrone rapide.

3. Extension du concept initial.

Utilisé tel quel, le courrier électronique est peu pratique : il permet juste l'envoi et la réception d'un message. C'est pour cette raison que les concepteurs des systèmes actuels ont essayé d'ajouter au système initial des outils rendant le courrier électronique efficace. Les quatre grands domaines d'extension sont :

1. Facilité de préparation des messages (grâce aux éditeurs de texte).
2. Extension des possibilités de communication (grâce aux réseaux).
3. Mémorisation des messages reçus et recherche sélective sur les messages mémorisés.
4. Possibilités d'action lors de la réception d'un message.

3. 1. Préparation d'un message.

Dans le système initial vu plus haut, les possibilités de préparation du message sont pratiquement nulles : ce sont celles que permet le terminal sur lequel travaille l'émetteur de message. Si nous prenons l'exemple d'un terminal à écran, les possibilités d'édition se résument à détruire le dernier caractère ou la ligne courante. Cela se révèle insuffisant si le message devient quelque peu volumineux. Il est alors intéressant de réafficher à l'écran le texte tapé et de pouvoir le modifier. Il faut donc mettre à la disposition de l'utilisateur d'un courrier électronique un éditeur de texte.

3. 2. Possibilités étendues de communication : Réseaux.

Le système de courrier électronique perd de son efficacité si les utilisateurs doivent être reliés à un même ordinateur, car le nombre de ces utilisateurs sera limité par la taille de l'ordinateur. En tout cas, le nombre de personnes utilisant le même système de courrier électronique ne dépassera pas quelques centaines. Par contre, l'emploi

de réseaux d'ordinateurs fait tomber cette contrainte. On peut même penser à l'utilisation de réseaux internationaux pour que des individus éparpillés aux quatre coins du monde puissent communiquer entre eux.

Prenons l'exemple d'ARPANET (*) et d'un des systèmes de courrier électronique développé sur ce réseau ([2]) :

* l'émetteur prépare à son terminal un message et désigne une liste de destinataires.
* Le système de courrier électronique crée, pour chaque destinataire, un fichier contenant une copie du message ; ce fichier est placé dans un espace de mémoire temporaire de l'ordinateur auquel est relié l'émetteur du message. Le nom du fichier contient également :
    * le nom de "l'ordinateur- destinataire" où le message doit être transféré.
    * un nom de compte sur cet "ordinateur-destinataire", nom à qui envoyer le message.
* Chaque ordinateur relié à ARPANET contient des programmes qui vérifient périodiquement s'il n'y a pas en mémoire des fichiers de message à envoyer. Si tel est le cas, une connection s'établit entre "ordinateur-émetteur" et "ordinateur-destinataire" et le fichier est transféré de l'un vers l'autre après avoir vérifié que le destinataire du message existe réellement.
* Chaque utilisateur possède son propre fichier de messages, protégé de telle sorte que l'utilisateur doit donner son mot de passe pour lire le message.

(*) ARPANET : réseau d'ordinateurs développé par "U.S. Departement of Defense Advanced Research Projects Agency".

3. 3. Mémorisation des messages reçus et recherche sélective.

En général, un message a une portée plus grande que les quelques secondes ou quelques minutes nécessaires pour le lire. Si nous prenons l'exemple d'un message indiquant la date d'un rendez-vous, la portée du message reçu sera le temps compris entre l'instant où le message est lu et l'instant du rendez-vous. Durant ce laps de temps, il est intéressant de garder le message en mémoire afin que le destinataire n'oublie pas le rendez-vous. Certains messages sont longs et contiennent des informations intéressantes à garder en mémoire pour pouvoir les consulter plus tard. Il est donc nécessaire de pouvoir stocker certains messages reçus en mémoire.

Si le nombre de messages mémorisés devient important, il faut que la recherche des messages archivés puisse se faire sélectivement. Les critères de sélection seront, par exemple, la date d'envoi du message, le nom de l'émetteur, le numéro du message ou le sujet associé à un message.

3. 4. Possibilités d'action sur les messages reçus.

Souvent la réception d'un message implique une ou plusieurs actions de la part du destinataire. Dans certains cas, le système de courrier électronique peut aider l'utilisateur. Par exemple, le message reçu doit être envoyé vers d'autres utilisateurs ou exige une réponse de la part du receveur. Ou bien le message contient des informations importantes et le destinataire voudrait que le message lui soit rappelé chaque fois qu'il se connecte sur le système. Dans ces cas-là, le système de courrier électronique peut fournir des commandes facilitant la tâche de l'utilisateur.

4. Avantages du courrier électronique.

Le courrier électronique basé sur un réseau d'ordinateurs fournit une facilité de communication entre individus que ne permet aucun autre moyen de communication. Un tel système permet de décentraliser l'information et donne plus d'indépendance à l'individu.

La propriété asynchrone du courrier électronique fait de ce système un moyen de communication efficace, faisant gagner aux personnes l'utilisant beaucoup de temps. En effet, l'émetteur du message ne doit plus se soucier de la disponibilité du destinataire au moment de l'envoi. Si l'on se souvient du temps que l'employé de bureau passe à communiquer, on se rend compte que le courrier électronique permet à ce dernier de mieux gérer son temps de travail.

Prenons un exemple pour mettre en évidence ce dernier point : Un manager A a un problème pour lequel peut l'aider un manager B.

Premier scénario : Le moyen de communication utilisé est le téléphone.

A appelle B au téléphone, mais B est en conférence. A laisse un message à la secrétaire de B, demandant à ce dernier de le rappeler lorsqu'il le pourra. Après la conférence, B reçoit le message de sa secrétaire ; il téléphone à A, mais ce dernier n'est pas dans son bureau. Lorsque A revient dans son bureau, il essaye à nouveau de contacter B, mais celui-ci est maintenant en réunion. Ce petit jeu peut durer plusieurs jours. A et B ont perdu du temps à attendre d'être libres simultanément, car le téléphone est un moyen de communication synchrone.

Deuxième scénario : Le moyen de communication utilisé est le courrier électronique.

A tape le message au terminal et donne des instructions pour que le message

soit envoyé à B. Lorsque B est disponible, il lit le message à son terminal. Il tape sa réponse et donne des instructions pour qu'elle soit envoyée à A. A et B n'ont perdu aucun temps à attendre que l'autre soit disponible. Mais pour que le système soit vraiment efficace, deux conditions doivent être remplies :

1. : les managers A et B doivent savoir manipuler rapidement le clavier d'un terminal.
2. : les managers A et B doivent avoir un terminal dans leur bureau ou sinon le temps passé à attendre que l'autre soit disponible risquerait d'être remplacé par le temps passé à attendre qu'un terminal soit libre.

Un autre avantage du courrier électronique est qu'une même copie d'un message peut être envoyé à plusieurs personnes. Le téléphone ne donne pas cette facilité. Imaginons qu'un message doive être envoyé à dix personnes. Par le courrier électronique, l'émetteur tape une fois le message et donne des instructions pour que le message soit envoyé aux dix personnes concernées. Par téléphone, il faudra donner dix coups de téléphone – avec les pertes de temps que cela engendre (voir ci-avant) – et répéter dix fois le même discours, action fastidieuse s'il en est.

Le courrier électronique permet également de faire circuler facilement un message à travers les individus d'une organisation. Cela peut être intéressant dans une entreprise où des messages circulent depuis le plus haut niveau jusqu'au bas de la hiérarchie. Un système de courrier électronique permet de faire circuler un message - sans devoir le recopier - à travers tous les niveaux hiérarchiques d'une entreprise.

Malgré tous les avantages cités plus haut, le courrier électronique ne remplacera pas complètement les autres moyens de communication tels le téléphone, la lettre postale

ou la conversation face à face ; c'est un autre outil qui a également ses désavantages. Une lecture trop rapide ou inattentive d'un message reçu par courrier électronique peut mener à une réponse ne cadrant pas avec la question posée. Cela peut facilement être résolu par un coup de téléphone ou une conversation face à face. En fait, ce qui serait intéressant dans un système de courrier électronique, c'est de pouvoir associer la voix à un message, car la voix donne un ton, une orientation à la conversation, ce que ne donne pas l'écriture. Des recherches dans ce sens se font actuellement.

Un dernier obstacle à l'utilisation du courrier électronique réside dans la difficulté d'utilisation d'un tel système par des non-initiés à l'informatique. Ou bien le système doit être évident à utiliser, ou bien il doit exister une documentation précise quant au mode d'emploi du système. L'idéal serait qu'un utilisateur non-initié utilise un système de courrier électronique avec la même facilité avec laquelle il utilise le téléphone.

Enfin, si un système de courrier électronique possédant toutes les qualités citées plus haut existe, il doit être, en plus, fiable. Si les travailleurs d'une société se reposent sur un système de courrier électronique pour communiquer à l'intérieur et à l'extérieur de leur entreprise et si le système tombe en panne, c'est l'activité de toute l'organisation qui est arrêtée.

IIème Partie. Description du courrier électronique proposé.

1. Introduction.

Nous venons de voir dans la première partie de ce mémoire une présentation générale des propriétés que peut posséder un système de courrier électronique.

Le but de ce mémoire était de réaliser un courrier électronique sur un équipement existant. Il n'était pas possible de construire un système possédant toutes les extensions décrites plus haut.

Une première contrainte était matérielle. La réalisation du courrier s'est faite sur un ordinateur PDP 11/45 équipé du système d'exploitation UNIX ([9]), non relié à un réseau d'ordinateurs. Donc l'optique "Possibilités étendues de communications" a été laissée de côté, car elle n'aurait pu être implémentée.

Une deuxième contrainte était une contrainte de temps. Il n'était pas possible de développer et mettre au point de façon égale toutes les extensions du courrier électronique. Ont été réalisées en priorité les fonctions qui semblaient les plus importantes dans un tel système. Ensuite ont été mises au point les fonctions permettant une utilisation plus agréable des fonctions de base.

Il faut donc considérer le courrier électronique décrit ci-après comme un noyau auquel il est possible d'ajouter de nouvelles fonctions, afin de rendre le système plus efficace.

2. Description des concepts manipulés.

2.1. Utilisateur individuel.

Tout utilisateur de UNIX peut devenir membre du courrier électronique. Il doit avoir un nom l'identifiant dans le système de courrier électronique. Ce nom est le nom de login (login name) sur le système UNIX ([9]). Donc, lorsqu'un membre du courrier électronique envoie un message à un autre utilisateur du système, c'est le nom de login de ce dernier qu'il doit spécifier comme nom de récepteur.

2.2. Répertoire des utilisateurs.

L'ensemble des utilisateurs individuels est recensé dans un répertoire des utilisateurs. Une entrée de ce répertoire contient les informations suivantes :

* nom de login sur UNIX de l'utilisateur.
* patronyme de l'utilisateur.
* mot de passe encrypté de l'utilisateur dans le courrier.

Les entrées de ce répertoire sont triées par ordre alphabétique sur le nom de login.

2.3. Liste de destinataires.

Une liste de destinataires est constituée de un ou plusieurs noms de login d'utilisateurs du courrier électronique. Tout membre du courrier électronique peut créer une telle liste. Pour ce faire, il doit donner un nom à la liste, ainsi que le nom de login des personnes qu'il désire insérer dans cette liste. Par après, le créateur de la liste pourra invoquer le nom de cette liste comme nom de récepteur lors de l'envoi d'un message.

Prenons un petit exemple. Pierre est membre du courrier. Il envoie souvent des messages à Jean, Jacques, Paul, André et Thomas qui sont également membres du courrier. Pierre crée une liste DISCIPLE reprenant les cinq noms cités plus haut. Lorsque Pierre enverra une lettre aux cinq disciples, il donnera comme nom de récepteur DISCIPLE. Jean, Jacques, Paul, André et Thomas recevront chacun une copie du message.

La liste des destinataires est donc une facilité fournie à l'utilisateur lorsque celui-ci envoie souvent des messages aux mêmes personnes ; La liste évite de répéter chaque fois tous les noms de récepteur.

2.4. Message.

Un message est divisé en deux parties :

* en-tête : contient des informations telles le nom de l'émetteur, la date d'envoi du message, etc.
* corps.

2.4.1. En-tête.

Quelles sont les informations contenues dans l'en-tête ?

Elles sont de deux types :

a) informations fournies par le système :

* numéro de message.

  chaque message est identifié par un numéro ; ce numéro est un nombre qui est incrémenté de une unité à chaque message envoyé.

* nom de login de l'émetteur.
* date et heure d'envoi du message.

b) informations données par l'émetteur.

* nom du (des) destinataire (s) :

  nous avons vu que pouvait être utilisé comme nom de récepteur soit un nom de login, soit un nom de liste de destinataires (2.1 et 2.3). A un message peuvent être associés un ou plusieurs récepteurs. Nous retrouverons donc dans le répertoire des récepteurs d'un message une combinaison de zéro, un ou plusieurs noms de login d'utilisateurs individuels et de zéro, un ou plusieurs noms de liste de destinataires.

* nom du (des) "Carbon Copy(ies)"

  Un "carbon copy" est une personne à qui le message n'est pas adressé directement, mais qui en reçoit une copie pour information. La liste des "carbon copies" est une combinaison de zéro, un ou plusieurs noms de login et noms de liste de destinataires.

* Indicateur :

  — le message est urgent.

  — le message est important.

  — le message n'est ni urgent, ni important.

* Le message est-il secret ou non ?

  Si l'émetteur indique le message comme étant secret, le récepteur devra donner son mot de passe dans le courrier pour pouvoir lire le message.

* Sujet du message.

  L'émetteur y met ce qu'il veut, rien à la rigueur, mais en principe

cette rubrique permet au récepteur d'être renseigné sur le contenu du message.

2.4.2. Corps.

Le corps du message contient le texte qu'a préparé (tapé édité) l'émetteur du message. Aucune limite de taille n'est fixée à ce texte ; il peut contenir une seule ligne aussi bien qu'un rapport de plusieurs pages.

2.5. Confirmation.

Une confirmation est envoyée à l'émetteur d'un message lorsque le récepteur a lu le message. Cette confirmation est complètement fabriquée par le système de courrier électronique.

Les renseignements contenus dans une confirmation sont les suivants :

* numéro du message.
* date et heure d'envoi.
* nom du récepteur.

3. Description des fonctions.

Les fonctions du courrier électronique peuvent être divisées en trois niveaux :

Niveau 1 : Fonctions exécutables dès l'entrée dans le système de courrier électronique ; elles peuvent être réparties en deux blocs :

Bloc 1 : Fonctions principales : fonctions manipulant les concepts de "message" et "confirmation".

1. Annonce de message en attente et confirmation.
2. Envoi d'un message.
3. Lecture des messages en attente.
4. Lecture des messages archivés.

Bloc 2 : Fonctions annexes : fonctions manipulant les concepts autres que "message" et "confirmation".

1. Devenir utilisateur du courrier.
2. Se retirer du courrier.
3. Créer une liste d'utilisateurs.
4. Détruire une liste de destinataires.
5. Consulter le répertoire des utilisateurs.

Niveau 2 : Fonctions appelables à partir d'une fonction de niveau 1. Elles peuvent être réparties en deux blocs :

Bloc 1 : Fonctions appelables par les fonctions 3 et 4 du bloc 1, niveau 1.

1. Imprimer un message.

2. Détruire un message.

3. Imprimer + détruire un message.

Bloc 2 : Fonctions appelables par la fonction 3 du bloc 1, niveau 1.

1. Faire suivre un message.

2. Répondre à un message.

3. Garder un message en attente.

4. Archiver un message.

Niveau 3 : Fonctions appelables par les fonctions de niveau 1 et de niveau 2.

1. Sauver un message sur fichier personnel.

2. Exécution d'une commande du système d'exploitation.

3.1. Niveau 1.

3.1.1. Fonctions principales.

3.1.1.1. Annonce des messages en attente et des confirmations.

Lorsque l'utilisateur tape son nom de login pour entrer dans UNIX ou lorsqu'il tape la commande permettant d'accéder au système de courrier électronique, il est informé s'il a des messages en attente ou des confirmations.

Pour chaque message en attente, les informations suivantes sont affichées à l'écran :

* numéro du message.

* nom de login de l'émetteur.

* message urgent ou important si tel est le cas.

* date et heure d'envoi. Ex. : feb 25 14 : 32

* sujet du message.

En ce qui concerne les confirmations, les informations affichées sont :

* numéro du message.

* nom de login du récepteur.

* date et heure d'envoi.

La fonction d'annonce des messages en attente et des confirmations est implicite ; elle ne nécessite aucune opération de l'utilisateur, Ce dernier ne peut donc faire appel à cette fonction lorsqu'il travaille dans le courrier.

3.1.1.2. Envoi d'un message.

L'opération d'envoi d'un message peut être divisée en deux parties :

* préparation du message.

* remplissage des formalités d'envoi.

Préparation du message.

Dans un premier temps, l'utilisateur tape le message au terminal sans autres possibilités d'édition que celles du terminal : destruction du dernier caractère tapé ou de la ligne courante. Lorsqu'il a terminé il peut éditer le message.

Pour ce faire, il dispose de l'éditeur ED de UNIX (I 9 I).

Voici brièvement les différentes commandes de cet éditeur :

* append : ajouter des lignes au texte à partir de la ligne spécifiée dans la commande.
* change : remplacer les lignes spécifiées dans la commande par un nouveau texte.
* delete : détruire les lignes spécifiées dans la commande.
* insert : insérer des lignes dans le texte au-dessus de la ligne spécifiée dans la commande.
* move : déplacer les lignes du texte spécifiées dans la commande en-dessous d'une ligne donnée.
* transfert : recopier les lignes du texte spécifiées dans la commande en-dessous d'une ligne donnée.
* print : afficher à l'écran les lignes spécifiées.
* read : lecture d'un fichier dans le texte à partir d'une ligne spécifiée dans la commande.
* write : écrire les lignes du texte spécifiées dans la commande dans un fichier.
* subtitute : substituer une chaîne de caractères par une autre dans une ou plusieurs lignes de texte.

L'inconvénient de cet éditeur est qu'il s'agit d'un éditeur de programmes et non d'un éditeur de texte, ce qui rend son utilisation peu pratique pour la préparation de messages. Malheureusement, il n'était pas possible – pour des raisons de temps – de

construire un éditeur de texte dans le cadre de ce mémoire, mais il serait intéressant de pouvoir en disposer pour le courrier électronique.

Après avoir préparé le message, l'utilisateur peut encore décider d'envoyer ou non le message. Dans l'affirmative, il doit remplir certaines formalités :

Remplissage des formalités d'envoi.

Nous avons vu qu'un message est constitué d'un corps et d'un en-tête (2.3). Le corps a été constitué lors de la préparation du message. Maintenant, l'émetteur doit donner les renseignements utiles pour remplir l'en-tête du message :

* le nom du (des) destinataire(s).

  Rappelons qu'il s'agit de la combinaison de :

  - zéro, un ou plusieurs noms de login d'utilisateurs.
  - zéro, un ou plusieurs noms de liste de destinataires.

  Si l'utilisateur ne met aucun nom, le message n'est pas envoyé ; il est donc encore possible à ce stade-ci de décider de ne pas envoyer le message.

* le nom du (des) "Carbon Copies".

  Les possibilités de remplissage de cette rubrique sont les mêmes que pour la rubrique précédente.

* le sujet du message.

* le message est-il urgent ou important ?

* le message est-il secret ?

En fin d'exécution de la fonction d'envoi – que l'utilisateur ait envoyé le message ou non – il est possible de faire appel à la fonction de sauvetage du corps de message dans un fichier de travail.

3.1.1.3. Lecture des messages en attente.

L'utilisateur peut demander la lecture des messages en attente. Il peut le faire en sélectionnant ce qu'il désire voir afficher. Les possibilités de sélection sont les suivantes

* les messages envoyés par un émetteur donné.
* les messages dont le sujet contient une chaîne de caractères donnés.
* les messages envoyés avant une date donnée.
* les messages envoyés après une date donnée.
* les messages dont l'utilisateur a donné le numéro.

  Les numéros de message peuvent être donnés sous forme :

  — de séquence.

  Ex. 101,97,81,103,

  — d'intervalle.

  Ex. : 101 : 109

  Les messages dont les numéros sont compris entre 101 et 109 (inclus) seront affichés.

* les messages urgents.
* les messages importants.
* tous les messages. (option par défaut).

Le système de courrier électronique recherche les messages vérifiant la clé de sélection. Pour chaque message répondant à la condition, deux solutions peuvent se présenter :

— le message est secret :

l'utilisateur doit donner son mot de passe pour lire le message ; s'il n'y arrive pas, le courrier passe à l'affichage du message suivant. Sinon, la suite se déroule comme si le message n'était pas secret.

-- le message n'est pas secret :

l'en-tête et le corps du message sont affichés à l'écran. Voici un petit exemple de message tel qu'il apparaît à l'écran :

34 From : JD Febr 25 14 : 32 Exemple.

Chers lecteurs,

Voici un exemple de message tel qu'il apparaît à l'utilisateur lorsqu'il utilise la fonction de lecture de messages en attente.

Jules

Lorsque le message est affiché à l'écran, les fonctions de niveau 2 et de niveau 3 peuvent être utilisées.

3.1.1.4 Lecture des messages archivés.

La fonction de lecture des messages archivés fonctionne de la même façon que la fonction de lecture des messages en attente. Les clés de sélection sont identiques dans les deux cas. C'est dans la liste des fonctions appelables à partir de la fonction principale qu'il y a une différence : seules les fonctions de niveau 2, bloc 1 et de niveau 3 sont appelables lorsque l'utilisateur emploie la fonction

de lecture des messages archivés.

3.1.2. Fonctions annexes.

3.1.2.1 Devenir membre du courrier.

Devenir membre du courrier est une fonction comme une autre. Un utilisateur de UNIX qui veut devenir membre du courrier électronique n'a qu'à rentrer dans le système de courrier et utiliser la commande permettant d'être membre du courrier électronique.

Deux renseignements sont demandés au nouvel utilisateur :

* son patronyme
* le mot de passe qu'il désire utiliser dans le courrier.

  Ce mot de passe servira lorsque l'utilisateur recevra un message secret. Dans ce cas, il devra donner son mot de passe pour lire le message.

Les renseignements ainsi obtenus et le nom du login de l'utilisateur sont introduits dans le répertoire des utilisateurs.

3.1.2.2. Se retirer du courrier.

Pour se retirer du courrier, il suffit de taper la commande correspondant à la fonction. Aucun renseignement n'est demandé à l'utilisateur. Si un membre du courrier utilise cette fonction, l'entrée contenant les informations le concernant est supprimée du répertoire des utilisateurs, ainsi que les messages en attente et les messages archivés qui lui appartenaient.

3.1.2.3. Créer une liste de destinataires.

Il existe une fonction permettant de créer une liste de destinataires. Dans la description des concepts manipulés est expliqué ce qu'est une liste de destinataires et quelle est son utilité. Tout utilisateur peut employer cette fonction. Il devra donner le nom de login des utilisateurs qu'il désire insérer dans la liste ainsi que le nom de la liste, nom qu'il pourra invoquer comme nom de récepteur lors de l'envoi d'un message.

3.1.2.4. Détruire une liste de destinataires.

La fonction inverse à la création d'une liste de destinataires est naturellement la destruction d'une liste de destinataires. Le seul renseignement que doit donner l'utilisateur de la fonction est le nom de la liste à détruire. Un défaut de cette fonction est qu'il n'est pas nécessaire d'être créateur d'une liste pour pouvoir la détruire. N'importe quel utilisateur du système peut détruire n'importe quelle liste.

3.1.2.5. Consulter le répertoire des utilisateurs.

La consultation du répertoire des utilisateurs peut se faire sélectivement ou non. Une première possibilité est d'afficher le répertoire complet des membres du courrier. Une autre possibilité est d'afficher une partie seulement du répertoire en prenant comme critère de sélection soit le nom de login, soit le patronyme. Il n'est pas nécessaire de donner l'entièreté du nom comme clé de sélection.

Si l'utilisateur donne les n premières lettres de la clé, il recevra en retour la liste de toutes les entrées dont la clé commence par les n premières lettres données.

3.2. Niveau 2.

Une ou plusieurs fonctions de niveau 2 peuvent être appliquées sur un même message sous réserve de compatibilité.

3.2.1. Bloc 1.

3.2.1.1. Imprimer un message.

Il est intéressant de garder une trace écrite de certains messages contenant des informations importantes. C'est pour cette raison qu'est fournie à l'utilisateur la possibilité d'imprimer le corps du message courant c'est-à-dire celui qui est affiché à l'écran au moment où la commande d'impression est demandée.

3.2.1.2. Destruction d'un message.

Cette fonction permet de détruire le message courant lorsque ce dernier a perdu tout intérêt. Une fois le message détruit, l'utilisateur en perd la trace définitivement : il n'a plus aucun moyen de récupérer le message.

3.2.1.3. Impression destruction d'un message.

Cette fonction est une combinaison des deux précédentes. Elle permet de garder une trace écrite de l'information avant de

détruire le message.

3.2.2. Bloc 2.

3.2.2.1. Faire suivre un message.

Un utilisateur du courrier électronique venant de lire un message en attente peut faire suivre le message vers d'autres membres du courrier électronique. Ces derniers recevront le même message avec une indication supplémentaire : le nom de login de celui qui a fait suivre. Les noms repris dans la liste des nouveaux récepteurs du message peuvent être des noms de login ou des noms de liste de destinataires.

3.2.2.2. Répondre à un message.

Cette fonction est une facilité accordée au lecteur d'un message et qui désire y répondre ; il ne doit plus remplir de formalités d'envoi, contrairement au cas de la fonction d'envoi d'un message :

* le récepteur du message est l'émetteur du message que vient de lire l'utilisateur.
* le sujet du message indique qu'il s'agit d'une réponse :

  Ex : In Reply to message 107 (27 Feb 17 : 10)
* si le message lu était urgent ou important, il en sera de même pour la réponse.
* si le message était secret, il en sera de même pour le réponse.

L'utilisateur fabrique le corps de la réponse de la même façon que dans la fonction d'envoi d'un message. Il peut également appe-

ler la fonction de sauvetage d'un message sur fichier.

3.2.2.3. Garder un message en attente.

Il peut être intéressant de garder un message en attente -- par par exemple, un message fixant un rendez-vous -- même s'il a déjà été lu. Cela permet de se rappeler de sa présence à chaque entrée dans le système UNIX ou dans le courrier électronique. C'est ce que permet cette fonction.

3.2.2.4. Archivage d'un message.

Cette fonction d'archivage est appliquée par défaut sur un nouveau message venant d'être lu. Si le récepteur d'un message ne demande pas explicitement de le détruire ou de le garder en attente, le message est transféré dans un fichier d'archive. Ce fichier, l'utilisateur pourra le consulter comme il a été expliqué au paragraphe 3.1.1.4.

3.3. Niveau 3.

3.3.1. Sauver un message sur fichier personnel.

Dans certains cas, il peut être utile de sauver un corps de message sur fichier personnel. Ce fichier se trouvera dans le répertoire de l'utilisateur, et non dans celui du courrier électronique. L'utilisateur de cette fonction doit donner le nom qu'il désire pour le fichier. Ce nouveau fichier est sous l'entière responsabilité de celui qui l'a créé ; il n'existe pas pour le courrier électronique.

3.3.2. Exécuter une commande du système d'exploitation.

Cette fonction n'est pas tout à fait à sa place au niveau 3, car si elle est effectivement appelable à partir d'une fonction de niveau 1 ou niveau 2, elle est également appelable dès l'entrée dans le courrier. Cette fonction ne permet d'exécuter qu'une seule commande du système d'exploitation par appel de la fonction. Si l'utilisateur désire exécuter deux commandes du système d'exploitation, il doit appeler deux fois la fonction.

4. Améliorations fonctionnelles.

Le courrier électronique proposé ci-avant est un noyau auquel il est possible d'ajouter de nouvelles fonctions. Je vais essayer ici de donner quelques idées susceptibles d'améliorer les services que rend le courrier électronique que je viens de décrire.

Une première amélioration — et sans doute la plus urgente — serait d'ajouter une commande "HELP". Cette commande aurait pour effet de donner la liste des commandes possibles, ainsi que la liste des attributs pour chaque commande au moment où la fonction est appelée. En effet, il arrive souvent lorsqu'on utilise un tel système de ne plus savoir quel type de commande on peut utiliser et de commettre ainsi des erreurs irréparables. Pour cette raison, une commande "HELP" est nécessaire.

Dans la description des fonctions, nous avons vu qu'il existait une fonction d'annonce des messages en attente et des confirmations, mais que cette commande était implicite . Il serait agréable que cette fonction puisse être appelée explicitement lorsqu'on travaille dans le courrier, afin de savoir si de nouveaux messages ou de nouvelles confirmations ne sont pas arrivés depuis l'entrée dans le système de courrier électronique.

Le système de courrier permet de créer des listes de destinataires, mais aucune fonction ne permet de consulter la liste des listes de destinataires ni le répertoire des membres d'une liste dont l'utilisateur aurait donné le nom. Ces deux fonctions sont nécessaires car un utilisateur peut avoir oublié quelles listes il a créé ou quels sont les membres appartenant à une liste.

Il est possible également d'améliorer la fonction d'envoi de messages. Une première amélioration pourrait être de créer un éditeur de textes en place de l'éditeur de programmes du système UNIX, car ce dernier est peu pratique pour la préparation de messages. Avec l'éditeur actuel, des opérations telles qu'ajouter une chaîne de caractères dans une ligne ou ajouter des lignes au texte exigent parfois un remaniement du texte. par l'opérateur. Il faudrait que cela soit fait automatiquement par l'éditeur.

Un autre ennui de l'éditeur actuel est que son utilisation n'est pas aisée du fait d'une formalisation peu explicite. Voici un exemple de commande sur l'éditeur actuel : /party/--1 S/good/bad/. Sa signification est : chercher la ligne contenant la chaîne de caractères "party", passer à la ligne précédente ; substituer la chaîne de caractères "good" par "bad".

Donc, l'idéal serait un éditeur très facile d'emploi et conçu pour la mise en page de texte.

Une deuxième amélioration au niveau de la fonction d'envoi de messages consisterait à créer dans le système de courrier des messages ou des morceaux de messages standards que l'utilisateur pourrait appeler lorsqu'il prépare un message, Ceci est très pratique pour l'en-tête d'un message ou pour la formule de politesse par exemple.

A plus long terme, il serait intéressant d'appliquer l'idée de réseau développée dans la première partie de ce mémoire. Une première étape serait d'utiliser les ordinateurs

PDP II de l'institut d'informatique et DEC 2060 du centre de calcul pour créer un mini réseau. Les utilisateurs de ces deux ordinateurs pourraient s'échanger des messages grâce au courrier électronique qui serait implémenté sur les deux machines. Une deuxième étape serait de créer un réseau pour toutes les Facultés de Namur et d'implémenter le courrier électronique sur ce réseau, pour permettre l'échange de messages à travers toutes les Facultés.

IIIème Partie : Implémentation sur UNIX.

1. Descriptions des informations.

1.1. Organisation générale des données.

Le but de ce chapître est d'expliquer comment a été organisé l'ensemble des informations manipulées dans le courrier électronique, que ces informations soient propres à chaque utilisateur (messages, confirmations) ou destinées au bon fonctionnement du système.

1.1.1. Les fichiers sur UNIX.

Tout d'abord, voyons brièvement comment sont organisés les fichiers sur UNIX ([10]) :

Les fichiers sont organisés de façon hiérarchique en directoire et peuvent être munis de protection d'accès . Les fichiers se subdivisent en :

* directoires qui fournissent des chemins d'accès aux fichiers qui y figurent.
* fichiers spéciaux représentant les périphériques (ils ne nous intéressent pas).
* fichiers au sens classique du terme.

Les directoires sont organisés sous forme d'un arbre dont la racine est appelée "ROOT". Deux fichiers portant le même nom ne sont identiques que s'ils se trouvent dans le même répertoire ou si un lien a été établi entr'eux.

Du point de vue de UNIX, un fichier est considéré comme un paquet de bytes ; aucune organisation classique de fichier (séquentiel, indexé, direct,...) n'est disponible. L'utilisateur doit organiser ses fichiers lui-même. Le système d'exploitation fournit à l'utilisateur des primitives de création, ouverture, écriture, lecture, parcours, destruction des fichiers.

1.1.2. Répartition des informations.

Dès le départ, deux solutions se présentaient quant à la répartition des informations propres aux utilisateurs :

1. créer dans le répertoire de chaque membre du courrier un sous-répertoire contenant les fichiers du courrier électronique lui appartenant.
2. créer un répertoire centralisé contenant les fichiers de tous les membres du courrier électronique.

La première solution présente l'avantage de pouvoir facilement donner des noms aux fichiers de messages. En effet, les noms ne doivent pas être uniques, car dans le système d'exploitation UNIX, deux fichiers portant le même nom ne sont identiques que s'ils appartiennent au même répertoire. Ceci est intéressant car il existe dans le système de courrier électronique beaucoup de fichiers contenant le même type d'information, mais qui doivent néanmoins être uniques : par exemple, chaque utilisateur possèdera deux fichiers de messages (en attente et archivés). Par contre le désavantage de la première solution est de placer dans le répertoire de l'utilisateur des fichiers qui théoriquement lui sont transparents. Si les fichiers

ne sont pas protégés, l'utilisateur peut détruire ou modifier les fichiers du courrier, ce qui est très ennuyeux pour le bon fonctionnement du système. Si les fichiers sont protégés contre les manipulations de son propriétaire théorique, le problème cité plus haut disparaît, mais il reste qu'il n'est pas élégant d'encombrer le répertoire d'un utilisateur d'informations auxquelles il ne peut accéder directement. De toute façon, l'utilisateur doit rester maître de son répertoire.

La deuxième solution présente la situation inverse : tous les fichiers étant regroupés dans un même répertoire, ils doivent avoir un nom unique pour être uniques. Ceci pose un problème de dénomination de fichiers. Par contre, tous les fichiers sont transparents à l'utilisateur. C'est pour cette raison que la deuxième solution a été choisie pour le rangement des informations manipulées dans le courrier électronique.

Ceci étant posé, voyons maintenant quels sont les fichiers manipulés dans le courrier électronique :

1.2. Types de fichiers manipulés.

Les fichiers du courrier électronique peuvent être divisés en deux catégories :

* les fichiers propres à chaque utilisateur, comme par exemple les fichiers de messages.
* les fichiers nécessaires au fonctionnement du système.

1.2.1. Fichiers propres à l'utilisateur.

1.2.1.1. Fichiers de messages.

Nous avons vu dans la description du courrier électronique qu'un message était composé d'un en-tête et d'un corps (II ème Partie : 2.4.). Du point de vue de l'implémentation, en-tête et corps seront mémorisés dans des fichiers différents. De plus, un fichier d'index sera créé pour accélérer la recherche des messages dans le cas où la clé de recherche est soit le numéro de message, soit la date d'envoi. Il aurait été intéressant pour des raisons de rapidité de recherche d'implémenter pour chaque clé de recherche (IIème Partie : 3.1.1.3.) une liste inversée sur la clé. L'inconvénient de ce système était qu'il était lourd à mettre en oeuvre et surtout à mettre à jour (surtout lors de la destruction des messages).

Décrivons maintenant les trois types de fichiers de messages :

* fichier "BODY"

  Un fichier du type "BODY" contient les corps de message. Par utilisateur, il y a un seul fichier "BODY". Les corps de messages en attente et archivés sont donc rassemblés dans un seul fichier. L'avantage d'un tel système est que lorsqu'un message en attente est transféré en archive, il n'y a physiquement aucune opération à réaliser.

  Un corps de message est de longueur variable. Un fichier de type "BODY" est divisé en blocs de longueur fixe. Lorsqu'il

faut mémoriser un corps de message dans un tel fichier, la longueur du message est divisée par la taille d'un bloc ; la valeur ainsi obtenue est augmentée d'une unité pour obtenir le nombre de blocs à réserver.

Comment est organisé un fichier "BODY" ?

A chaque fichier "BODY" est associée une table des blocs libres mémorisée dans un fichier de type "FBODY". Chaque bit de cette table correspond à un bloc du fichier "BODY". Si le bit est à zéro, le bloc est libre. Si le bit est à 1, le bloc est occupé.

Lorsqu'un corps de message est inséré dans un fichier "BODY" la table "FBODY" correspondante est balayée jusqu'à ce que soit trouvé le nombre de bits à 0 consécutifs égal au nombre de blocs nécessaires pour mémoriser le message. La position du premier des bits à 0 donne la position du premier bloc libre dans le fichier "BODY". La destruction d'un corps de message revient à mettre à 0 les bits de la table correspondant à la position des blocs contenant le corps de message à détruire.

* fichier "HEAD".

Un fichier de type "HEAD" contient les en-têtes de message. Un utilisateur possède un tel fichier pour les messages en attente et un autre pour les messages archivés. L'article d'un fichier "HEAD" est de longueur fixe. En plus des informations décrites dans le paragraphe 2.4.1 de la Ième Partie, l'article d'un fichier

"HEAD" contient un pointeur vers le corps de message correspondant qui se trouve dans un fichier "BODY". Ce pointeur est constitué de la position du bloc contenant le début du corps par rapport au début du fichier.

A chaque fichier "HEAD" est associé une table des blocs libres mémorisée dans un fichier de type "FHEAD". Chaque bit de cette table correspond à un bloc du fichier "HEAD". Si le bit est à 0, le bloc est libre. Si le bit est à 1, le bloc est occupé. Lorsqu'un nouvel en-tête est inséré, la table des blocs libres "FHEAD" est balayée jusqu'à ce qu'un bit à 0 soit trouvé. La destruction d'un en-tête de message revient à remettre à 0 le bit de la table correspondant à la position du bloc contenant l'en-tête à détruire.

* fichier "INDEX".

Un utilisateur possède un tel fichier pour les messages en attente et un autre pour les messages archivés. Les informations reprises dans l'article d'un fichier "INDEX" sont :

* numéro du message.
* date d'envoi du message sous forme AAMMJJ.
* pointeur vers l'en-tête correspondant dans le fichier "HEAD".

Le pointeur est constitué de la position du bloc contenant l'en-tête correspondant par rapport au début du fichier.

Les articles d'un fichier "INDEX" sont triés par ordre croissant sur le numéro de message.

1.2.1.2. Fichier "CONF".

Un fichier de type "CONF" contient la liste des confirmations (IIème Partie 2.5). Un article de ce fichier contient les informations suivantes :

* numéro du message confirmé.
* nom du destinataire.
* date d'envoi.
* sujet du message confirmé.

Chaque fois que la fonction d'annonce des messages en attente et des confirmations (IIème Partie 3.1.1.1.) est exécutée, le contenu du fichier est affiché à l'écran avant d'être détruit.

1.2.1.3. Fichier "LST".

Les fichiers du type "LST" contiennent les listes de destinataires créées par l'utilisateur. Le nom d'un fichier "LST" est constitué de la façon suivante :

* suffixe = LST
* préfixe = nom donné par le créateur de la liste.

  Ex : STUDENT. LST

L'article d'un fichier "LST" est constitué d'un nom de login.

1.2.1.4. Dénomination des fichiers.

Dans le paragraphe 1.1.2, nous avons vu que le fait de ne disposer que d'un seul répertoire pour l'ensemble des fichiers constitués par le courrier électronique pose un problème de dénomination de fichiers. Dans ce paragraphe va être expliqué comment sont créés les noms de fichier propres à chaque utilisateur.

* préfixe

Un préfixe de nom de fichier d'utilisateur est constitué de six caractères. Il est fabriqué à partir du nom de login de l'utilisateur propriétaire du fichier. Trois cas peuvent se présenter lors de la composition d'un préfixe :

— nom de login compte moins de six caractères.

Ex ; nom de login= BILL préfixe = BILLXX

Les positions restantes sont complétées par le caractère "X".

— nom de login compte six caractères.

Ex : nom de login = ALEXIS préfixe = ALEXIS

— nom de login compte plus de six caractères

Ex : nom de login = ALEXANDRE préfixe= ALEXAN

Les caractères dépassant la sixième position sont tronquées,

* suffixe

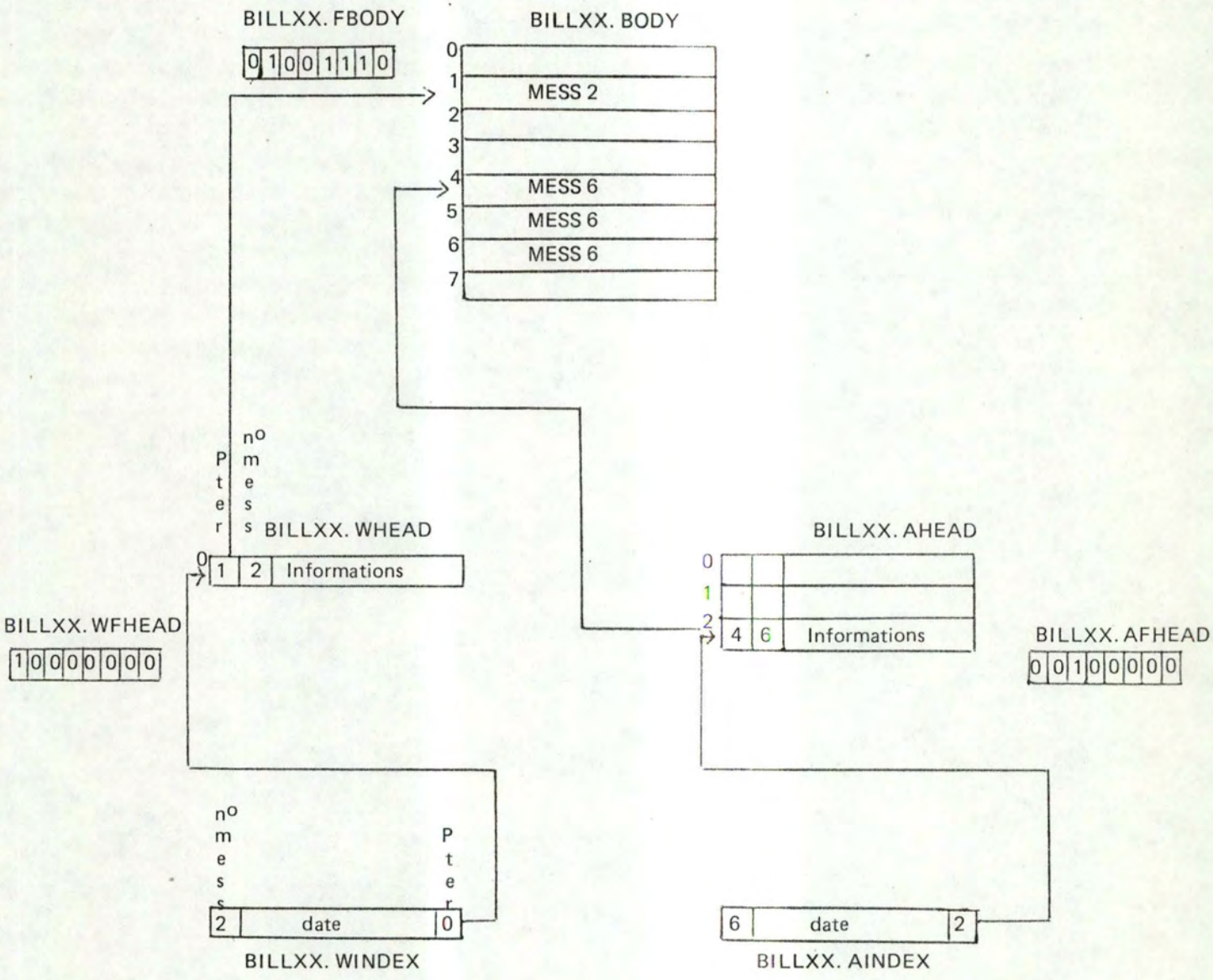
Voici la liste des suffixes possibles avec les types de fichiers auxquels ils correspondent :

- BODY : fichier "BODY"
- FBODY : fichier "FBODY"
- WHEAD : fichier des en-têtes de messages en attente
- WFHEAD : fichier des tables de blocs libres d'un fichier WHEAD
- AHEAD : fichier des en-têtes de messages archivés
- AFHEAD : fichier des tables de blocs libres d'un fichier AHEAD
- WINDEX : fichier d'index pour les messages en attente
- AINDEX : fichier d'index pour les messages archivés
- CONF : fichier des confirmations

1.2.1.5. Exemple.

Voici un exemple réduit (un message en attente, un message archivé) de fichiers de message appartenant à l'utilisateur dont le nom de login est BILL.

Hypothèse : une table de blocs libres se réduit à un byte (8 bits).


BILLXX. FBODY
0 1 0 0 1 1 1 0
BILLXX. BODY
0
1
MESS 2
2
3
4
MESS 6
5
MESS 6
6
MESS 6
7
nº mess Pter
BILLXX. WHEAD
0
1 2 Informations
BILLXX. AHEAD
0
1
2
4 6 Informations
BILLXX. WFHEAD
1 0 0 0 0 0 0 0
BILLXX. AFHEAD
0 0 1 0 0 0 0 0
nº mess
Pter
2 date 0
BILLXX. WINDEX
6 date 2
BILLXX. AINDEX

1.2.2. Fichiers généraux.

1.2.2.1. Fichier USER.LST

Ce fichier contient le répertoire des utilisateurs.

Un article de ce fichier contient les informations suivantes :

* nom de login du membre.
* patronyme du membre.
* mot de passe (encrypté) choisi par l'utilisateur.

Les articles sont triés par ordre alphabétique sur le nom de login.

L'organisation du fichier est séquentielle.

1.2.2.2. Fichier NBER.LAST

Ce fichier contient le dernier numéro de message utilisé. A l'initialisation du système de courrier électronique, le numéro est mis à 0. Ce chiffre est incrémenté de une unité chaque fois qu'un message est envoyé.

2. Description des traitements.

2.1. Description des niveaux.

Les modules de traitement du système de courrier électronique sont découpés en quatre niveaux. Etant donné un module, il n'est appelable que par un module de même niveau ou de niveau supérieur.

2.1.1. Niveau 1.

Le niveau 1 contient :

* le module principal : sa fonction est d'appeler le module d'affichage des messages en attente et des confirmations (CONFWAIT) et de passer la main au module de choix des commandes de niveau 1 (GETCMD1)
* module de choix des commandes (GETCMD1)
* module d'exécution des commandes correspondant aux fonctions de niveau 1 décrites dans la IIème Partie, chapitre 3.1. Voici le tableau qui, à chaque fonction de niveau 1, fait correspondre un module de niveau 1.

FONCTION	MODULE
– Annonce des messages en attente et confirmations	CONFWAIT
– Envoi d'un message	SEND
– Lecture des messages en attente	READMESS ('W') (1)
– Lecture des messages archivés	READMESS ('A') (1)
– Devenir membre du courrier	NEWMBER
– Se retirer du courrier	DELMBER
– Créer une liste de destinataires	CREATLST
– Détruire une liste de destinataires	DELETLST
– Consulter le répertoire des utilisateurs	GETUSER

(1) Les parenthèses signifient que des paramètres sont passés au module ; dans ce cas-ci le 'W' indique qu'il faut lire le fichier des messages en attente, le 'A' qu'il faut lire le fichier des messages archivés.

2.1.2. Niveau 2,

Le niveau 2 résulte d'une décomposition de modules de niveau 1 dont la taille aurait été trop importante à écrire et à mettre au point d'un seul tenant. Ainsi le module CONFWAIT est décomposé en deux modules de niveau 2 : TYPECONF (Affichage des confirmations) et TYPEWAIT (Affichage des messages en attente).

Le module de niveau 1 GETUSER est décomposé en les modules de niveau 2 :

* USERLOG : Affichage des articles du répertoire dont le nom de login correspond à celui tapé dans la ligne de commande.
* USERPATRO : Affichage des articles du répertoire dont le patronyme correspond à celui tapé dans la ligne de commande.
* USERALL : Affichage de tout le répertoire.

Le module READMESS est également découpé en sous-module selon la clé de sélection choisie dans la ligne de commande :

CLE de SELECTION	MODULE (niveau 2)
— tous les messages	ALLES
— messages envoyés par un émetteur donné	FROM
— messages envoyés avant une date donnée	BEFORE
— messages envoyés après une date donnée	AFTER
— messages dont le numéro est donné	NUMBER
— messages urgents	IMPORURG ('U')
— messages importants	IMPORURG ('I')
— messages dont le sujet contient une chaîne de caractères donnés	SUBJECT

Par contre le module SEND n'est pas décomposé en sous-modules de niveau 2. En effet, les sous-modules appelés par SEND sont également appelés par un module de niveau 3 ; c'est pourquoi les sous-modules de

SEND ne sont pas de niveau 2, mais de niveau 3.

2.1.3. Niveau 3.

Le niveau 3 est constitué de :

* modules d'affichage du message ayant satisfait à la sélection demandée lors d'une commande TYPE ou SEARCH (voir mode d'emploi en annexe)
* module de traitement du message courant :
  - module demandant à l'utilisateur de choisir une commande (GET CMD2)
  - module exécutant la commande demandée (fonction de niveau 2 et 3)

FONCTION	MODULE
Imprimer un message	PRINT
Détruire un message	KILL
Faire suivre un message	FORWARD
Répondre à un message	REPLY
Exécuter une commande de l'O.S.	EXECSHELL
Sauver un message sur fichier	SAVE

La fonction " Garder un message en attente" ne nécessite aucun traitement. La fonction "Imprimer + Détruire un message" est obtenue en exécutant les modules PRINT et KILL à la suite l'un de l'autre.

* sous-modules utilisés par SEND et REPLY

GETBODY : mémorisation du message tapé et préparé.

DIALOG : demande à l'utilisateur de remplir les formalités d'envoi.

DATNUM : garnissage de l'en-tête par le courrier : nom d'émetteur, numéro de message, date et heure d'envoi.

SENDLST : transfert des messages vers les utilisateurs individuels.

TOLST : transfert des messages vers les membres d'une liste.

2.1.4. Niveau 4.

Le niveau 4 contient les modules utilitaires et notamment les modules de gestion de fichier. J'insiste sur ce point, car ce fut un gros problème rencontré dans l'implémentation du système de courrier électronique sur UNIX.

En effet, UNIX ne dispose pas de logiciel d'organisation de fichiers. Or gérer un système de courrier électronique est sans doute avant toute chose un problème de manipulation d'informations, donc de gestion de fichiers. Il a donc fallu créer un mini-logiciel de gestion de fichiers, composé des modules nécessaires à la manipulation des messages. Pour la description de ces modules, je renvoie à l'annexe 2.

2.2. Description des modules.

2.2.1. Méthode de description.

La description d'un module se fait en six points :

1. But

   Dans ce point est expliqué grossièrement ce que fait le module.

2. Fonction = quoi ?

   Ce que réalise la fonction en détail.

3. Algorithme = comment ?

   Quels sont les moyens utilisés pour atteindre le but fixé ?

4. Entrées.

   Dans les entrées sont reprises les informations utilisées lors de l'exécution du module.

   Elles sont divisées en :

   4.1. Fichiers

   4.2. Variables globales

   4.3. Paramètres

5. Sorties.

   Les sorties reprennent les informations modifiées lors de l'exécution du module. Elles sont divisées en :

   5.1. Fichiers

   5.2. Variables globales

   5.3. Paramètres

   Attention, une entrée **non modifiée** n'est pas reprise en sortie, ce qui ne veut pas dire qu'elle est détruite après l'exécution du module.

6. Programmes appelants.

   Ici sont repris les modules qui, lors de leur exécution font appel au module décrit.

2.2.2. Exemple.

Avant de donner un exemple, des explications quant à certaines formulations utilisées dans la description des traitements sont nécessaires.

1. Appel d'une routine.

   Si le nom de la routine appelée n'est pas suivi de parenthèses, l'appel se fait sans passage de paramètres ; sinon les paramètres passés sont placés entre parenthèses.

   Exemple : appel de MAKEWAIT (LOGIN)

   le paramètre passé à la routine MAKEWAIT est LOGIN

2. Noms de fichier.

   Certains noms de fichier sont exprimés sous la forme (nom) suffixe

   Exemple : (LOGIN). AHEAD

   (LOGIN) signifie que le suffixe est créé à partir de la donnée contenue dans LOGIN comme expliqué en 1.2.1.4.

3. Variables.

   Certaines variables sont exprimées sous la forme :

   NOM1.NOM2

   NOM1 = nom d'une structure

   NOM2 = élément de la structure NOM1

Voici maintenant un exemple de description de module :

CREATLST

1. But.

   Création d'une liste de destinataires.

2. Fonction.

   – Indiquer à l'utilisateur qu'il peut taper la liste des login names

   – transférer les noms tapés par l'utilisateur dans le fichier dont le préfixe du nom est donné par l'utilisateur et dont le suffixe est LST

3. Algorithme.

   – ranger le nom pointé par PTLINE dans NAMELST

   – créer le fichier (NAMELST). LST

   si le fichier existe déjà :

   afficher : "Allready such a list."

   sinon : tant que l'utilisateur n'a pas tapé "(CR)"

   – si le contenu de la ligne tapée a une longueur >8 char :

   – afficher "To long logname."

   – détruire le fichier (NAMELST).LST

   – retourner au prog appelant

   sinon : transférer le contenu de la ligne ds (NAMELST).LST

4. Entrées.

   4.1. Fichiers

   –

4.2. Variables globales

– LINE

– PTLINE

4.3. Paramètres

–

5. Sorties.

5.1. Fichiers

– (NAMELST).LST

5.2. Variables globales

–

5.3. Paramètres

–

6 Programmes appelants.

– GETCMD1

2.3. Choix du langage de programmation.

Le langage de programmation choisi pour implémenter le système de courrier électronique sur UNIX est le langage C ([1 8]). La raison principale de ce choix est le fait que pratiquement tout l'O.S. de UNIX est écrit en C. Le courrier électronique, en tant qu'utilitaire, fait partie du système d'exploitation. C'est donc pour rester cohérent avec l'ensemble du système que le langage C a été choisi comme langage de programmation du courrier électronique. De plus, c'est un langage possédant des dispositifs intéressants (structures, pointeurs) et souple à utiliser. Son gros défaut est sans doute une formalisation difficile à employer.

2.4. Méthode de test.

Les premières routines à avoir été écrites et testées furent les routines de gestion des fichiers de messages, car c'est la base du courrier électronique. Donc si les routines de gestion de fichiers ne tournent pas, il est impossible de tester le reste du système. Cela étant acquis, les autres modules ont été testés de façon "top-down" fonction par fonction en choisissant la fonction à tester de telle sorte que les tests ultérieurs en soient facilités. C'est donc tout d'abord la fonction d'insertion d'un nouvel utilisateur qui a été testée, car cela permettait d'envoyer par après des messages à un utilisateur connu dont les fichiers existaient déjà. En effet, lorsqu'un utilisateur s'inscrit comme membre du courrier, les fichiers de messages le concernant sont créés et initialisés (Annexe 2 : Description des traitements page 4 : NEWMBER). Ensuite la fonction d'envoi de messages fut testée, car cela permettait de garnir les fichiers de messages. Une fois cette fonction au point, il devenait facile de tester la fonction de lecture des messages, car les fichiers étaient créés et contenaient des messages. D'autre part, il est évident que la fonction de création d'une liste fut testée avant celle de destruction d'une liste. Les autres fonctions étant indépendantes les unes des autres, l'ordre de test n'avait plus d'importance.

Conclusion.

Le but de ce mémoire était d'équiper le système UNIX d'un courrier électronique. Toutes les fonctions décrites dans la deuxième partie du mémoire ont été programmées et testées. Les tests ont montré que le système fonctionnait. L'implémentation d'un tel système a nécessité une étude complète, depuis l'analyse fonctionnelle jusqu'à la programmation et la mise au point du logiciel. Là réside sans doute l'intérêt de ce mémoire : en réalisant un projet informatique du début à la fin, j'ai pu me rendre compte des problèmes qui apparaissent dans la réalisation d'un travail d'une certaine envergure. Il y a souvent un fossé entre les idées émises lors de la conception d'un projet et la concrétisation de ces idées. On rencontre lors de la réalisation des tas de petits problèmes qui la rendent plus difficile qu'on ne le pensait.

De plus, ce mémoire m'a permis de connaître une nouvelle discipline de l'informatique : la bureautique. Peut-être serait-il intéressant d'améliorer le courrier électronique proposé dans ce mémoire et de développer autour de celui-ci une série de projets tels qu'éditeur de texte, téléconférence... afin de construire un système bureautique à l'institut d'informatique.

Bibliographie.

1. L.Naugès

La révolution informationnelle ou les mutations du monde de bureau

dans informations IBM nº 95 (2ème trimestre 1980) : pages 40 - 47

2. R. Uhlig, D. Farber, J. Bair

The office of the future

édité par North-Holland (1979)

3. M. Turoff, S. Hiltz

Meeting through your computer

dans IEEE Spectrum (Mai) pages 58 -64

4. K. Schoens

Mail Reference on UNIX (Version 1.3.)

édité par Berkeley university (Avril 1979)

5. M. Turoff, T. & P. Johnson-Lenz

How to use EIES (Electronic Information Exchange System)

édité par New Jersey Institute of Technology (Septembre 1977)

6. G. Engel, J. Groppuso, Lowenstein

An office communication system

dans IBM System Journal Vol. 18 nº 3 (1979) pages 402 - 431

7. H. Linpinski, R. Miller

Forum : a computer - assisted communications medium

dans Proceedings of the ICCC (1974) page T 4189

8. B. Kernigham, D. Ritchie

The C Programming language

édité par Prentice-Hall (1978)

9. D. Ritchie, K. Thompson

The UNIX Time-Sharing System

dans Communications of the ACM, Vol. 17 n°7 (Juillet 1974)

10. J.P. Adans

Mesure de performances de UNIX sous une charge universitaire.

Mémoire présenté en vue de l'obtention du grade de licencié et maître en sciences informatiques (1979)

ANNEXES

A1 : Mode d'emploi et sa critique

A : Mode d'emploi.

1 Entrée et sortie du courrier.

Pour entrer dans le courrier, tapez la commande NMAIL.

Le commentaire :

New goodbye user ! Type send search list rmlist exit.
*

est affiché. Vous pouvez alors exécuter une des commandes de NMAIL.

Pour sortir du courrier, tapez la commande EXIT.

2 Fonctions principales.

2.1. Annonce des messages en attente et des confirmations.

Lorsque vous tapez votre LOGIN, le courrier annonce si vous avez des messages en attente :

Les informations données sur les messages en attente sont les suivantes :

- Numéro de message
- login name de l'émetteur
- si le message est urgent ou important
- date d'envoi
- sujet du message

La présentation est la suivante :

You have mail.
34 from : bill Feb 25 14 : 32 exemple

40 from : boule urgent ! Feb 26 09 : 32 exemple

63 from : jeff important ! Feb 28 14 : 45 exemple

Les confirmations sont également affichées :

Les informations contenues dans une confirmation sont :

- numéro du message
- nom du destinataire
- date d'envoi

La présentation est la suivante :

Confirmations.

30 to : jeff Feb 24 14 : 20

2.2. Envoi d'un message.

Pour envoyer un message, tapez la commande SEND. Vous pouvez alors fabriquer votre message. Indiquer la fin de message par le catactère " . " en début de ligne. Le courrier demande alors si vous voulez utiliser l'éditeur de UNIX pour corriger votre message :

Do you want to use ED (y/n) ?

Si vous répondez "y", vous entrez dans l'éditeur ; vous n'avez pas à donner de nom de fichier car cela est géré par le courrier.

Lorsque le message est frappé et édité, le courrier demande :

OK to send (y/n) ?

Si vous répondez "y", le courrier demande :

— to :

Vous avez 3 possibilités de réponse :

- suite de login name séparés par " "
- list name
- combinaison des 2 premières possibilités

— Cc :

Vous avez 3 possibilités de réponse :

- suite de login name séparés par " "
- list name
- combinaison des 2 premières possibilités

— Subject :

Vous y mettez ce que vous voulez (rien, à la rigueur) du moment que le nombre de caractères de la zone est plus petit que 50.

– URGENT or IMPORTANT ?

Vous pouvez répondre : u pour URGENT

ou

i pour IMPORTANT

ou taper directement sur RETURN.

– SECRET (y/n) ?

Si vous répondez "y", le destinataire devra donner son mot de passe pour pouvoir lire le message.

Finalement, le courrier vous demande si vous voulez sauver le corps de message dans un fichier de votre choix :

Save it in a work file (y/n) ?

L'utilisation de cette commande est identique à celle de la commande SAVE (3.2.)

2.3. Lecture des messages en attente.

Pour afficher les messages en attente, utilisez la commande TYPE.

La syntaxe exacte est :

TYPE "nom de clé" "valeur de la clé"

Voici le tableau des différents types de clés avec le type de valeurs qu'il faut leur associer.

CLE	* VALEUR	* signification
FROM	* login-name	* messages envoyés par l'émetteur "login-name"
BEFORE	* date (1)	* messages envoyés avant "date"
AFTER	* date (1)	* messages envoyés après "date"
NUMBER	* séquence (2)	* messages dont le numéro est précisé dans "sequence"
URGENT	*	* messages mentionnés URGENT par l'émetteur
IMPORTANT	*	* messages mentionnés IMPORTANT par l'émetteur
SUBJECT	* string	* messages dont le sujet contient la chaîne "string"
(blanc)	*	* tous les messages en attente

(1) date = JJ: MM : AA

(2) séquence = suite de numéros séparés par " "

ou

numéro 1 : numéro 2 (messages dont le numéro est compris entre numéro 1 et numéro 2.)

Les messages précédés de leur header sont affichés un à un.

Si le message à afficher est secret, le courrier demande votre mot de passe en posant la question : "Your password, please ?"

Vous avez trois chances pour taper votre mot de passe correctement. Si vous n'y avez pas réussi, le commentaire :

"Sorry, you are not authorized to read message (nº de message)" est affiché.

Pour tout message affiché, les commandes suivantes sont disponibles :

PRINT : imprimer le message.

KILL : détruire le message.

PRKILL : imprimer puis détruire le message.

SAVE : sauver le corps de message dans un fichier de travail. Le courrier demande dans quel fichier vous voulez transférer le corps en posant la question :

"File name (< 15 char) ?"

Le nom que vous donnerez au fichier doit être composé d'au plus 15 caractères.

Si le fichier existe déjà, le message est ajouté en fin de fichier, sinon un fichier du nom que vous avez donné sera crée et le corps de message sera transféré dans le nouveau fichier.

KEEP : garder le message dans le fichier des messages en attente.

FORWARD : faire suivre le message vers d'autres utilisateurs.

Le courrier demande "To :".

Vous répondez par :

- suite de login names séparés par ' '
- nom de liste
- combinaison des 2 premières possibilités

REPLY : utilisation équivalente à la fonction d'envoi d'un message (2.2.) si ce n'est que vous ne devez plus donner le nom du destinataire ni le sujet (le destinataire recevra dans le domaine "Subject" du message la mention :

Reply to message "numéro de message" ("date")

Pour passer à l'affichage du message suivant, tapez sur RETURN.

2.4. Lecture des messages archivés.

La commande SEARCH permet la recherche de messages archivés, les clés de recherche sont celles disponibles pour la lecture des nouveaux messages (3.1.)

Les messages ainsi trouvés sont affichés un à un. Les diverses commandes possibles sur ces messages sont : PRINT KILL PRKILL SAVE (voir explications en 3.1.)

3. Fonction annexes.

3.1. Consulter le répertoire des utilisateurs.

Pour afficher une ou plusieurs entrées du répertoire des utilisateurs, tapez la commande USER "clé de recherche" "valeur de la clé"

"clé de recherche" — LOGIN (login name)
— NAME(patronyme)
— (blanc) (pour obtenir la liste complète.)

n.b. : il n'est pas nécessaire de donner toute la clé. Si vous donnez les n premières lettres de la clé, vous recevrez en retour la liste de toutes les entrées dont la clé commence par les n premières lettres données.

3.2. Exécuter une commande du SHELL.

Il est possible d'exécuter une commande du SHELL tout en ne quittant pas le courrier. Pour ce faire, tapez le catactère " ! ".
Le courrier répond en affichant le caractère " % ". Vous pouvez alors taper la commande de votre choix.

3.3. Créer ou détruire une liste de destinataires.

LIST "list name" crée une liste du nom de "list name". Si une liste de ce nom existe déjà, le commentaire

"Already such a list in mail system" est affiché.

Si ce n'est pas le cas, le commentaire

"Login names ?" est affiché.

Tapez les login name des membres de la liste.

Entre chaque login name, tapez sur la touche RETURN .

Marquez la fin de liste en frappant le caractère '.' en début de ligne.

Si vous voulez détruire une liste, utilisez la commande

RMLIST "list name"

Si la liste n'existe pas, le commentaire

"No such list" est affiché.

3.4. Devenir membre du courrier

Rentrez dans le courrier par NMAIL.

Tapez la commande NEW.

Il est impérieux que lorsque vous tapez cette commande, vous vous trouviez dans votre répertoire, car le courrier se sert de votre login name pour créer une entrée dans le répertoire des utilisateurs.

Le courrier demande votre nom (patronyme !) :

"Your name, please (< 25 char) ?"

La réponse doit contenir moins de 25 caractères.

Après cela, le courrier demande le mot de passe que vous désirez utiliser dans NMAIL :

"Your password, please ?"

Si vous existez déjà en tant qu'utilisateur du courrier, cela vous est signalé par :

"You are already in system mail"

3.5. Ne plus être membre du courrier

Rentrez dans le courrier par la commande NMAIL.

Tapez la commande GOODBYE

B. Critique du mode d'emploi.

Après utilisation du courrier électronique, il s'est avéré que le système n'était pas toujours très pratique à manipuler. Voici les critiques émises par les utilisateurs :

1. Il manque une commande "HELP".
2. Lors de commande SEND, aucun commentaire n'annonce que l'émetteur peut commencer à préparer son message.
3. Les listes de commande ne sont pas assez explicites. Cette remarque est à rapprocher du point 1.
4. Le fait que ce soit toujours une * qui annonce à l'utilisateur qu'il a la main, est équivoque. Il faudrait un signal différent par niveau de commande.
5. Lors de la commande NEW, dans le cas où la personne est déjà dans le répertoire des utilisateurs, le message "You are already in mail system" n'apparait que lorsque l'utilisateur a donné tous les renseignements ; ce message devrait apparaître dès que la commande a été tapée.
6. Lorsqu'une personne utilise le courrier électronique sans en être membre, ça devrait lui être signalé.
7. Dans la liste des commandes du niveau 1 (NEW goodbye user...) appuyer sur la touche "RETURN" est considéré comme une erreur. Ce devrait être considéré comme une commande sans effet.
8. Aucune commande ne permet d'afficher les membres d'une liste de destinataires.

A 2 Description organique

1 Description des informations

1.1. Structures

1.1.1. HEADER — header associé à un message

- . PTBODY — pointeur vers le corps de message correspondant
- . LMESS — longueur du corps de message
- . NBER — numéro du message
- . DATHEUR — date et heure d'envoi du message
- . SENDER — nom de l'émetteur du message
- . GROUP — si message de groupe : nom du groupe à qui est destiné le message
- . IORU — indicateur :
  - si IORU = ' i ', le message est important
  - si IORU = ' u ', le message est urgent
- . SUBJECT — sujet du message

1.1.2. INDEX — index associé à un message

- . NBRE — numéro du message
- . DATE — date d'envoi du message
- . PTHEAD — pointeur vers le header correspondant

1.1.3. MEMBER — informations relatives à un utilisateur du courrier

- . LOG — login name du nouvel utilisateur
- . NAME — nom du nouvel utilisateur
- . PASS — mot de passe du nouvel utilisateur

1.1.4. CONF — confirmation envoyée lorsqu'un message en attente est lu

- . NBER — numéro du message confirmé
- . DEST — nom du destinataire du message confirmé
- . DATHEUR — date et heure d'envoi du message confirmé
- . SUBJECT — sujet du message confirmé

1.2. Variables simples.

1.2.1. Variables contenant les noms de fichier.

- – BODY — variable contenant nom du fichier body courant
- – FBODY — variable contenant nom du fichier contenant la table des blocs libres dans un fichier body
- – WHEAD — variable contenant nom du fichier des headers en attente courant
- – WFHEAD — variable contenant nom du fichier contenant la table des blocs libres dans un fichier des headers en attente courant
- – WINDEX — variable contenant nom du fichier d'index associé aux fichiers body et des headers en attente courants
- – AHEAD — variable contenant nom du fichier des headers archivés coutant
- – AFHEAD — variable contenant nom du fichier contenant la table des blocs libres dans le fichier des headers archivés courant
- – AINDEX — variable contenant nom du fichier d'index associé aux fichiers body et des headers archivés courants
- – CONFIRM — variable contenant nom d'un fichier des confirmations

1.2.2. File descriptors.

- FHD — file descriptor du fichier HEAD courant
- FBD — file descriptor du fichier BODY courant
- FIN — file descriptor du fichier INDEX courant
- FD — file descriptor du fichier dont le contenu doit être sauvé par une commande SAVE
- FDO — file descriptor du fichier dont le contenu doit être sauvé dans un fichier BODY

1.2.3. Varia

1.2.3. Variables de travail.

- DEST — liste des destinataires d'un message
- CC — liste des carbon copies d'un message
- LINE — ligne de commande
- PTLINE — pointeur vers le caractère courant dans LINE
- NBLOC — nombre de blocs dont est constitué un corps de message
- DATE — date transformée sous la forme aammjj
- TOKILL — indicateur :

  si TOKILL = 1 : détruire le message courant
- TOKEEP — indicateur :

  si TOKEEP = 1 : détruire le message en attente courant
- LOGIN — login name de l'utilisateur courant
- LOGKEY — login name sur lequel est effectué une recherche dans le répertoire des utilisateurs
- NAME — préfixe d'un nom de fichier

2. Description des traitements

- Niveau 1 -

MAIN
----

1. But .

Ensemble des fonctions du courrier .

2. Fonction .

- obtenir le login name de l'utilisateur
- affichage des headers des messages en attente
- affichage des confirmations
- tant que l'utilisateur le demande , executer la commande tapee

3. Algorithme

- appel READLOG
- appel de CONFWAIT
- afficher: "* "
- tant que l'utilisateur n'a pas tape "exit" :
    - appel de GETCMD1

4. Entrees .

-

5. Sorties.

5.1. Fichiers

-

5.2. Variables globales

- LOGIN

5.3. Parametres

-

6. Programmes appelants .

-

GETCMD1
----

1. But .

Quelle fonction du courrier veut utiliser l'utilisateur ?

2. Fonction .

- lire la commande tapee par l'utilisateur
- si la commande tapee est correcte : executer cette

commande
- si la commande est erronee : afficher un message d'erreur

3. Algorithme

- appel de GETLINE
- si LINE = "type" appel de READMESS('w')
- si LINE = "search" appel de READMESS('a')
- si LINE = "send" appel de SEND
- si LINE = "user" appel de GETUSER
- si LINE = "list" appel de CREATLST
- si LINE = "delete" appel de DELETE
- si LINE = "!" appel de EXECSHELL
- si LINE = "exit" retour au programme appelant
- sinon : affichage de : "Unknown command"

4. Entrees .

-

5. Sorties.

5.1. Fichiers

-

4.2. Variables globales

- LINE

- PTLINE

4.3. Parametres

Valeur retournee : 0 si l'utilisateur a tape sur "exit"
1 sinon

6. Programmes appelants .

- MAIN

NEWMBER
---

1. But .

Operations a realiser pour qu'un utilisateur du systeme devienne membre du courrier .

2. Fonction .

- demander quel mot de passe l'utilisateur desire avoir dans le courrier
- demander le nom de l'utilisateur
- composer le nom des fichiers propres a l'utilisateur avec comme suffixe le login name de l'utilisateur

- creer et initialiser ces fichiers .
- inserer les informations relatives a l'utilisateur dans le repertoire des utilisateurs (USER.LST) .
- n.b. : si l'utilisateur existe deja dans le courrier , cela lui est indique sans que ses fichiers de messages soient modifies .

3. Algorithme .

- appel de INSMBER
- si valeur retournee par INSMBER = 0 : retourner au programme appelant ;
  sinon :
  - appel de MAKEWAIT(LOGIN)
  - creer et initialiser les fichiers :
  (LOGIN).BODY
    .FBODY
    .WHEAD
    .WFHEAD
    .WINDEX
    .CONF
  - appel de MAKEARCH(LOGIN)
  - creer et initialiser les fichiers :
  (LOGIN).AHEAD
    .AFHEAD
    .AINDEX

4. Entrees .

4.1. Fichiers

-

4.2. Variables globales

- LOGIN

4.3. Parametres

-

5. Sorties.

5.1. Fichiers

- tous les fichiers propres au nouvel utilisateur sont crees et initialises . (voir description des info)

- USER.LST

5.2. Variables globales

- MEMBER.LOG
  .NAME
  .PASS

5.3. Parametres

-

6. Programmes appelants

-

INSMBER

1. But .

Insertion des informations relatives a un membre dans le repertoire des utilisateurs (USER.LST) .

2. Fonction .

- demander le patronyme de l'utilisateur
- demander a l'utilisateur le mot de passe qu'il desire avoir dans le courrier ; cryptage de ce mot de passe
- inserer MEMBER dans USER.LST

3. Algorithme .

- ranger LOGIN dans MEMBER.LOG
- afficher : "Your name , please ? "
- rangement de la reponse dans MEMBER.NAME
- afficher : "Your password , please ? "
- cryptage du mot de passe et rangement dans MEMBER.PASS
- tant que MEMBER n'a pas ete insere dans USER.LST et non EOF :
  - lire un article de ce fichier
  - si MEMBER.LOG = login name lu dans USER.LST :
    afficher : "You are already in system mail"
    retourner la valeur 0
  - si MEMBER.LOG > login name lu dans USER.LST :
    inserer MEMBER dans le fichier
- si MEMBER n'a pas ete insere , l'ajouter en fin de fichier
- retourner la valeur 1

4. Entrees .

4.1. Fichiers

- USER.LST

4.2. Variables globales

- LOGIN

4.3. Parametres

-

4. Sorties.

4.1. Fichiers

- USER.LST

4.2. Variables globales

- MEMBER.LOG
  .NAME
  .PASS

4.3. Parametres

-

5. Programmes appelants .

- NEWMBER

DELMBER
---

1. But .

Destruction des informations relatives a un membre dans le repertoire des utilisateurs (USER.LST) .

2. Fonction .

- obtenir le login name de l'utilisateur
- detruire dans USERLST les informations relatives a l'utilisateur .

3. Algorithme .

- tant que LOGIN > MEMBER.LOG :
  - lire un article de USER.LST dans MEMBER
- si LOGIN pas = MEMBER.LOG :
  afficher : "(LOGIN) isn't a mail member ."
- si LOGIN = member.LOG :
  - destruction de l'article dans USER.LST
  - appel de MAKEWAIT(LOGIN)
  - destruction des fichiers ayant comme prefixe LOGIN et comme suffixe : BODY FBODY WHEAD WFHEAD WINDEX
  - appel de MAKEARCH(LOGIN)
  - destruction des fichiers ayant comme prefixe LOGIN et comme suffixe : AHEAD AFHEAD AINDEX
  - destruction du fichier (LOGIN).CONF

4. Entrees .

4.1. Fichiers

- USER.LST

- PASSWRD

4.2. Variables globales

-

4.3. Parametres

- LOGIN

5. Sorties.

5.1. Fichiers

- USER.LST

5.2. Variables globales

- LOGIN

5.3. Parametres

-

6. Programmes appelants .

-

SEND
----

1. But .

Envoi d'un message .

2. Fonction .

- Enregistrer dans un fichier de travail le corps de message tape par l'utilisateur .
- Si l'utilisateur le desire , lui permettre d'utiliser l'editeur
- Demander a l'utilisateur s'il est d'accord pour envoyer le message :
  si oui :
  - obtenir - la date d'envoi
    - le numero du message
    - le nom d'emetteur

    (ces 3 renseignements ne necessite pas l'aide de l'utilisateur)
  - demander a l'utilisateur
    - le sujet du message
    - si le message est important ou urgent
    - si le message est secret
    - le login name du(des) destinataire(s) (et) carbon copy(ies)

- inserer la liste des destinataires et des cc au debut du corps de message .
- transferer le message (corps + header + index dans les fichiers de messages en attente du(des) destinataire(s)

- Si l'utilisateur le desire , sauver le corps de message dans un fichier de travail.

3. Algorithme .

- appel de GETBODY
- afficher "Ok to send (y/n) ? "
  si reponse = 'y' :
  - appel de DATNUM
  - appel de DIALOG
  - inserer DEST et CC en debut du fichier de travail contenant le message
  - appel de SENDLST(DEST)
  - appel de SENDLST(CC)

- afficher : "Save the body in a work file (y/n) ? "
  si reponse = 'y' :
  appel de SAVE(FDWRK,HEADER.LMESS)
- destruire le fichier de travail contenant le message

4. Entrees .

4.1. Fichiers

- NBER.LAST

4.2. Variables globales

- LOGIN

4.3. Parametres
- DEST
- CC

5. Sorties.

5.1. Fichiers

- NBER.LAST
- les fichiers de message des destinataires

5.2. Variables globales

- HEADER
- INDEX

5.3. Parametres

-

6. Programmes appelants .

- GETCMD1

READMESS
---

1. But .

Affichage des messages en attente ou des messages archives .

2. Fonction .

- si l'utilisateur a tape la commande "type" : composition des noms de fichier des messages en attente de l'utilisateur
- si l'utilisateur a tape la commande "search" : composition des noms de fichier des messages archives de l'utilisateur
- afficher les messages dont le contenu du header repond a la valeur de la cle associee a la commande

3. Algorithme .

- si WORA = 'w' : - appel MAKEWAIT(LOGIN)
- si WORA = 'a' : - appel MAKEARCH(LOGIN)
- si LINE = "from" : appel de FROM(WORA)
- si LINE = "before" : appel de BEFORE(WORA)
- si LINE = "after" : appel de AFTER(WORA)
- si LINE = "number" : appel de NUMBER(WORA)
- si LINE = "urgent" : appel de URGORIMP(WORA, 'u')
- si LINE = "important" : appel de URGORIMP(WORA, 'i')
- si LINE = "subject" : appel de SUBJECT(WORA)
- si LINE = "CR" : appel de ALLES(WORA)
- sinon :
  afficher : "Invalid key"

4. Entrees .

3.1. Fichiers

-

3.2. Variables globales

- PTLINE
- LINE
- LOGIN

3.3. Parametres

- WORA

4. Sorties.

4.1. Fichiers

-

4.2. Variables globales

- PTLINE
- LINE

4.3. Parametres

-

5. Programmes appelants .

- GETCMD1

GETUSER
---

1. But .

Consultation du repertoire des utilisateurs .

2. Fonction .

- Selon la demande de l'utilisateur :
  - afficher le repertoire
  - afficher les entrees du repertoire dont les premieres lettres du login sont egales aux lettres tapees apres "login" par l'utilisateur
  - afficher les entrees du repertoire dont les premieres lettres du nom sont egales aux lettres tapees apres "name" par l'utilisateur

3. Algorithme .

- si LINE = "login" : appel de USERLOG
- si LINE = "name" : appel de USERPATRO
- si LINE = CR : appel de USERALL

4. Entrees .

4.1. Fichiers

- USER.LST

3.2. Variables globales

- LINE
- PTLINE

3.3. Parametres

-

4. Sorties.

-

5. Programmes appelants .

- GETCMD1

CREATLST
---

1. But .

Creation d'une liste de destinataires .

2. Fonction .

- Indiquer a l'utilisateur qu'il peut taper la liste des login name
- transferer les noms tapes par l'utilisateur dans le fichier dont le prefixe du nom est donne par l'utilisateur et dont le suffixe est LST

3. Algorithme .

- ranger le nom pointe par PTLINE dans NAMELST
- creer le fichier (NAMELST).LST
  si le fichier existe deja :
    afficher : "Allready such a list ."
  sinon : tant que l'utilisateur n'a pas tape ".<CR>"
    - si le contenu de la ligne tapee par l'
      utilisateur a une longueur > 8 char :
        - afficher "To long logname ."
        - detruire le fichier (NAMELST).LST
        - retourner au prog appelant
      sinon : transferer le contenu de la ligne
        ds (NAMELST).LST

4. Entrees .

4.1. Fichiers

-

4.2. Variables globales

- LINE
- PTLINE

4.3. Parametres

-

5. Sorties.

5.1. Fichiers

- (NAMELST).LST

5.2. Variables globales

-

5.3. Parametres

-

5. Programmes appelants .

- GETCMD1

DELETLST

1. But .

Destruction d'une liste de destinataires .

2. Fonction .

- detruire la liste dont le prefixe est donne par l'utilisateur

3. Algorithme .

- rangement du nom pointe par PTLINE dans NAMELST
- destuction du fichier (NAMELST).LST

4. Entrees .

4.1. Fichiers

- (NAMELST).LST

4.2. Variables globales

- LINE
- PTLINE

4.3. Parametres

-

5. Sorties.

-

6. Programmes appelants .

- GETCMD1

CONFWAIT

---

1. But .

   Affichage des confirmations et des headers des messages en attente .

2. Fonction .

   voir 1.

3. Algorithme .

   - Appel de TYPEWAIT .
   - Appel de TYPECONF .

4. Entrees .

   4.1. Fichiers

   - (LOGIN).CONF
   - (LOGIN).WHEAD

   4.2. Variables globales

   - LINE
   - PTLINE
   - LOGIN

   4.3. Parametres

   -

5. Sorties.

   5.1. Fichiers

   - (LOGIN).CONF

   5.2. Variables globales

   -

   5.3. Parametres

   -

6. Programmes appelants .

   - main

- Niveau 2 -

TYPEWAIT
----

1. But .

Affichage des headers des messages en attente .

2. Fonction .

- constituer le nom de fichier (LOGIN).WHEAD
- lire et afficher les articles de ce fichier

3. Algorithme .

- Constitution du nom de fichier (LOGIN).HEAD .
- Tant que pas EOF :
  - lire un article de (LOGIN).CONF dans HEADER
  - appel de TYPEHEAD

4. Entrees .

4.1. Fichiers

- (LOGIN).WHEAD

4.2. Variables globales

- LOGIN

4.3. Parametres

-

5. Sorties.

-

6. Programmes appelants .

- CONFWAIT

TYPECONF
----

1. But .

Affichage des confirmations .

2. Fonction .

- Constitution du nom de fichier (LOGIN).CONF .
- Lire et afficher les articles de ce fichier
- Detruire tous les aricles du fichier

3. Algorithme .

- Constitution du nom de fichier (LOGIN).CONF
- Tant que pas EOF :
  - lire un article de (LOGIN).CONF dans CONFIRM
  - afficher :
    - CONFIRM.NBER
    - CONFIRM.READER
    - CONFIRM.HEUR
- detuire (LOGIN).CONF
- creer un fichier (LOGIN).CONF

4. Entrees .

4.1. Fichiers

- (LOGIN).CONF

4.2. Variables globales

- LOGIN

4.3. Parametres

-

5. Sorties.

5.1. Fichiers

- (LOGIN).CONF

5.2. Variables globales

-

5.3. Parametres

-

6. Programmes appelants .

- CONFWAIT

USERLOG
---

1. But .

Recherche de(s) entree(s) dans le repertoire dont les premieres lettres du login name sont egales aux lettres tapees apres "login" dans LINE .

2. Fonction .

Pour chaque article lu dans USER.LST :
tester si le login name repond a la valeur tapee apres "login" dans la ligne de commande ; si oui : afficher l'article au terminal .

3. Algorithme .

Tant que non EOF :
- lire un article de USER.LST ds MEMBER
- appel de COMPAR(PTLINE,MEMBER.LOG)
si egalite : afficher MEMBER a l'ecran

4. Entrees .

4.1. Fichiers

- USER.LST

3.2. Variables globales

- LINE
- PTLINE

3.3. Parametres

-

5. Sorties.

-

5. Programmes appelants .

- GETUSER

USERPATRO
---

1. But .

Recherche de(s) entree(s) dans le repertoire dont les premieres lettres du patronyme sont egales aux lettres tapees apres "name" dans LINE

2. Fonction .

Pour chaque article lu ds USER.LST :
tester si le nom ds l'article lu repond a la valeur tapee apres "name" ds la ligne de commande ; si oui afficher l'aricle lu .

3. Algorithme .

Tant que non EOF :
- lire un article de USER.LST ds MEMBER
- appel de COMPAR(PTLINE,MEMBER.NAME)

- si egalite : afficher MEWMBER a l'ecran

4. Entrees .

4.1. Fichiers

- USER.LST

4.2. Variables globales

- LINE
- PTLINE

4.3. Parametres

-

5. Sorties.

-

6. Programmes appelants .

- GETUSER

USERALL
---

1. But .

Affichage de toutes les entrees du repertoire .

2. Fonction .

Pour chaque article lu ds USER.LST :
- afficher l'article lu

3. Algorithme .

Tant que non EOF :
- lire un article de USER.LST ds MEMBER
- afficher le contenu de MEMBER

4. Entrees .

4.1. Fichiers

- USER.LST

4.2. Variables globales

- LINE
- PTLINE

4.3. Parametres

-

5. Sorties.

-

6. Programmes appelants .

- GETUSER

FROM
----

1. But .

Lecture des messages envoyes par un emetteur donne .

2. Fonction .

Pour chaque header lu ds (LOGIN).WHEAD ou (LOGIN).AHEAD :
(depend de la valeur de WORA)
- tester si le nom d'emetteur dans le header est egal au nom d'emetteur tape apres "from" :
si oui :
- si le message est secret : demander le mot de passe de l'utilisateur (celui-ci a 3 chances pou donner le mot de passe correct)
- si le message n'est pas secret ou si l'utilisateur a donne le mot de passe correct :
- afficher le message
- demander a l'utilisateur quelle(s) commande(s) il veut effectuer sur le message

3. Algorithme .

- Tant que non EOF :
- lire un article de (FHD) ds HEADER
- lire un article de (LOGIN).WHEAD ou (LOGIN).AHEAD
- appel de COMPAR(PTLINE,HEADER.SENDER)
- si egalite : appel de TRTMESS(WORA)

4. Entrees .

4.1. Fichiers

- (LOGIN).AHEAD ou (LOGIN).WHEAD
- (LOGIN).BODY
- (USER).LST

4.2. Variables globales

- PTLINE

- LINE
- AHEAD ou WHEAD
- ABODY ou WBODY

4.3. Parametres

- WORA

5. Sorties.

-

6. Programmes appelants .

- READMESS

BEFORE

1. But .

Lecture des messages envoyes avant une date donnee .

2. Fonction .

- Verifier la validite de la date tapee apres "before" ; si la date est valide , passer de la forme jjmmaa sous la forme aammjj
- Pour chaque article lu dans (LOGIN).AINDEX ou (LOGIN).WINDEX
  - comparer la date tapee apres "before" avec la date de l'index (dates sous la forme aammjj)
    date de l'index :
    - si le message est secret : demander le mot de passe de l'utilisateur
    - si le message n'est pas secret ou que l'utilisateur a donne le mot de passe correct :
      - afficher le message
      - executer la(les) commande(s) demandee(s) par l'utilisateur

3. Algorithme .

- appel de GETDATE(DATE)
  si date erronee : afficher "Uncorrect date ."
- Tant que non EOF :
  - lire un article de (LOGIN).WINDEX ou (LOGIN).AINDEX dans INDEX
  - si DATE < INDEX.DATE : appel de TRTMESS(WORA)

4. Entrees .

4.1. Fichiers

- (LOGIN).AINDEX ou (LOGIN).WINDEX

- (LOGIN).AHEAD ou (LOGIN).WHEAD
- (LOGIN).BODY
- USER.LST

4.2. Variables globales

- PTLINE
- LINE
- FBD
- FHD

4.3. Parametres

- WORA

5. Sorties.

-

6. Programmes appelants .

- READMESS

AFTER
---

1. But .

Lecture des messages envoyes apres une date donnee .

2. Fonction .

- verifier la validite de la date tapee apres "after" ; si la date est valide , passer de la forme jjmmaa sous la forme aammjj .
- Pour chaque article lu dans (LOGIN).AINDEX ou (LOGIN).WINDEX
  - comparer la date tapee apres "after" avec la date de l'index :
  - si la date lue apres "after" est anterieure a la date de l'index :
    - si le message est secret : demander le mot de passe de l'utilisateur
    - si le message n'est pas secret ou que l'utilisateur a donne le mot de passe correct :
      - afficher le message
      - executer la(les) commande(s) demandee(s) par l'utilisateur

3. Algorithme .

- Appel de GETDATE(DATE)
  si date erronee : afficher "Uncorrect date ."
- Tant que non EOF :
  - lire un article de (FDI) ds INDEX
  - si DATE > INDEX.DATE : appel de TRTMESS(WORA)

4. Entrees .

4.1. Fichiers

- (LOGIN).AINDEX ou (LOGIN).WINDEX
- (LOGIN).AHEAD ou (LOGIN).WHEAD
- (LOGIN).BODY
- USER.(LST)

4.2. Variables globales

- PTLINE
- LINE
- FBD
- FHD
- FDI

4.3. Parametres

- WORA

5. Sorties.

-

6. Programmes appelants .

- READMESS

NUMBER
---

1. But .

Lecture des messages dont les numeros sont donnes dans la ligne de ommande .

2. Fonction .

- Tester si les numeros sont tapes sous forme de sequence ou sous forme de rang (affichage d'un message d'erreur si forme est indeterminee)
- Affichage des messages dont le numeros sont indiques dans la ligne de commande (voir SEQUENCE et RANGE)

3. Algorithme .

- si le premier separateur rencontre dans LINE est ',' :
  - appel de SEQUENCE(WORA)
- si le premier separateur rencontre dans LINE est ':' :
  - appel de RANGE :
- sinon : afficher "Sequence or range error ."

4. Entrees .

4.1. Fichiers

- (LOGIN).AHEAD ou (LOGIN).WHEAD
- (LOGIN).AINDEX ou (LOGIN).WINDEX
- (LOGIN).BODY
- USER.LST

4.2. Variables globales

- PTLINE
- LINE
- AHEAD ou WHEAD
- AINDEX ou WINDEX
- ABODY ou WBODY

4.3. Parametres

- WORA

5. Sorties.

-

6. Programmes appelants .

- READMESS

SEQUENCE
---

1. But .

Lecture des messages dont les numeros appartiennent a la sequence precisee dans LINE .

2. Fonction .

- Pour chaque index lu dans (LOGIN).AINDEX ou (LOGIN).WINDEX :
  - si le numero de l'index lu est un numero de la sequence :
    - si le message est secret : demander le mot de passe de l'utilisateur
    - si le message est secret ou si l'utilisateur a donne le mot de passe correct :
      - afficher le message
      - executer la(les) commande(s) demandee(s) par l'utilisateur.

3. Algorithme

- Tant que fin de LINE(<==> CR) n'a pas ete trouvee :
  - lire LINE jusqu'au prochain separateur
  - ranger le numero obtenu dans une table apres avoir effectue la transformation ASCII -> numerique

- Tant que non EOF :
    - lire un article de (FDI) dans INDEX
    - Tant que pas fin de table et 'pas trouve' :
        - si INDEX.NBRE = numero de la table :
            - appel de TRTMESS(WORA)
            - indiquer : 'trouve'
        - sinon passer a l'element suivant de la table

4. Entrees .

4.1. Fichiers

- (LOGIN).AINDEX ou (LOGIN).WINDEX
- (LOGIN).AHEAD ou (LOGIN).WHEAD
- (LOGIN).BODY
- USER.LST

4.2. Variables globales

- PTLINE
- LINE
- FBD
- FHD
- FDI

4.3. Parametres

- WORA

5. Sorties.

-

6. Programmes appelants .

- NUMBER

RANGE

1. But .

Lecture des messages dont le numero appartient a l'intervalle constitue par les 2 numeros du rang .

2. Fonction .

- Verification de la coherence ; il faut que le deuxieme numero tape soit plus grand que le premier . S'il y a incoherence : affichage d'un message d'erreur .
- S'il y a coherence :
  pour chaque index lu dans (LOGIN).WINDEX ou (LOGIN).AINDEX :
    - si le numero de l'index lu est superieur au plus petit nbre tape et inferieur au plus grand nbre tape
        - si le message est secret : demander le mot

de passe de l'utilisateur
- si le message n'est pas secret ou si l'
utilisateur a donne le mot de passe correct :
- afficher le message
- executer la(les) commande(s)
demandee(s) par l'utilisateur

3. Algorithme .

- lire LINE jusqu'a ':'
- transformer le contenu de LINE en numerique dans NUM1
- lire LINE jusqu'a CR
- transformer le contenu de LINE en numerique dans NUM2
- si NUM1 > NUM2 : afficher "Range error ."
- si NUM1 < NUM2 :
- tant que non EOF :
- lire un article de (FDI) dans INDEX
- si NUM1 < INDEX.NBRE < NUM2 :
appel de TRTMESS(WORA)

4. Entrees .

4.1. Fichiers

- (LOGIN).AINDEX ou (LOGIN).WINDEX
- (LOGIN).AHEAD ou (LOGIN).WHEAD
- (LOGIN).BODY
- USER.LST

4.2. Variables globales

- PTLINE
- LINE
- FBD
- FHD
- FDI

4.3. Parametres

- WORA

5. Sorties.

-

6. Programmes appelants .

- NUMBER

IMPORURG
---

1. But .

Lecture des messages urgents ou importants .

2. Fonction .

Pour chaque header lu ds (LOGIN).WHEAD ou (LOGIN).AHEAD :
- si le message est important et que l'utilisateur a demande d'afficher les messages importants ou si le message est urgent et l'utilisateur a demande d'affiche les messages urgents :
  - si le message est secret , demander le mot de passe de l'utilisateur
  - si le message n'est pas secret ou si l'utilisateur a donne le mot de passe correct :
    - afficher le message
    - demander a l'utilisateur quelle(s) commande(s) il veut effectuer

3. Algorithme .

- Tant que non EOF :
  - lire un article de (FHD) dans HEADER
  - si HEADER.IORU = 'i' et iu = 'i'
    ou
    si HEADER.IORU = 'u' et IU = 'u' :
    appel de TRTMESS(WORA)

4. Entrees .

4.1. Fichiers

- (LOGIN).AHEAD ou (LOGIN).WHEAD
- (LOGIN).BODY
- USR.LST

4.2. Variables globales

- PTLINE
- LINE
- FHD
- FBD

4.3. Parametres

- WORA
- IU

5. Sorties.

-

6. Programmes appelants .

- READMESS

SUBJECT
---

1. But .

Lecture des messages dont le domaine "subject" contient la chaine de caractere donnee .

2. Fonction .

Pour chaque header lu dans (LOGIN).WHEAD ou (LOGIN).AHEAD :
- si le domaine "subject" contient la chaine de caracteres tapes dans la ligne de commande :
  - si le message est secret : demander le mot de passe de l'utilisateur
  - si le message n'est pas secret ou si l'utilisateur a donne le mot de passe correct :
    - afficher le message
    - demander a l'utilisateur quelle(s) commande(s) il veut effectuer

3. Algorithme .

- Tant que non EOF :
  - lire un article du fichier FHD ds HEADER
  - faire pointer PTSUB vers HEADER.SUBJECT
  - Tant que 'pas trouve' et HEADER.SUBJECT non entierement balaye :
    - appel de COMPAR(PTLINE,PTSUB)
    - si = : - indiquer 'trouve'
      - appel de TRTMESS(WORA)
    - si pas = : faire pointer PTSUB vers le caractere suivant de HEADER.SUBJECT

4. Entrees .

4.1. Fichiers

- (LOGIN).AHEAD ou (LOGIN).WHEAD
- (LOGIN).BODY
- USER.LST

4.2. Variables globales

- PTLINE
- LINE
- FHD
- FBD

4.3. Parametres

- WORA

5. Sorties.

-

6. Programmes appelants .

- READMESS

ALLES
---

1. But .

Lecture de tous les messages .

2. Fonction .

Pour chaque header lu dans (LOGIN).AHEAD ou (LOGIN).WHEAD :
- si le message est secret : demander le mot de passe de l'utilisateur
- si le message n'est pas secret ou si l'utilisateur a donne le mot de passe correct :
  - afficher le message
  - demander a l'utilisateur quelle(s) commandes il veut effectuer

3. Algorithme .

- Tant que non EOF :
  - lire un article de FHD ds HEADER
  - appel de TRTMESS(WORA)

4. Entrees .

4.1. Fichiers

- (LOGIN).AHEAD ou (LOGIN).WHEAD
- (LOGIN).BODY
- USER.LST

4.2. Variables globales

- PTLINE
- LINE
- FHD
- FBD

4.3. Parametres

- WORA

5. Sorties.

-

6. Programmes appelants .

- READMESS

- Niveau 3 -

TRTMESS
---

1 But .

Tester si un message est secret . si oui demander le mot de passe et verifier s'il est correct ; si oui , afficher le message .

2. Fonction .

Voir 1

3. Algorithme .

- appel de SECRETMESS
- si valeur retournee par SECRETMESS est 0 :
  appel de TYPEMESS(WORA)
- si valeur retournee par SECRETMESS est 1 :
  afficher "You are not authorized to read this message .

4. Entrees .

4.1. Fichiers

- (FBD)
- (FHD)

4.2. Variables globales

- FBD
- FHD

4.3. Parametres

- WORA

5. SORTIES

-

6. Programmes appelants

- FROM
- BEFORE
- AFTER
- SUBJECT
- RANGE
- NUMBER
- IMPORURG
- ALLES

SECRETMESS
---

1. But .

Tester si le message courant est secret et dans ce cas , demander le mot de passe .

2. Fonction .

- Le message courant est-il secret ?
  Si oui :
    - demander a l'utilisateur son mot de passe .
    - tester si le mot de passe est correct ?
        Si non : demander a l'utilisateur de retaper sor
                 mot de passe .
      (L'utilisateur a trois chances pour donner son mot de
        passe correct .)

3. Algorithme .

- si HEADER.SECRET pas = 's' : retourner la valeur 0
- si HEADER.SECRET = 's' :
    - appel de GETMBER
    - tant que 'reponse incorrecte' et
      que nbre de coups < 3 :
        - afficher "Your password , please ? "
        - si nom tape par l'utilisateur = MEMBER.PASS :
            'reponse correcte'
    - si 'reponse correcte' : retourner la valeur 0
    - si 'reponse incorrecte' : retourner la valeur 1

4. Entrees .

4.1. Fichiers

- USER.LST

4.2. Variables globales

- HEADER.SECRET
- LOGIN

4.3. Parametres

-

5. Sorties.

5.1. Fichiers

-

5.2. Variables globales

- MEMBER

5.3. Parametres

Valeur retournee : 1 <==> le message peut etre affiche

0 <==> le message ne peut etre
affiche

6. Programmes appelants .

- TRTMESS

TYPEMESS
---

1. But .

Affichage du message courant .

2. Fonction .

- affichage du header
- affichage du corps
- ecrire ds le fichier (HEADER.SENDER).CONF de l'emetteur
  la confirmation correspondant au header lu
- tant que l'utilisateur le demande :
  - executer une des commandes possibles sur les messages
    en veillant a la coherence des commandes demandees

3. Algorithme .

- appel de TYPEHEAD
- appel de TYPEBODY
- appel de WRITCONF
- tant que valeur retournee par GETCMD2 = 1 :
  - appel de GETCMD2(WORA, TOKILL, TOKEEP)
- si TOKEEP = TOKILL = 1 :
  afficher : "Stupid to keep and kill the same message.
  The message is keeped ."
- si TOKILL = 1 : appel de DELMESS(WORA)
- si TOKEEP = TOKILL = 0 : appel WRITMESS(LOGIN, WORA, FBD)

4. Entrees .

4.1. Fichiers

- (LOGIN).WHEAD ou (LOGIN).AHEAD
- (LOGIN).WINDEX ou (LOGIN).AINDEX
- (LOGIN).BODY

4.2. Variables globales

- FHD
- FBD
- FDI

4.3. Parametres

- WORA

5. Sorties.

5.1. Fichiers

- (LOGIN).WHEAD ou (LOGIN).AHEAD
- (LOGIN).WINDEX ou (LOGIN).AINDEX
- (LOGIN).BODY

5.2. Variables globales

- FHD
- FBD
- FDI

5.3. Parametres

-

6. Programmes appelants .

- TRTMESS

TYPEHEAD
---

1. But .

Affichage du header .

2. Fonction .

- afficher le numero
- afficher le nom de l'emetteur
- afficher la date d'envoi
- afficher si le message est urgent ou important
- afficher le sujet

3. Algorithme .

Voir 2

4. Entrees .

4.1. Fichiers

-

4.2. Variables globales

- HEADER.DATHEUR
- HEADER.SENDER
- HEADER.SUBJECT
- HEADER.NBER
- HEADER.IORU

4.3. Parametres

-

5. Sorties.

-

6. Programmes appelants .

- TYPEMESS

TYPEBODY
---

1. But .

Affichage du corps du message courant .

2. Fonction .

Voir 1 .

3. Algorithme .

- se positonner ds le fichier (FBD) au debut du corps de message dont l'indice par rapport au debut de (FBD) est HEADER.PTBODY .
- NBLOC = HEADER.LMESS / longueur d'un bloc physique + 1
- repeter NBLOC fois :
  - lire un bloc physique ds (FBD)
  - afficher le bloc a l'ecran

4. Entrees .

4.1. Fichiers

- (LOGIN).BODY

4.2. Variables globales

- FBD
- HEADER.PTBODY
- HEADER.LMESS

4.3. Parametres

-

5. Sorties.

-

6. Programmes appelants .

- TYPEMESS

EXECSHELL

1. But

   Executer une commande du shell .

2. Fonction

   - appeler le programme systeme SH

3. Algorithme .

   Voir 2

4. Entrees

   -

5. Sorties

   -

6. Programmes appelants .

   - GETCMD1
   - GETCMD2

PRINT

1. But .

   Imprimer le corps du message courant .

2. Fonction .

   - Voir 1.

3. Algorithme .

   - se positionner ds le fichier (FBD) au debut du message
     dont l'indice par rapport au debut de (FBD) est HEADER.PTBODY
   - appel de MAKETEMP
   - appel de TRF(FBD,HEADER.LMESS,WORK)
   - appel du programme systeme PRN(WORK)

4. Entrees .

   4.1. Fichiers

- (LOGIN) BODY

4.2. Variables globales

- HEADER.LMESS
- HEADER.PTBODY
- FBD

4.3. Parametres

-

5. Sorties.

-

6. Programmes appelants .

- GETCMD2

SAVE
---

1. But .

Sauver le contenu du fichier (FD) dont la longueur est LMESS dans le fichier dont le nom est demande a l'utilisateur

2. Fonction .

Voir 1 .

3. Algorithme .

- afficher : "File name (< 15 char) ? "
- ranger la reponse ds NAME
- appel de TRF(FD,LMESS,NAME)

4. Entrees .

4.1. Fichiers

- (FD)

4.2. Variables globales

-

4.3. Parametres

- FD
- LMESS

5. Sorties.

5.1. Fichiers

- (NAME)

5.2. Variables globales

-

5.3. Parametres

-

6. Programmes appelants .

- GETCMD2
- SEND

FORWARD
---

1. But .

Faire suivre un message lu par une commande TYPE .

2. Fonction .

- demander a l'utilisateur vers qui faire suivre le message
- transferer le message vers les fichiers de messages en attente des utilisateurs indiques par l'utilisateur (si l'utilisateur est inconnu du courrier , afficher un message d'erreur)

3. Algorithme .

- afficher : "To : "
- ranger la reponse ds DEST
- appel de SENDLST(DEST)

4. Entrees .

4.1. Fichiers

- (LOGIN).BODY
- (LOGIN).WHEAD
- (LOGIN).AHEAD

4.2. Variables globales

-

4.3. Parametres

-

5. Sorties.

5.1. Fichiers

- les fichiers de messages en attente du(des) destinataire(s)

5.2 Variables globales

- DEST

5.3. Parametres

-

6. Programmes appelants .

- GETCMD2

REPLY
---

1. But .

Reponse a un message lu lors du TYPE .

2. Fonction .

- sauver HEADER et INDEX
- enregistrer ds un fichier de travail le corps de message tape par l'utilisateur
- si l'utilisateur le desire , lui permettre d'utiliser l'editeur
- demander a l'utilisateur s'il est d'accord pour envoyer le message :
  si oui :
  - obtenir - la date d'envoi
    - le numero du message
    - le nom d'emetteur
    (renseignements ne necessitant pas l'aide de l'utilisateur)
  - indiquer dans la zone "subject" qu'il s'agit d'une reponse a un message
  - transferer le message(corps + header + index) dans les fichiers de messages en attente de l'emetteur du message courant le message ds son fichier des messages archives
- si l'utilisateur le desire , sauver le corps de message ds un fichier de travail

3. Algorithme .

- transfert de HEADER dans SAVEHEADER
- transfert de INDEX dans SAVINDEX

- appel de GETBODY
- afficher : "Ok to send (y/n) ?"
  si reponse = 'y' :
    - appel de DATNUM
    - garnir HEADER.SUBJECT avec :
      "Reply to message (SAVEHEAD.NBER)"
    - appel de SENDLST(SAVEHEAD.SENDER)
- afficher : "Save the body in a work file (y/n) ?"
  si reponse = 'y' : appel de SAVE(FDWRK,HEADER.LMESS)
- detruire le fichier de travail contenant le message
- restorer SAVEHEAD ds HEADER
- restorer SAVINDEX ds INDEX

4. Entrees .

4.1. Fichiers

- (LOGIN).BODY
- (LOGIN).WHEAD
- (LOGIN).WINDEX
- NBER.LAST

4.2. Variables globales

- HEADER
- INDEX
- LOGIN

4.3. Parametres

-

5. Sorties.

5.1. Fichiers

- NBER.LAST
- les fichiers de messages en attente de l'emetteur du message courant

5.2. Variables globales

- HEADER
- INDEX
- LOGIN

5.3. Parametres

-

6. Programmes appelants .

- GETCMD2

GETCMD2

---

1. But .

Quelle operation l'utilisateur desire-t-il realiser sur le message courant ?

2. Fonction .

- lire la commande tapee par l'utilisateur
- si la commande tapee est syntaxiquement correcte :
  executer cette commande
- si la commande est erronee : afficher un message d'erreur

3. Algorithme .

- appel de GETLINE
- si LINE = "print" appel de PRINT
- si LINE = "prkill" . - appel de PRINT
  - TOKILL = 1
- si LINE = "save" appel de SAVE(FDB, HEADER.LMESS)
- si LINE = "keep" TOKEEP = 1
- si LINE = "forward" et WORA = 'w'
  appel de FORWARD
- si LINE = "reply" et WORA = 'w'
  appel de REPLY
- si LINE = "!" appel de SHELL
- si LINE = CR retourner la valeur 0
- sinon : afficher : "Incorrect command ."
- retourner la valeur 1

4. Entrees .

4.1. Fichiers

- (LOGIN).BODY
- (LOGIN).AHEAD ou (LOGIN).WHEAD
- (LOGIN).AINDEX ou (LOGIN).WINDEX

4.2. Variables globales

- FBD
- FHD
- FDI
- HEADER
- LINE

4.3. Parametres

- WORA
- TOKEEP
- TOKILL

5. Sorties.

5.1. Fichiers

-

5.2. Variables globales

-

5.3. Parametres

- TOKILL
- TOKEEP
- valeur retournee : 0 si l'utilisateur a tape sur CR
  1 sinon

6. Programmes appelants .

- TYPEMESS

GETBODY
---

1. But .

Lecture a l'ecran du corps de message tape par l'utilisateur et stockage dans un fichier temporaire .

2. Fonction .

- indiquer a l'utilisateur qu'il peut taper le corps de message
- lire a l'ecran le message tape par l'utilisateur et rangement dans un fichier de travail
- si l'utilisateur le desire lui permettre d'utiliser l'editeur
- calcul de la longueur du message

3. Algorithme .

- afficher " *"
- creer un fichier de travail
- tant que l'utilisateur n'a pas tape un '.' suivi de CR au debut d'une ligne :
  - ranger le caractere lu dans le fichier de travail
- afficher : "Do you want to use ED (y/n) ?"
- si reponse = 'y' : appel du programme du systeme ED
- appel du prog systeme STAT et rangement de la valeur retournee dans HEADER.LMESS

4. Entrees .

-

5. Sorties.

5.1. Fichiers

- fichier de travail contenant le message .

5.2. Variables globales

- FDWRK
- HEADER.LMESS

5.3. Parametres

6. Programmes appelants .

- SEND
- REPLY

DATNUM
---

1. But .

Obtention du numero , de la date d'envoi et du nom d'emetteur du message .

2. Fonction .

Voir 1 .

3. Algorithme

- Appel de la routine systeme TIME avec transfert du resultat dans HEADER.DATHEUR .
- lecture du fichier NBER.LAST dans HEADER.NBER
- incrementer HEADER.NBER de 1
- reecrire HEADER.NBER dans NBER.LAST
- transferer LOGIN dans HEADER.SENDER

4. Entrees .

4.1. Fichiers

- NBER.LAST

4.2. Variables globales

- LOGIN

4.3. Parametres

-

5. Sorties.

5.1. Fichiers

- NBER.LAST

5.2. Variables globales

- HEADER.DATHEUR
- HEADER.NBER
- HEADER.SENDER

5.3. Parametres

-

6. Programmes appelants .

- SEND
- REPLY

DIALOG
------

1. But .

Demande a l'utilisateur des renseignements necessaires a l'envo
d'un message .

2. Fonction .

- Quel(s) est(sont) le(s) destinataire(s) ?
- Quelle(s) est(sont) la(les) CC ?
- Inserer les noms des destinataires et des carbon copies
  en debut du fichier temporaire contenant le message
- Quel est le sujet du message ?
- Message Urgent ou Important ?
- Message Secret ?

3. Algorithme .

- afficher : "To : "
- ranger la reponse dans DEST
- ecrire DEST dans (FDWRK)
- afficher : "Cc : "
- ranger la reponse dans CC
- ecrire CC dans (FDWRK)
- tant que 'reponse incorrecte' :
  - afficher : "Subject : "
  - si l'utilisateur a tape plus de 50 caracteres :
    'reponse incorrecte'
  - si l'utilisateur a tape moins de 50 caracteres :
    - ranger la reponse ds HEADER.SUBJECT
    - 'reponse correcte'
- afficher "Urgent or Important (i/u) ?"
- ranger le caractere tape par l'utilisateur ds HEADER.IORU
- afficher "Secret message (y/n) ? "
- ranger le caractere tape par l'utilisateur ds HEADER.SECRET

4. Entrees .

4.1. Fichier

- (FDWRK)

4.2. Variables globales

- FDWRK

4.3. Parametres

-

5. Sorties.

5.1. Fichiers

- (FDWRK)

5.2. Variables globales

- HEADER.IORU
- HEADER.SECRET
- HEADER.SUBJECT
- DEST
- CC
- FDWRK

5.3. Parametres

-

6. Programmes appelants .

- SEND

SENDLST
---

1. But .

Transfert du message dans les fichiers de message en attente du(des) destinataire(s) dont le(s) nom(s) se trouve(nt) dans la liste LST .

2. Fonction .

Pour chaque nom appartenant a LST :
- si le nom est celui d'un utilisateur :
  transfert du message(body + header + index) dans les fichiers de messages en attente dont le prefixe est donne ds LST (pour structure des noms de fichiers de messages , voir description des informations)
- si le nom est un nom de liste : realiser ce qui est

ecrit au paragraphe superieur pour tous les membres de la liste
- si le nom est inconnu pour le courrier : afficher un message d'erreur

3. Algorithme .

tant que non fin de LST :
- tant que caractere courant dans LST est alphabetique:
  - ranger le caractere dans NAME
  - passer au caractere suivant de LST
- appel de WRITMESS(NAME,'w',FBD)
- si valeur retournee par WRITMESS est 0 :
  - appel de TOLST(NAME)
  - si valeur retournee par TOLST est 0 :
    afficher"(NAME) unknown . "

4. Entrees .

4.1. Fichiers

-

4.2. Variables globales

-

4.3. Parametres

- LST

5. Sorties.

5.1. Fichiers

- les fichiers de messages (body , head , index) des destinataires

5.2. Variables globales

-

5.3. Parametres

-

6. Programmes appelants .

- SEND
- REPLY

TOLST
---

1. But .

Transfert du message dans les fichiers de messages en attente des membres de la liste NAME .

2. Fonction .

Le fichier (NAME).LST existe-t-il ?
- si non , retourner un indicateur d'erreur
- si oui : pour chaque nom de (NAME).LST :
  - transfert du message ds les fichiers des messages en attente de l'utilisateur

3. Algorithme .

- ouvrir le fichier(NAME).LST
- si impossible : retourner une valeur nulle
- sinon :
  - tant que non EOF :
    - lire un nom de (NAME).LST ds DEST
    - appel de WRITMESS(DEST,'w',FDO)
    - si valeur retournee est 0 :
      afficher : "(DEST) unknomwn ."
  - retourner la valeur 1

4. Entrees .

4.1. Fichiers

-

4.2. Variables globales

-

4.3. Parametres

- NAME

5. Sorties.

5.1. Fichiers

- les fichiers de messages en attente des utilisateurs existants ds (NAME).LST

5.2. Variables globales

-

5.3. Parametres

Valeur retournee : 1 si la liste NAME existe
0 sinon

6. Programmes appelants .

- SENDLST

- Niveau 4 -

GETLINE
----

1. But .

Lecture a l'ecran de la commande tapee par l'utilisateur .

2. Fonction .

Voir 1 .

3. Algorithme .

- Afficher : "* "
- tant que caractere lu au terminal pas = CR :
  - transferer le caractere ds LINE
- faire pointer PTLINE vers LINE

4. Entrees .

-

5. Sorties.

5.1. Fichiers

-

5.2. Variables globales

- PTLINE
- LINE

5.3. Parametres

-

6. Programmes appelants .

- GETCMD1
- GETCMD2

GETDATE
----

1. But .

Analyse de la date pointee par PTLINE sous la forme ASCII JJ:MM:AA et modification de cette date sous la forme numerique AAMMJJ dans DATE .

2. Fonction .

Voir 1 .

3. Algorithme .

- tant que non fin de LINE
- lecture de LINE jusqu'au prochain separateur :
  - si separateur pas = ':' : retourner valeur 0
  - sinon :
    - transformer la chaine de caracteres en numerique dans DATE
    - si DATE > 31 : retourner valeur 0
- lecture de LINE jusqu'au prochain separateur :
  - si separateur pas = ':' : retourner valeur 0
  - sinon :
    - transformer la chaine lue en INTEGER ds DATE
    - si DATE > 12 : retourner valeur 0
- lecture de LINE jusqu' a la fin
  - transformer la chaine lue en INTEGER ds DATE
  - si DATE > 99 : retourner valeur 0
- retourner valeur 1

4. Entrees .

4.1. Fichiers

-

4.2. Variables globales

- PTLINE
- LINE

4.3. Parametres

-

5. Sorties.

5.1. Fichiers

-

5.2. Variables globales

-

5.3. Parametres

- DATE
- valeur retournee : 0 si date incorrecte
  1 si date correcte

6. Programmes appelants .

- BEFORE
- AFTER

WRITCONF
---

1. But .

Ecrire une confirmation dans le fichier CONF de l'emetteur du message courant .

2. Fonction .

- transfert de HEADER.NBER dans CONFIRM.NBER
  transfert de LOGIN dans CONFIRM.DEST
  transfert de HEADER.DATHEUR dans CONFIRM.DATHEUR
  transfert de HEADER.SUBJECT dans CONFIRM.SUBJECT
- ecrire CONFIRM dans le fichier (HEADER.SENDER).CONF

3. Algorithme .

Voir 2 .

4. Entrees .

4.1. Fichiers

-

4.2. Variables globales

- HEADER.NBER
- HEADER.DATHEUR
- HEADER.SUBJECT
- HEADER.SENDER
- LOGIN

4.3. Parametres

-

5. Sorties.

5.1. Fichiers

- (HEADER.SENDER).CONF

5.2. Variables globales

- CONFIRM

5.3. Parametres

-

6. Programmes appelants .

- TYPEMESS

WRITMESS
----

1. But .

Ecriture d'un message dans les fichiers BODY HEAD INDEX dont le prefixe du nom est NAME .

2. Fonction .

- si WORA = 'w' : constitution des noms de fichier (NAME).BODY (NAME).WHEAD (NAME).WINDEX
  sinon : constitution des noms de fichier (NAME).BODY (NAME).AHEAD (NAME).AINDEX
- tester si les fichiers existent et s'il y a encore assez de place :
  si oui : inserer le corps de message etant ds (FDO) ds le fichier (NAME).BODY
  inserer HEADER ds (NAME).WHEAD ou (NAME).AHEAD
  creer l'index correspondant au message et l'
  inserer ds (NAME).WINDEX ou (NAME).AINDEX
  si non : renvoyer un indicateur d'erreur

3. Algorithme .

- si WORA = 'w' : - appel de MAKEWAIT(NAME)
  - appel de WRITBODY(NAME)
  si valeur retournee est 0 ou -1 : retou
  au prog appelant avec la meme valeur
- si WORA = 'a' : - appel de MAKEARCH(NAME)
  - NBLOC = HEADER.LMESS / longueur d'un bloc +
  - appel de MAKEBUSY(AFHEAD, NBLOC, HEADER.PTBOD
- appel de WRITHEAD(WORA)
- appel de WRITIND(WORA)
- retourner la valeur 1

4. Entrees .

4.1. Fichiers

- (FDO)

4.2. Variables globales

- HEADER

4.3. Parametres

- NAME
- WORA
- FDO

5. Sorties.

5.1. Fichiers

- (NAME).BODY
- (NAME).FBODY
- (NAME).WHEAD ou (NAME).AHEAD
- (NAME).AWHEAD ou (NAME).WFHEAD
- (NAME).AINDEX ou (NAME).WINDEX

5.2. Variables globales

-

5.3. Parametres

Valeur retournee : 1 si les ficheirs existent
0 si si les fichiers n'existent pas
-1 si plus de place

6. Programmes appelants .

- TYPEMESS
- SEND
- SENDLST
- TOLST

WRITBODY
---

1. But .

Ecriture du corps de message se trouvant dans le fichier (FDO) dans le fichier dont le nom est donne dans WBODY(si WORA = 'w') ou dans ABODY (si WORA = 'a')

2. Fonction .

Voir 1

3. Algorithme .

- calcul de NBLOC = HEADER.LMESS / ln d'un bloc physique + 1
- si WORA = 'w' : appel SEEKFREE(WFBODY,NBLOC)
- si WORA = 'a' : appel SEEKFREE(AFBODY,NBLOC)
- si valeur retournee par SEEKFREE = 0 ou -1 : retourner la meme valeur au programme appelant

- se positionner sur le bloc libre dans le fichier BODY (valeur retournee par SEEKFREE * ln d'un bloc physique)
- repeter NBLOC fois :
  - lire un bloc ds (FDO)
  - transferer le bloc lu ds le fichier dont le nom est donne ds ABODY(si WORA = 'a') ou ds WBODY(si WORA = 'w')

3. Algorithme .

Voir 2 .

4. Entrees .

4.1. Fichiers

- (FDO)

4.2. Variables globales

- HEADER.LMESS
- WBODY ou ABODY

4.3. Parametres

- WORA
- FDO

5. Sorties.

5.1. Fichiers

- (NAME).BODY
- (NAME).FBODY

5.2. Variables globales

- HEADER.PTBODY

5.3. Parametres

- valeur retournee : 1 si ok
  0 si fichier n'existe pas
  -1 si plus de place

6. Programmes appelants .

- WRITMESS

WRITHEAD
---

1. But .

Ecriture de HEADER dans le fichier dont le nom se trouve dans AHEAD (si WORA = 'a') ou dans WHEAD (si WORA = 'w')

2. Fonction .

Voir 1

3. Algorithme .

- si WORA = 'w' : - ouverture du fichier WHEAD
  - appel SEEKFREE(WFHEAD,1)
- si WORA = 'a' : - ouverture du fichier AHEAD

- appel SEEKFREE(AFHEAD, 1)
- se positionner sur le bloc libre dans le fichier HEAD
- ecriture de HEADER dans le fichier HEAD

4. Entrees .

4.1. Fichiers

-

4.2. Variables globales

- (WHEAD et WFHEAD) ou (AHEAD et AFHEAD)
- HEADER

4.3. Parametres

- WORA

5. Sorties.

5.1. Fichiers

- ((NAME).WHEAD et (NAME).WFHEAD)

ou

((NAME).AHEAD et (NAME).AFHEAD)

5.2. Variables globales

- INDEX.PTHEAD

5.3. Parametres

-

6. Programmes appelants .

- WRITMESS

WRITIND
---

1. But .

Ecriture de l'index associe a un message dans le fichier dont le nom se trouve dans AINDEX (si WORA = 'w) ou dans WINDEX (si WORA = 'a')

2. Fonction .

Voir 1

3. Algorithme .

- si WORA ='w' : ouverture du fichier (NAME).WINDEX

- si WORA = 'a' : ouverture du fichier (NAME).AINDEX
- remplissage des index INDEX.NBRE et INDEX.DATE
- tant que 'non insere' et non EOF :
    - lire un article de (NAME).WINDEX
    - si numero de message lu > INDEX.NBRE :
        - inserer INDEX ds (NAME).WINDEX ou (NAME).AINDEX
        - indiquer 'insere'
- si 'non insere' : inserer INDEX ds (NAME).WINDEX ou (NAME).AINDEX

4. Entrees .

4.1. Fichiers

-

4.2. Variables globales

- WINDEX ou AINDEX
- HEADER.NBER
- HEADER.DATHEUR

4.3. Parametres

- WORA

5. Sorties.

5.1. Fichiers

- (NAME).WINDEX ou (NAME).AINDEX

5.2. Variables globales

- INDEX

5.3. Parametres

-

6. Programmes appelants .

- WRITMESS

DELMESS
---

1. But .

Destruction d'un message .

2. Fonction .

- destruction de l'index

- indiquer le bloc pointe par INDEX.PTHEAD comme etant libre
- indiquer les blocs pointes par HEADER.PTBODY comme etant libr

3. Algorithme .

- appel DELINDEX(WORA)
- calcul de nbloc = HEADER.LMESS/ln d'un bloc physique + 1
- si WORA = 'w' : appel MAKEFREE(WFHEAD, 1, INDEX.PTHEAD)
  appel MAKEFREE(WFBODY, NBLOC, HEADER.PTBODY)
- si WORA = 'a' : appel MAKEFREE(AFHEAD, 1, INDEX.PTHEAD)
  appel MAKEFREE(AFBODY, NBLOC, HEADER.PTBODY)

4. Entrees .

4.1. Fichiers

- (LOGIN).FBODY
- (LOGIN).AFHEAD ou (LOGIN).WFHEAD
- (LOGIN).AINDEX ou (LOGIN).WINDEX

4.2. Variables globales

- HEADER.LMESS
- HEADER.PTBODY
- INDEX.PTHEAD

4.3. Parametres

- WORA

5. Sorties.

5.1. Fihiers

- (LOGIN).FBODY
- (LOGIN).AFHEAD ou (LOGIN).WFHEAD
- (LOGIN).AINDEX ou (LOGIN).WINDEX

5.2. Variables globales

-

5.3. Parametres

-

6. Programmes appelants .

- GETCMD2

DELINDEX
---

1. But .

Destruction de l'index se trouvant ds le fichier (LOGIN).WINDEX (si WORA = 'w') ou (LOGIN).AINDEX (si WORA = 'a') et dont l'identifiant INDEX.NBRE = HEADER.NBER

2. Fonction .

Voir 1

3. Algorithme .

- Lire le premier article du fichier dindex ds CPTDEL
- Tant que INDEX.NBRE pas = HEADER.NBER :
  - lire un article du fichier d'index ds INDEX
- Reecriture de INDEX ds le fichier d'index.
- incrementer CPTDEL de 1
- si CPTDEL > 10 : operer un tassement :
  tant que non EOF :
  - lire un index du fichier ds INDEX
  - si INDEX.NBRE > 0 : reecrire INDEX ds le fichier

4. Entrees .

4.1. Fichiers

- (LOGIN).WINDEX ou (LOGIN).AINDEX

4.2. Variables globales

- HEADER.NBER
- WINDEX ou AINDEX

4.3. Parametres

- WORA

5. Sorties.

5.1. Fichiers

- (LOGIN).WINDEX ou (LOGIN).AINDEX

5.2. Variables globales

-

5.3. Parametres

-

6. Programmes appelants .

- DELMESS

MAKEFREE
----

1. But .

Liberer NBLOC blocs dans le fichier dont la table des blocs se trouve ds le fichier de nom FILE

2. Fonction .

3. Algorithme .

- a partir de la valeur d'INDICE , calcul de la position du premier bit a mettre a 0 :
  - position du byte = INDICE / 8 (/ = division entiere)
  - position du bit = INDICE % 8 (% = reste de la divisior entiere)
- lecture de FILE dans TAB
- se positionner sur le bon byte et le bon bit dans TAB
- repeter NBLOC fois :
  - mettre le bit courant a 0
  - passer au bit suivant

4. Entrees .

4.1. Fichiers

- fichier dont le nom est ds NFILE

4.2. Variables globales

-

4.3. Parametres

- FILE
- NBLOC
- INDICE

5. Sorties.

5.1. Fichiers

- fichier dont le nom est ds FILE

5.2. Variables globales

-

5.3. Parametres

-

6. Programmes appelants .

- DELMESS

MAKEBUSY
---

1. But .

Reserver NBLOC blocs dans le fichier dont la table des blocs se trouve ds le fichier de nom FILE

2. Fonction .

Voir 1

3. Algorithme .

- a partir de la valeur d'INDICE , calcul de la position du premier bit a mettre a 1 :
  - position du byte = INDICE / 8 (/ = division entiere)
  - position du bit = INDICE % 8 (% = reste de la divisio entiere)
- lecture de FILE dans TAB
- se positionner sur le bon byte et le bon bit dans TAB
- repeter NBLOC fois :
  - mettre le bit courant a 1
  - passer au bit suivant

4. Entrees .

4.1. Fichiers

- fichier dont le nom est ds NFILE

4.2. Variables globales

-

4.3. Parametres

- FILE
- NBLOC
- INDICE

5. Sorties.

5.1. Fichiers

- fichier dont le nom est ds FILE

5.2. Variables globales

-

5.3. Parametres

-

6. Programmes appelants .

- WRITMESS

SEEKFREE
---

1. But .

Rechercher NBLOC blocs libres ds un fichier dont la table des blocs est ds NFILE

2. Fonction .

Voir 1 .

3. Algorithme .

- si NBLOC = 0 : retourner la valeur -2
- Lecture de FILE dans TAB
- Balayage de TAB bit par bit jusqu'a ce que NBLOC bits consecutifs a 0 aient ete trouves .
- si apres balayage de la table , le nbre de bits consecutifs trouves a 0 est < NBLOC : retourner la valeur -1
  sinon : - mettre les bits a 1
  - retourner comme valeur la position du premier bit a 0 (= 8*numero du byte + numero du bit)

4. Entrees .

4.1. Fichiers

- fichier dont le nom est dans FILE

4.2. Variables globales

-

4.3. Parametres

- FILE
- NBLOC

5. Sorties.

5.1. Fichiers

- fichier dont le nom est dans FILE

5.2. Variables globales

-

5.3. Parametres

valeur retournee : -2 si NBLOC == 0

-1 si plus de place
position du premier bloc libre sinon

6. Programmes appelants .

- WRITBODY
- WRITHEAD

MAKEWAIT
---

1. But .

Composition du nom des fichiers de messages en attente , nom dont le prefixe est NAME .

2. Fonction .

- composition de :
  - WBODY
  - WFBODY
  - WHEAD
  - WFHEAD
  - WINDEX

3. Algorithme .

Voir 2

4. Entrees .

4.1. Fichiers

-

4.2. Variables globales

-

4.3. Parametres

- NAME

5. Sorties.

5.1. Fichiers

-

5.2. Variables globales

- WBODY
- WFBODY
- WHEAD
- WFHEAD

- WINDEX

5.3. Parametres

-

6. Programmes appelants .

- READMESS

MAKEARCH

---

1. But .

Composition du nom des fichiers de messages archives , nom dont le prefixe est NAME .

2. Fonction .

- composition de :
  - ABODY
  - AFBODY
  - AHEAD
  - AFHEAD
  - AINDEX

3. Algorithme .

Voir 2

4. Entrees .

4.1. Fichiers

-

4.2. Variables globales

-

4.3. Parametres

- NAME

5. Sorties.

5.1. Fichiers

-

5.2. Variables globales

- ABODY
- AFBODY

- AHEAD
- AFHEAD
- AINDEX

5.3. Parametres

-

6. Programmes appelants .

- READMESS

MAKETEMP

1. But .

Composer le nom d'un fichier de travail de telle sorte qu'il soit unique

2. Fonction .

Voir 1

3. Algorithme .

- 2 premieres lettres du nom sont CO
- les 5 caracteres suivants sont des chiffres generes aleatoirement

4. Entrees .

-

5. Sorties .

5.1. Fichiers

-

5.2. Variables globales

- WORK

5.3. Parametres

-

6. Programmes appelants

- GETBODY
- WRITIND
- DELINDEX

READLOG
---

1. But .

   Obtention du login name de l'utilisateur .

2. Fonction .

   - Recherche dans le fichier du systeme /etc/passwd de l'entree correspondant a l'utilisateur .
   - Transfert dans LOGIN du login name de l'utilisateur .

3. Algorithme .

   Voir 2

4. Entrees .

   4.1. Fichiers

   - /ETC/PASSWD

   4.2. Variables globales

   -

   4.3. Parametres

   -

5. Sorties.

   5.1. Fichiers

   -

   5.2. Variables globales

   - LOGIN

   5.3. Parametres

   -

6. Programmes appelants .

   - MAIN

OPENWAIT
---

1. But .

   Ouverture des fichiers (LOGIN).BODY (LOGIN).WHEAD (LOGIN).WINDE

2. Fonction .

- ouvrir (LOGIN).BODY ; ranger le file descriptor dans FBD
- ouvrir (LOGIN).WHEAD ; ranger le file descriptor dans FHD
- ouvrir (LOGIN).WINDEX ; ranger le file descriptor dans FDI
  se positionner 2 caracteres apres le debut du fichier (les 2 premiers caracteres sont reserves au compteur du nbre de destructions)

3. Algorithme .

Voir 2

4. Entrees .

4.1. Fichiers .

-

4.2. Variables globales

- WBODY
- WHEAD
- WINDEX

4.3. Parametres

-

5. Sorties .

5.1. Fichiers

- (LOGIN).BODY
- (LOGIN).WHEAD
- (LOGIN).WINDEX

5.2. Variables globales

- FBD
- FHD
- FDI

5.3. Parametres

-

6. Programmes appelants .

- READMESS

OPENARCH
---

1. But .

Ouverture des fichiers (LOGIN).BODY (LOGIN).AHEAD (LOGIN).AINDE

2. Fonction .

- ouvrir le fichier (LOGIN).ABODY ; ranger le file descriptor dans FBD

- ouvrir le fichier (LOGIN).AHEAD ; ranger le file descriptor dans FHD
- ouvrir le fichier (LOGIN).AINDEX ; ranger le file descriptor dans FDI

se positionner 2 caracteres apres le debut du fichier

3. Algorithme .

Voir 2

4. Entrees .

4.1. Fichiers

-

4.1. Variables globales

- ABODY
- AHEAD
- AINDEX

4.3. Parametres

-

5. Sorties .

5.1. Fichiers

- (LOGIN).BODY
- (LOGIN).AHEAD
- (LOGIN).AINDEX

5.2. Variables globales

- FDI
- FDH
- FBD

5.3. Parametres

-

6. Programmes appelants

- READMESS

COMPAR
---

1. But .

Comparer 2 chaines de caracteres ; la premiere est adressee par PT1 , la deuxieme par PT2 . La deuxieme chaine est terminee par ':' .

2. Fonction .

- tant que le contenu de PT1 = le contenu de PT2
  - incrementer PT1 et PT2
  - si PT2 pointe vers ':' : retourner la valeur 0
- si PT1 pointe vers une valeur plus grande que PT2 : retourner la valeur 1
- si PT1 pointe vers une valeur plus petite que PT2 : retourner la valeur -1

3. Algorithme .

Voir 2

4. Entrees .

4.1. Fichiers .

-

4.2. Variables globales

-

4.3. Parametres

- PT1
- PT2

5. Sorties .

5.1. Fichiers

-

5.2. Variables globales

-

5.3. Parametres

Valeur retournee :
0 si les 2 chaines sont egales
1 si le contenu de la premiere est > que le contenu de la seconde

-1 si le contenu de la premiere est < que le
contenu de la seconde

6. Programmes appelants .

- GETCMD1
- GETCMD2
- READMESS

GETMBER
---

1. But .

Recherche de l'entree ds l'entree du repertoire dont
le login name = LOG

2. Fonction .

- tant que non EOF et 'pas trouve' :
  - lire un article de USER.LST dans MEMBER
  - appel de COMPAR(MEMBER.LOG,LOG)
    si egalite : indiquer 'trouve'
- si 'trouve' : retourner la valeur 1
  sinon retourner la valeur 0

3. Algorithme .

Voir 2

4. Entrees

4.1. Fichiers

- USER.LST

4.2. Variables globales

-

4.3. Parametres

- LOG

5. Sorties

5.1. Fichiers

-

5.2. Variables globales

- MEMBER.LOG
  - .NAME
  - .PASS

5.3. Parametres

Valeur retournee : 0 si trouve
1 si pas trouve

6. Programmes appelants

- SECRETMESS

A.3 PROGRAMMES.

```
/*
*
*
*       Variables globales du systeme
*
*
*/

/* buffers pour les fichiers message */

struct header {
                int ptbody;             /* pointeur vers le corps */
                int lmess;              /* longueur du corps */
                int nber;               /* numero du message */
                int datheur[2];         /* date & heure d'envoi */
                char sender[9];         /* emetteur du message */
                char ioru;              /* important ou urgent ? */
                char secret;            /* secret ? */
                char subject[51];       /* sujet du message */
                 } bufhead;


struct index {
                int nbre;               /* numero du message */
                char date[3];           /* date sous la forme aammjj */
                int pthead;             /* pointeur vers le header */
              } bindex;



struct member {
                char log[9];            /* login name */
                char name[25];          /* patronyme */
                char pass[8];
                char nl;                /* new line */
                } bmemb;


/*      file descriptors        */
```

```
int fdi;                    /* fd du fichier d'index utilise dans READMESS */
int fhd;                    /* fd fu fichier header utilise ds READMESS */
int fbd ;                   /* fd du fichier body ds READMESS */
int fdl;                    /* fd du verrou associe aux fichiers utilises ds READMESS */
int fdwrk;                          /* fd du fichier contenant
                                       le message tape */


/*      ligne de commande       */

char line[100];
char *ptline;               /* pointeur vers char courant ds LINE  */


/*      login name de l'utilisateur courant     */

char login[9];


/*      Liste des destinataires et des carbon copies ;
        utilise lors de l'envoi d'un message
*/

char dest[150];                     /* destinataires */
char cc[150];                       /* carbon copies */


/* noms des fichiers message */

char wbody[] "/ss/mail/xxxxxx.body";            /* corps de message */
char abody[] "/ss/mail/xxxxxx.body";            /* corps de message */
char wfbody[] "/ss/mail/xxxxxx.fbody";          /* table des blocs libres dans body */
char afbody[] "/ss/mail/xxxxxx.fbody";          /* table des blocs libres dans body */
char whead[] "/ss/mail/xxxxxx.whead";           /* headers en attente */
char wfhead[] "/ss/mail/xxxxxx.fwhead";         /* table des blocs libres dans whead */
char windex[] "/ss/mail/xxxxxx.windex";         /* index en attente */
```

```
char ahead[] "/ss/mail/xxxxxx.ahead";          /* headers archives */
char afhead[] "/ss/mail/xxxxxx.fahead";        /* table des blocs libres dans ahead */
char aindex[] "/ss/mail/xxxxxx.aindex";        /* index archives */



char lock[] "/ss/mail/xxxxxx.lock";            /* semaphore */

char work[] "/ss/mail/coxxxxx";                /* fichier de travail pour :
                                               l'envoi d'un message */
char wrkind[] "/ss/mail/inxxxxx";              /* fichier de travail pour
                                          insertion d'un index */

char conf[] "/ss/mail/xxxxxx.conf";            /* confirmations */
char lockconf[] "/ss/mail/xxxxxx.lock";        /* verrou pour les conf */


/* chaine pour remplissage de bufhead.subject lors d'un REPLY */

char suj1[] "Reply to message ";


/* variable pour nom de liste */

char namlst[] "/ss/mail/xxxxxxxxxxxxx";
```

```
/**************************************************************************
*                                                                        *
*                                                                        *
*                                                                        *
*                                                                        *
*                Courrier    Electronique                                *
*                                                                        *
*                                                                        *
*                                                                        *
*                                                                        *
**************************************************************************/
```

```
#include "global.h"
#define LCONF    15                 /* Longueur dune confirmation */
#define LMEMB    43                 /* Longueur d'une entree ds USER.LST */
#define LBLOC    100                /* Longueur d'un bloc ds un fichier body */
#define LHEAD    72                 /* longueur d'un header */
#define LINDEX   7                  /* longueur d'un index */
#define LTAB     100                /* longueur de tab */

#define ECHO     010                /* <==> ne pas afficher le char tape */
#define ZERO     00



main()
        {
        readlog();
        confwait();
        while(getcmd1());
        }


/*
*
*
*
*        Cette routine lit la ligne de commande tapee par l'utilisateur ;
*       selon la commande tapee , appel d'une des fonctions principales .
*
*
*
*/


getcmd1()
        {
        printf("#new goodbye user ! type send search list rmlist exit ?#");
        getline();
        if(compar(line,"type:") == 0) readmess('w');
        else
        if(compar(line,"send:") == 0) send();
        else
        if(compar(line,"search:") == 0) readmess('a');
```

```
        else
        if(compar(line,"user:") == 0) getuser();
        else
        if(compar(line,"!:") == 0) shell();
        else
        if(compar(line,"list:") == 0) creatlst();
        else
        if(compar(line,"rmlist:") == 0) deletlst();
        else
        if(compar(line,"goodbye:") == 0) delmber();
        else
        if(compar(line,"new:") == 0) newmber();
        else
        if(compar(line,"exit:") == 0) return(0);
        else
        printf("incorrect command .#");
        return(1);
        }



/*
*
*
*       Demander a l'utilisateur quelle commande il veut
*       executer sur le message courant
*
*
*/


getcmd2(wora,ptkk)
char wora;
char *ptkk;                 /* pointe vers kk (voir TYPEMESS) */

        {
        printf("#print kill prkill save ! ");
        if(wora == 'w') printf("keep forward reply ");
```

```
printf("<return> ?#");
getline();

if(compar(ptline,"print:") == 0)
        print();
else
if(compar(ptline,"kill:") == 0)
        *ptkk = 'k';
else
if(compar(ptline,"prkill:") == 0)
        {
        print();
        *ptkk = 'k';
        }
else
if(compar(ptline,"save:") == 0)
        {
        seek(fbd,bufhead.ptbody*LBLOC,0);
        save(fbd,bufhead.lmess);
        }
else
if(compar(ptline,"keep:") == 0)
        {
        if (wora == 'w')
                *(ptkk+1) = 'k';
        else
                ptline=- 4;
        }
else
if(compar(ptline,"forward:") == 0)
        {
        if (wora == 'w')
                forward();
        else
                ptline=- 7;
        }
else
if(compar(ptline,"reply:") == 0)
        {
        if (wora == 'w')
                reply();
```

```
                else
                        ptline=- 5;
                }
        else
        if(compar(ptline,"!:") == 0)
                shell();
        else
        if(*ptline == '#') return(0);
        else
        printf("Invalid command .#");
        return(1);
        }

/*
*
*
*        si wora == 'w' : ouverture des fichiers de messages en attente
*        si wora == 'a' : ouverture des fichiers de messages archives
*       selon le type de cle tape ds la ligne de commande :
*               appel d'une des routines de lecture des messages
*
*
*/


readmess(wora)
char wora;

        {
        if(wora == 'w')
                {
                        /* messages en attente */
                makewait(login);
                openwait();
                }
        else
                        /* messages archives */
                {
                makearch(login);
                openarch();
                }
```

```
        if(compar(ptline,"from:") == 0) from(wora);
        else
        if(compar(ptline,"before:") == 0) before(wora);
        else
        if(compar(ptline,"after:") == 0) after(wora);
        else
        if(compar(ptline,"number:") == 0) number(wora);
        else
        if(compar(ptline,"urgent:") == 0) imporurg(wora,'u');
        else
        if(compar(ptline,"important:") == 0) imporurg(wora,'i');
        else
        if(compar(ptline,"#:") == 0) alles(wora);
        else
        if(compar(ptline,"subject:") == 0) subject(wora);
        else
        printf("incorrect key .#");
        closefil();
        return;
        }



/*
*
*
*       Affichage de tous les messages
*
*
*/


alles(wora)
char wora;
        {
        while(readhead(fhd))
                if(bufhead.nber != 0)
                                        trtmess(wora);
        }
```

```
/*
*
*
*         Affichage des messages envoyes par un emetteur donne
*
*
*/


from(wora)
char wora;

        {
        struct header *phead;

        int i;
        char name[50];              /* nom de l'emetteur sur lequel se
                                       fera la recherche */

        for(i=0;((name[i] = *ptline++) != '#') && i < 50; i++)
        if(i == 50)      {
                         printf("unknown :%s",&line[9]);
                         return;
                         }
        phead = &bufhead;
        while(read(fhd,&bufhead,LHEAD) == LHEAD)

                /* si header existe logiquement (bufhead.nber != 0) et si
                    nom de la ligne de commande == nom lu ds (fhd) */

                if((bufhead.nber != 0) && (compar(name,phead->sender) == 0))
                                trtmess(wora);
        }



/*
*
*
*       - si IU = 'i' :affichage des messages importants
*       - si IU = 'u' : affichage des messages urgents
```

```
*
*
*/


imporurg(wora,iu)
char wora;
char iu;

        {
        while(readhead(fhd))
                if((bufhead.ioru == 'i' && iu == 'i') ||
                   (bufhead.ioru == 'u' && iu == 'u'))
                                                        trtmess(wora);
        }


/*
*
*
*        Affichage des messages envoyes avant une date donnee
*
*
*/


before(wora)
char wora;

        {
        char dat[3];                /* aammjj */

        while(*ptline == ' ')ptline++;
        if(! getdate(&dat[2])) {
                               printf("incorrect date .#");
                               return;
                               }
        while(readindex(fdi))
                if(bindex.nbre != 0 && compdate(dat,bindex.date) >= 0)
```

```
                        {
                        readirhead(fhd);
                        trtmess(wora);
                        }
        }



/*
*
*
*       Affichage des messages envoyes apres une date donnee
*
*
*/


after(wora)
char(wora);

        {
        char dat[3];                /* aammjj */

        while(*ptline == ' ')ptline++;
        if(! getdate(&dat[2])) {
                               printf("incorrect date .#");
                               return;
                               }
        while(readindex(fdi))
                {
                if((bindex.nbre != 0 ) && ( compdate(dat,bindex.date) <= 0))
                        {
                        readirhead(fhd);
                        trtmess(wora);
                        }
                }
        }
```

```
/*
*
*
*       affichage des messages dont les numeros sont donnes
*
*
*/


number(wora)
char wora;

        {
        char *ptwrk;            /* pointeur de travail ds la ligne de commande */

        while(*ptline == ' ')ptline++;
        ptwrk = ptline;
        while(*ptwrk >= '0' && *ptwrk <= '9') ptwrk++;
        while(*ptwrk == ' ')ptwrk++;
        if(*ptwrk == ',' || *ptwrk == '#') sequence(wora);
        else
        if(*ptwrk == ':') range(wora);
        else
        printf("error in sequence .#");
        }


/*
*
*
*       Traitement du cas ou les numeros sont donnes sous forme de sequence
*
*
*/


sequence(wora)
char wora;

        {
```

```
        int inum;
        int j;
        int jcount;
        int num[100];

        inum = 0;
        while(*ptline != '#')
                {
                while(*ptline == ' ')ptline++;
                if(inum < 100) num[inum++] = atoi(ptline);
                while(*ptline >= '0' && *ptline <= '9') ptline++;
                while(*ptline == ' ') ptline++;
                if(*ptline != ',' && *ptline != '#')
                                        {
                                        printf("Error in sequence .#");
                                        return;
                                        }

                if(*ptline == ',') ptline++;
                }


                for(j = 0; j < inum; j++)
                for(jcount = 0; (jcount < inum) && (readindex(fdi)) ; )
                        {
                        for(j = 0; (j < inum) && (num[j] != bindex.nbre); j++);
                        if(j <= inum && num[j] == bindex.nbre)
                                        {
                                        readirhead(fhd);
                                        trtmess(wora);
                                        jcount++ ;
                                        }
                        }
}



/*
*
*
*       Traitement du cas ou les numeros sont donnes sous forme de ran
```

```
*
*
*/


range(wora)
char wora;

        {
        int num1,num2;

        num1 = atoi(ptline);                    /* borne inferieure */
        while(*ptline >= '0' && *ptline <= '9') ptline++;
        while(*ptline == ' ' || *ptline == ':') ptline++;
        num2 = atoi(ptline);                    /* borne superieure */
        if(num1 > num2) {
                        printf("range error .#");
                        return;
                        }

        while(readindex(fdi) && bindex.nbre <= num2)
                if(num1 <= bindex.nbre)
                        {
                        readirhead(fhd);
                        trtmess(wora);
                        }
        }



/*
*
*
*        Affichage des messages dont le domaine "subject" contient la
*       chaine de caracteres tapes
*
*
*/
```

```
subject(wora)
char wora;
        {
        struct header *phead;
        char *ptwrk;                    /* pointeur de travail ds LINE */

        int i;
        int ok;


        while(*ptline == ' ') ptline++;
        ptwrk = ptline;
        while(*ptwrk != '#') ptwrk++;
        *ptwrk = ':';
        phead = &bufhead;
        ptwrk = ptline ;
        while(readhead(fhd))
                {
                ok = 0;
                for(i = 0;phead->subject[i] != '#' && !ok; i++)
                        {
                        if(compar(&(phead->subject[i]),ptwrk) == 0)
                                /* si chaine est contenue dans "subject" */
                                {
                                trtmess(wora);
                                ok = 1;
                                }
                        }
                }
        }



/*
*
*
*       Si le message n'est pas secret ou si l'utilisateur a donne son
```

```
*       mot de passe correctement : afficher le message.
*       Sinon afficher message d'erreur
*
*
*/

trtmess(wora)
char wora;
        {
        if(! secretmess())
                typemess(wora);
        else
        printf("Sorry,you are not authorized to read message %d .#",bufhead.nber);
        }



/*
*
*
*       - afficher header et body
*       - demander a l'utilisateur quelles commandes il veut executer sur
*         le message
*
*
*/

typemess(wora)
char(wora);
        {
        char kk[2];                             /* kk[0] = 'k' <==> kill
                                                   kk[1] = 'k' <==> keep */

        typehead();
        typebody();

        /* si le message lu est un message en attente : ecrire une
                confirmation */

        if(wora == 'w') writconf();
```

```
        kk[0] = kk[1] = 0;
        while( getcmd2(wora,kk));
        if((kk[0] == 'k') && (kk[1] == 'k'))
                {
                printf("Stupid to kill and keep the same message .#");
                printf("The message is keeped .#");
                }
        else
        if(kk[0] == 'k') delmess(wora);
        else
        if((kk[0] != 'k') && (kk[1] != 'k'))
                {
                if(wora == 'w')
                        {
                        /* transfert le message du fichier des messages
                           en attente dans le fichier des messages archives
                        */
                        delmess('w');
                        seek(fbd,bufhead.ptbody*LBLOC,0);
                        if(writmess(login,'a',fbd) < 0)
                                printf("No more place in your messages files .#");
                        }
                }
        }


/*
*
*
*       Affichage du header
*
*
*/

typehead()
        {
        int i;

        char *ptime;
        char *ctime();
```

```
        printf("%d ",bufhead.nber);

        printf("From : ");
        for(i = 0;bufhead.sender[i] != ':';i++)
                        putchar(bufhead.sender[i]);
        for(;i < 10;i++)
                        putchar(' ');


        if(bufhead.ioru == 'i') printf("Important ! ");
        else
        if(bufhead.ioru == 'u') printf("Urgent ! ");

        ptime = ctime(bufhead.datheur);
        ptime =+ 4;
        for(i=0;i<12;i++)
                {
                putchar(*ptime);
                ptime++;
                }

        putchar(' ');
        for(i=0;bufhead.subject[i] != '#';i++)
                        putchar(bufhead.subject[i]);
        printf("#");
        }



/*
*
*
*       Afficher le corps du message
*
*
*/

typebody()
        {
        int i;
```

```
        int j;
        int n;
        int nbloc;                  /* nbre de blocs ds le corps */
        char bufbody[LBLOC];

        printf("#");

        /* pointer au debut du corps */

        seek(fbd, bufhead.ptbody * LBLOC, 0);

        nbloc = bufhead.lmess/LBLOC + 1;
        for(i = 0; i < nbloc; i++)
                {
                n = read(fbd, bufbody, LBLOC);
                /* si dernier bloc : lire les caracteres restants */
                if(i == (nbloc-1)) n = bufhead.lmess % LBLOC ;

                /* afficher le nbre de char lus a l'ecran */
                for(j = 0; j < n; j++)
                        putchar(bufbody[j]);
                }
        }



/*
*
*
*        Affichage des headers en attente et des confirmations
*
*
*/


confwait()
        {
        makewait(login);                    /* constituer nom de fichier  CONF */
        typeconf();
```

```
        typewait();
        }



/*
*
*
*        Lire le fichier des headers de messages en attente de
*       l'utilisateur;
*       Afficher les headers lus
*
*
*/


typewait()
        {
        int fdw;                    /* fd du fichier des headers en attente */
        int iwait;                  /* cpte le nbre de messages en attente */

        fdw = open(whead,0);

        iwait = 0;
        while(read(fdw,&bufhead,LHEAD) == LHEAD)
                if(bufhead.nber != 0)
                        {
                        if(iwait == 0) printf("#You have mail .#");
                        iwait++;
                        typehead();
                        }
        close(fdw);
        }

/*
*
*
*        Lire le fichier des confirmations de l'utilisateur .
*       Affichage de chaque confirmation lue .
*
*
```

```
*/


typeconf()
        {
        int fdc;                        /* fd du fichier conf */
        int iconf;                      /* cpte le nbre de conf */
        int i;

        char *ptime;
        char *ctime();

        struct  {
                int num;
                int heur[2];
                char reader[9];
                } confirm;

        fdc = open(conf,2);

        iconf = 0;
        while(read(fdc,&confirm,LCONF) == LCONF)
                {
                if(iconf == 0) printf("Confirmations : #");
                iconf ++;

                printf("%d : ",confirm.num);

                printf("To : ");
                for(i = 0; confirm.reader[i] != ':'; i++)
                                        putchar(confirm.reader[i]);
                for(; i < 10; i++) putchar(' ');

                ptime = ctime(confirm.heur);
                ptime =+ 4 ;
                for(i = 0; i < 15; i++)
                                {
                                putchar(*ptime);
                                ptime++;
                                }
                printf("#");
```

```
                }

        close(fdc);

        /* Quand les confirmations ont ete lues , il faut
           les detruire */
        unlink(conf);
        close(creat(conf,0777));
        }




/*
*
*
*       Ecrire la confirmation ds le fichier des confirmations
*       de l'emetteur
*
*       Il faut :
*               - composer le nom du fichier
*               - ajouter la confirmation en fin de fichier
*
*
*/



writconf()
        {
        extern char conf[];
        extern char lockconf[];

        int i;
        int j;
        int fdc;        /* fd du fichier des conf de l'emetteur */

        struct  {
                int num;
```

```
                int heur[2];
                char reader[9];
                } confirm ;


        /* composer le nom (sender).conf */

        for(j = 9;  j < 15;  j++)
                {
                conf[j] = 'x';
                lockconf[j] = 'x';
                }
        j = 9;
        for(i = 0;  i < 6 && bufhead.sender[i] != ':';i++)
                {
                conf[j] = bufhead.sender[i];
                lockconf[j++] = bufhead.sender[i];
                }

        confirm.num = bufhead.nber ;

        confirm.heur[0] = bufhead.datheur[0];
        confirm.heur[1] = bufhead.datheur[1];

        for(i = 0;  i < 9;  i++) confirm.reader[i] = login[i] ;

        fdc = open(conf,1);
        seek(fdc,0,2);
        write(fdc, &confirm, LCONF);
        close(fdc);
        }



/*
*
*
*       Sauvetage d'un corps de message dans un fichier de travail ;
*
*       Il faut :
*               - demander le nom du fichier
```

```
*               - creer un fichier du mom demande
*               ou si le fichier existe deja,
*               - ouvrir ce fichier
*               - calculer le nbre de blocs de 512 char dont est compose le message
*               - transferer 'nbloc' fois un bloc du fichier input -> output
*
*
*/



save(fd, lmess)
int fd;                 /* file descriptor du fichier contenant le corps
                           de message */
int lmess;              /* longueur du corps */

        {


        char name[15];          /* nom du fichier de sauvetage */

        int i;
        char ok;                /* indique si nom de fichier correctement
                                   tape */
        char c;                 /* caractere tape au terminal */

        ok = 0;
        while (! ok)
                {
                printf("File name (<= 15 char) ? ");

                for(i = 0;(i < 15) && ((c = getchar()) != '#');i++)
                        name[i] = c;
                if(i == 15)
                                {
                                while(getchar() != '#');
                                printf("To long.#");
                                }
                else
                                ok = 1;
                }
```

```
        for( ; i < 15; i++) name[i] ='0';
        trf(fd,lmess,name);
        }

/*
*
*
*       transferer le contenu fu fichier dont le file descriptor est fd
*       et dont la longueur est lmess dans le fichier dont le nom est nfile
*
*
*/


trf(fd,lmess,nfile)
int fd;
int lmess;
char nfile[];
        {
        int fds;                    /* fd du fichier de sauvetage */
        int i;
        int nbloc;                  /* nbre de blocs physiques ds le corps */
        int n;                      /* nbre de char lus ds le fichier input */

        char buf512[512];                   /* buffer pour le corps */

        if((fds = open(nfile,1)) < 0)   /* si nfile n'existe pas */
                {
                fds = creat(nfile,0777);                /* creer nfile */
                }

        seek(fds,0,2);
        nbloc = lmess/512 + 1;

        for (i = 0; i < nbloc; i++)
                {
                if(i == (nbloc-1))
                                n =read(fd,buf512, lmess%512);
                else
                                n = read(fd, buf512, 512);
                write(fds, buf512, n);
```

```
                }
        close(fds);
        }



/*
*
*
*       Imprimer le corps de message courant
*
*
*/


print()
        {
        extern char work[];

        /* variables utilisees lors du fork */
        int pid ;
        int ret;
        int retcode;

        /* pointer au debut du corps */
        seek(fbd,bufhead.ptbody*LBLOC, 0);

        /* transferer le corps ds un fichier de travail */
        maketemp();
        trf(fbd, bufhead.lmess, work);

        /* executer le programme systeme PRN */

        pid = fork();
        if(pid == 0)
                {
                        /* processus fils */


                execl("/bin/prn","prn",work,0);
                exit();
```

```
                }
        /* attendre la fin de l'impression */
        while((ret = wait(&retcode)) != pid && ret != -1);

        unlink(work);
        }



/*
*
*
*       Faire suivre le message vers les destinataires dont le login
*       name est donne par l'utilisateur courant ds la ligne de cde
*
*
*/

forward()
        {
        int fdw;                        /* fd pour fichier de travail */
        int ok ;
        int i;

        char to[100];                   /* liste des utilisateurs vers qui
                                           faire suivre */

        int savptb;                     /* pour sauver le pter vers message */
        int savlm;                      /* pour sauver la longueur du message */

        ok = 0;
        while(! ok)
                {
                printf("To : ");
                i = 0;
                while( i < 100 && (to[i++] = getchar()) != '#');
                if( i == 100)
                                {
                                while( getchar() != '#');
                                printf("To long .#");
```

```
                        }
        else
                        ok = 1;
        i--;
        to[i] = ':';
        }

if(compar(to, login) == 0)
        {
        printf("Stupid to forward a message to yourself .#");
        return;
        }
to[i] = '#';

savptb = bufhead.ptbody;
savlm = bufhead.lmess;

/* ajouter en debut de message le nom du "forwarder" qui
  est le nom de l'utilisateur courant */
maketemp();
fdw = creat(wrkind, 0777);
write(fdw, "Forwarded by ", 13);
for(i = 0; login[i] != ':'; i++);
write(fdw, login, i);
write(fdw, "#", 1);

/* concatener le fichier contenant le "forwarder" et le corps */
seek(fbd, bufhead.ptbody*LBLOC, 0);
cat(fdw, fbd);
fdwrk = open(wrkind, 0);

sendlst(to);
close(fdwrk);
unlink(wrkind);

bufhead.ptbody = savptb;
bufhead.lmess = savlm;
}
```

```
/*
*
*
*       Tester si le message est secret ;
*               si oui : demander le mot de passe de l'utilisateur
*               (celui-ci a 3 chances pour donner le mot de passe correct
*
*       Valeur retournee : 0 si le message pas secret ou si l'utilisateur
*       a donne le mot de passe correct
*                       1 sinon
*
*
*/

secretmess()
        {

        char *getpsw();
        char *ppass;

        int ok;
        int i, j;

        if(bufhead.secret != 'y') return(0);

        getmber();

        ok = 0;
        for(i = 0;i < 3 && !ok; i++)
                {
                ppass = getpsw();

                /* si mot de passe correct , c'est ok */
                for(j = 0; (j < 8) && (bmemb.pass[j] == *ppass++); j++);
                if(j == 8) ok = 1;
                }
        if( ok ) return(0) ; else return(1) ;
        }


/*
```

```
*
*
*        Tester si le verrou est mis ; si oui attendre .
*       Si non , ouvrir les fichiers .
*
*
*/


                /* messages en attente */

openwait()
        {

        while((fdl = open(lock,0)) > 0)close(fdl);      /* verrou ouvert ? */
        fdl = creat(lock,0777);

        fhd = open(whead,2);
        fbd = open(wbody,2);
        fdi = open(windex,2);
        seek(fdi,2,0);
                /* le premier mot est occupe par le cpt de destructions */
        }


                /* messages archives */
openarch()
        {

        while((fdl = open(lock,0)) > 0)close(fdl);
        fdl = creat(lock,0777);

        fhd = open(ahead,2);
        fbd = open(abody,2);
        fdi = open(aindex,2);
        seek(fdi,2,0);
        }


/*
*
*
```

```
*         Fermeture des fichiers de messages
*
*
*/

closefil()
        {

        close(fhd);
        close(fbd);
        close(fdi);

        close(fdl);
        unlink(lock);               /* fermer le verrou */
        }


/*
*
*
*         Lire la ligne de commande
*
*
*/

getline()
        {
        int ok,i;

        printf("* ");                /* <==> utilisateur , vous pouvez taper ! */
        ok = 0;
        while(! ok)
                {
                ptline = line;
                for(i = 0;i < 100 && (*ptline++ = getchar()) != '#'; i++);
                if(i < 100) ok = 1;
                }
        ptline = line;
        }
```

```
/*
*
*
*       Comparer les chaines de caracteres pointees par pt1 et pt2 .
*       si =                : renvoyer 0
*       si (pt1) > (pt2) : renvoyer 1
*       si (pt1) < (pt2) : renvoyer -1;
*
*
*/

compar(pt1,pt2)
char *pt1, *pt2;
        {
        while(*pt1 == ' ')pt1++;
        while(*pt2 == ' ')pt2++;
        while(*pt1 == *pt2)
                        {
                        *pt1++;
                        *pt2++;
                        if(*pt2 == ':')
                                {
                                /* ':' <==> fin de chaine */
                                ptline = pt1;
                                return(0);
                                }
                        }
        if(*pt1 > *pt2) return(1);
        else
        return(-1);
        }



/*
*
*
*        - Lire une date qui est sous forme jj:mm:aa ds LINE
```

```
*       - tester la validite ( jj < 31 et mm < 12 et aa <= 99 )
*       - mettre sous forme aammjj dans dat
*
*
*/


getdate(pdate)
char *pdate;                /* pour mettre le resultat */
        {
        if((*pdate-- = atoi(ptline)) > 31) return(0);
                            /* si jj > 31 : erreur */

        while(*ptline != ':')ptline++;
        ptline++;           /* ptline pointe vers mm */
        if((*pdate-- = atoi(ptline)-1) > 12) return(0);
                            /* si mm > 12 :erreur */

        while(*ptline != ':')ptline++;
        ptline++;           /* ptline pointe vers aa */
        if((*pdate = atoi(ptline)) > 99) return(0);
        return(1);
        }


/*
*
*
*       Comparer dat1 et dat2 (sous la forme aammjj)
*               - si dat1 < dat2 return -1
*               - si dat1 > dat2 return +1
*               - si dat1 = dat2 return 0
*
*
*/

compdate(dat1,dat2)
char dat1[],dat2[];
        {
        int i;
```

```
        for(i = 0;i < 3;i++)
                {
                if(dat1[i] > dat2[i]) return(1);
                if(dat1[i] < dat2[i]) return(-1);
                }
        return(0);
        }



/*
*
*
*       Rechercher ds USER.LST l'entree dont le login name = LOGIN
*
*
*/

getmber()
        {
        int n ;
        int ret;                        /* valeur retournee par COMPAR */
        int fd;                         /* fd de USER.LST */
        int fd2;                        /* fd du verrou */

        /* attendre que verrou soit ouvert */
        while((fd2 = open("/ss/mail/user.lock",0)) > 0)close(fd2);
        /* mettre le verrou */
        fd2 = creat("/ss/mail/user.lock",0777);

        fd = open("/ss/mail/user.lst",0);

        n = LMEMB;
        ret = 1;                        /* init : pas trouve */
        while(n == LMEMB && ret != 0)
                {
                n = read(fd, &bmemb, LMEMB);
                ret = compar(bmemb.log,login);
                }

        close(fd);
```

```
        close(fd2);
        unlink("/ss/mail/user.lock");    /* ouvrir le verrou */
        }



/*
*
*
*       Demander le mot de passe de l'utilisateur ;
*       Se mettre en mode non echo et memoriser les char tapes
*
*
*/

char *getpsw()
        {
        int i;

        char passwrk[50];               /* mot de passe a encrypter */
        char *ptpass;                   /* pointe vers le mot de passe a encrypter */
        char *crypt();                  /* parametre retourne par CRYPT est un pointeur */

        struct {
                char ispeed,ospeed;
                char erase, kill;
                int flag;
                } term, *pterm ;                /* caracteristiques du terminal */

        printf("#Your password , please ? ");

        gtty(0,&term);
        term.flag =& ~ECHO ;
        stty(0,&term);          /* ne pas afficher le caractere tape */

        i = 0;
        while(((passwrk[i] = getchar()) != '#') && (i < 50))i++;

        term.flag =| ECHO ;
        stty(0,&term);
```

```
        passwrk[i] = '0';
        ptpass = crypt(passwrk);                    /* encrypter le mot de passe */
        putchar('#');
        return(ptpass);
        }



/*
*
*
*       Transferer ds LOGIN le nom de l'utilisateur courant
*
*
*/

readlog()
        {
        int i,uid;
        char pwbuf[80];
        char *buf;

        buf = pwbuf;

        /* user-id de l'utilisateur ? */
        uid = getuid() & 0377;

        /* a partir de uid obtenir le login name de l'utilisateur */
        getpw(uid,buf);
        i=0;
        while((login[i++] = *buf++) != ':') ;
        }
```

```
/*
*
*
*
*       Module d'envoi d'un message
*
*
*
*/
```

```
send()
        {
        extern char login[];
        extern int fdwrk;
        extern char work[];
        extern char wrkind[];
        extern char dest[];
        extern char cc[];
        extern struct header {
                int ptbody;             /* pointeur vers le corps */
                int lmess;              /* longueur du corps */
                int nber;               /* numero du message */
                int datheur[2];         /* date & heure d'envoi */
                char sender[9];         /* emetteur du message */
                char ioru;              /* important ou urgent ? */
                char secret;            /* secret ? */
                char subject[51];       /* sujet du message */
                 } bufhead;

        char c;                         /* caractere lu au terminal */

        getbody();
        printf("#OK to send (y/n) ? ");
        if((c = getchar()) != '#')
                while(getchar() != '#');        /* vider le buffer */
        if(c == 'y')
                {
                datnum();
                dialog();
                sendlst(dest);
                sendlst(cc);
                }
        printf("#Save the body in a work file (y/n) ? ");
        if ((c = getchar()) != '#')
                while(getchar() != '#');
        if(c == 'y')
                        {
                        seek(fdwrk,0,0);
                        save(fdwrk, bufhead.lmess);
                        }
        close(fdwrk);
```

```
        unlink(work);
        }



/*
*
*
*       Lecture au terminal du message tape par l'utilisateur .
*       Edition du message si l'utilisateur le demande .
*       Calcul de la longueur du message .
*
*
*/

getbody()

        {
        extern struct header {
                int ptbody;             /* pointeur vers le corps */
                int lmess;              /* longueur du corps */
                int nber;               /* numero du message */
                int datheur[2];         /* date & heure d'envoi */
                char sender[9];         /* emetteur du message */
                char ioru;              /* important ou urgent ? */
                char secret;            /* secret ? */
                char subject[51];       /* sujet du message */
                 } bufhead;

        extern char work[];
        extern int fdwrk;

        struct inode                    /* buffer rempli par l'appel STAT */
                {
                char m[2];
                int n[2];
                char p[4];
                int size;               /* seule la taille du fichier nous interesse */
                int q[12];
                } bnode;
```

```
struct inode *ptnode;

/* variables servant pour l'appel de l'editeur */

int pid;                        /* process id du processus courant */
int rpid;                       /* valeur retournee par le processus fils */
int retcode;



int i;
char a;                 /* avant-avant dernier caractere tape */
char b;                 /* avant-dernier caractere tape */
char c;                 /* dernier caractere tape */
char bwork[512];        /* buffer pour le message */

maketemp();
fdwrk = creat(work,0777);
printf("##");           /* <==> tapez votre message */
a = b = '0' ;
i = 0;
c = '#' ;
while(!((a == '#')&&(b =='.')&&(c == '#')))     /* tant que pas fin du message */
        {
        a = b;
        bwork[i++] = b = c;
        if (i == 512)           /* si buffer rempli */
                        {
                        write(fdwrk,bwork,512);
                        i = 0;
                        }
        c = getchar();
        }
bwork[i++] = '#';
write(fdwrk,bwork,i);           /* vider le buffer */

printf("Do you want to use ED (y/n) ? ");
if((c = getchar()) != '#') while(getchar() != '#');     /* vider le buffer */
if(c != 'n')
        {
        pid = fork();
```

```
                if(pid == 0)                    /* si processus fils */
                        {
                        execl("/bin/ed","ed","-",work,0);
                        exit();
                        }
                while((rpid = wait(&retcode)) != pid && rpid != -1);
                }

        ptnode = &bnode;
        fstat(fdwrk,ptnode);                    /* taille du message ? */
        bufhead.lmess = ptnode->size;           /* reponse dans header.lmess */
        close(fdwrk);                           /* je ferme le fichier en creation */
        fdwrk = open(work,0);                   /* et l'ouvre en lecture */
        }



/*
*
*
*       Etant donne un pointeur vers une liste de destinataires ,
*       envoyer le message vers chacun de ces utilisateurs .
*
*
*/


sendlst(ptlst)
char *ptlst;                        /* pointeur vers liste de destinataires */
        {
        extern int fdwrk;

        char name[50];                          /* destinataire courant */
        char wname[50];
        char *pwname;
        char *pname;                            /* pointe vers caractere courant dans NAME **/
        int eol;                                /* indicateur de fin de ligne */
        int i;
        int ret;                                /* valeur retournee par writmess */

        if(*ptlst == '#')               return; /* liste ne contient aucun destinataire */
```

```
else
                        eol = 0;        /* pas fin de ligne */


/* Tant que la fin de liste n'a pas ete atteinte :
        - mettre dans NAME le nom pointe par PTLST (la fin de nom
          est marquee par ' ' ou '#' )
        - se positionner au debut du fichier contenant le message
        - appeler la routine WRITMESS
                - si WRITMESS retourne une valeur nulle indiquant
                  que NAME n'existe pas : tester si NAME n'est pas
                  le nom d'une liste de destinataire :
                        - si non imprimer un message indiquant que
                          NAME est inconnu .
*/

while(! eol)
        {
        pname = name;
        while((*ptlst != ' ') && ( *ptlst != '#')) *pname++ = *ptlst++;
        if(*ptlst++ == '#') eol = 1;
        *pname++ = ':';
        *pname = '0';
        pname = name;
        pwname = wname;
        while((*pwname = *pname++) != ':') pwname++;
        seek(fdwrk,0,0);
        ret = writmess(name,'w',fdwrk);
        if(ret < 0)
                printf("No more place in %s 's messages file .#",wname);
        else
        if(ret == 0)
                {
                 if(tolst(wname) == 0)
                        {
                        *pwname = '0';
                        printf("%s is unknown .#",wname);
                        }
                }
        }
}
```

```
/*
*
*
*       Tester si NAMLST est un nom de liste
*               - si oui : renvoyer un indicateur d'erreur
*               - si non : envoyer le message aux mbres de la liste
*
*
*/

tolst(nlst)
char nlst[];
        {
        extern int fdwrk;
        extern char namlst[];

        int fdl;        /* fd pour la liste */
        int n;
        int inam;       /* nbre de login names ds le buffer */
        int jnam;       /* adresse d fin du login name ds le buffer */
        int idest;
        int i, j;

        int ok;
        char bufnam[180];       /* buffer pour la liste */
        char namdest[9];        /* mbre de la liste */

        j = 9;
        for(i = 0; nlst[i] != ':' && i < 8; i++) namlst[j++] = nlst[i];
        if(i == 8) return(0);
        namlst[j++] = '.';
        namlst[j++] = 'l';
        namlst[j++] = 's';
        namlst[j++] = 't';
        namlst[j] = '0';
        if((fdl = open(namlst,0)) < 0) return(0);
        ok = 0;
        while(! ok)
                {
```

```
                n = read(fdl, bufnam, 180);
                inam = n/9 ;
                for(i = 0; i < inam; i++)
                        {
                        jnam = i*9 + 9;
                        idest = 0;
                        for(j = i*9; j < jnam; j++)
                                namdest[idest++] = bufnam[j];
                        seek(fdwrk,0,0);
                        if(writmess(namdest, 'w', fdwrk) < 0)
                                printf("No more place in %s 's files.#",namdest);
                        }
                if(n < 180) ok = 1;
                }
        close(fdl);
        return(1);
        }



/*
*
*
*       Lorsqu'un utilisateur envoie un message , il faut lui demander
*       certains renseignements :
*                       - destinataire(s)
*                       - carbon copies
*                       - sujet
*                       - message important ou urgent ?
*                       - message secret ?
*
*
*/


dialog()
        {
        extern char cc[];
        extern char dest[];

        extern struct header {
```

```
        int ptbody;             /* pointeur vers le corps */
        int lmess;              /* longueur du corps */
        int nber;               /* numero du message */
        int datheur[2];         /* date & heure d'envoi */
        char sender[9];         /* emetteur du message */
        char ioru;              /* important ou urgent ? */
        char secret;            /* secret ? */
        char subject[51];       /* sujet du message */
         } bufhead;
extern char wrkind[];
extern int fdwrk;
extern char work[];

int ok;
int fdw;
int i;

printf("#To : ");
i = 0;
while((dest[i++] = getchar()) != '#');

/* creer un fichier intermediaire ds lequel j'ecris le
  nom des destinataires ...   */
fdw = creat(wrkind,0777);
write(fdw,"To : ",5);
write(fdw,dest,i);

printf("Cc : ");
i = 0;
while((cc[i++] = getchar()) != '#');

/* ... et de cc - s'il y en a (si i > 1) */
if(i > 1)
        {
        write(fdw,"Cc : ",5);
        write(fdw,cc,i);
        }

ok = 0;
while(!ok)
        {
```

```
        printf("Subject : ");
        ok = 1;
        for(i = 0;(bufhead.subject[i] = getchar()) != '#';i++)
                if(i == 50)
                        {
                        printf("To long subject .#");
                        while(getchar() != '#');
                        ok = 0;
                        break;
                        }
        }

printf("Urgent or Important (u/i) or CR ? " );
if((bufhead.ioru = getchar()) != '#')
        while(getchar() != '#');                        /* vider buffer */

printf("Secret message (y/n) ? ");
if((bufhead.secret = getchar()) != '#')
        while(getchar() != '#');                        /* vider buffer */

/* concatener le fichier contenant les destinataires et le
  fichier contenant le corps de message */

cat(fdw,fdwrk);
close(fdwrk);
unlink(work);
link(wrkind,work);
unlink(wrkind);
fdwrk = open(work,0);
}



/*
*
*
*       Garnir  - header.nber
*               - header.datheur
*               - header.sender
*
*
```

```
*/


datnum()
        {
        extern struct header {
                int ptbody;             /* pointeur vers le corps */
                int lmess;              /* longueur du corps */
                int nber;               /* numero du message */
                int datheur[2];         /* date & heure d'envoi */
                char sender[9];         /* emetteur du message */
                char ioru;              /* important ou urgent ? */
                char secret;            /* secret ? */
                char subject[51];       /* sujet du message */
                 } bufhead;

        extern char login[];

        int fd;                 /* fd du verrou */
        int fdn;                /* fd du fichier NBER.LAST */
        int last;               /* buffer pour NBER.LAST */
        int i;

        time(bufhead.datheur);
        while((fd = open("/ss/mail/nber.lock",0)) > 0)  /* attendre que le verrou soit libere;
                                                           mettre le verrou */
                                        close(fd);
        fd = creat("/ss/mail/nber.lock",0777);
        fdn = open("/ss/mail/nber.last",2);
        read(fdn,&last,2);                              /* dernier numero attribue */
        bufhead.nber = ++last;                          /* est incremente */
        seek(fdn,0,0);
        write(fdn,&last,2);                             /* et recopie */
        close(fdn);
        close(fd);
        unlink("/ss/mail/nber.lock");                           /* liberer le verrou */

        for(i=0;(bufhead.sender[i] = login[i]) != ':';i++);
        }
```

```
cat(fd1,fd2)
int fd1;                    /* fichier de sortie */
int fd2;                    /* fichier d'entree */
        {
        extern struct header {
                int ptbody;                 /* pointeur vers le corps */
                int lmess;                  /* longueur du corps */
                int nber;                   /* numero du message */
                int datheur[2];             /* date & heure d'envoi */
                char sender[9];             /* emetteur du message */
                char ioru;                  /* important ou urgent ? */
                char secret;                /* secret ? */
                char subject[51];           /* sujet du message */
                 } bufhead;

        extern char wrkind[];

        struct inode                        /* buffer rempli par l'appel STAT */
                {
                char m[2];
                int n[2];
                char p[4];
                int size;                   /* seule la taille du fichier nous interesse */
                int q[12];
                } bnode;
        struct inode *ptnode;

        int nbloc;      /* nbre de blocs physiques ds fd2 */
        int i;
        char buf512[512];
        int n;

        nbloc = bufhead.lmess / 512 + 1;
        for(i = 0; i < nbloc; i++)
                {
                if(i == (nbloc - 1))    n = read(fd2,buf512,bufhead.lmess % 512);
                else
                                        n = read(fd2,buf512,512);
                write(fd1,buf512,n);
                }
```

```
        close(fd1);
        ptnode = &bnode;
        stat(wrkind, ptnode);
        bufhead.lmess = ptnode->size;
        }



/*
*
*
*       Repondre au message affiche
*
*
*/

reply()
        {
        int i, j;
        int num;

        char c;                         /*caractere lu au terminal */

        char *phead;
        char *pwhead;

        char *ptime;
        char *ctime();

        extern char suj1[];
        extern char dest[];
        extern int fdwrk;
        extern char work[];
        extern char login[];

        extern struct header {
                int ptbody;             /* pointeur vers le corps */
                int lmess;              /* longueur du corps */
                int nber;               /* numero du message */
                int datheur[2];         /* date & heure d'envoi */
```

```
        char sender[9];          /* emetteur du message */
        char ioru;               /* important ou urgent ? */
        char secret;             /* secret ? */
        char subject[51];        /* sujet du message */
         } bufhead;

extern struct index {
        int nbre;                /* numero du message */
        char date[3];            /* date sous la forme aammjj */
        int pthead;              /* pointeur vers le header */
     } bindex;

struct {
        int ptbody;              /* pointeur vers le corps */
        int lmess;               /* longueur du corps */
        int nber;                /* numero du message */
        int datheur[2];          /* date & heure d'envoi */
        char sender[9];          /* emetteur du message */
        char ioru;               /* important ou urgent ? */
        char secret;             /* secret ? */
        char subject[51];        /* sujet du message */
        } bthead;

struct {
        int nbre;                /* numero du message */
        char date[3];            /* date sous la forme aammjj */
        int pthead;              /* pointeur vers le header */
        } btind;


if(compar(bufhead.sender,login) == 0)
                {
                printf("Stupid to reply to yourself .#");
                return;
                }

/* sauver header et index courants */

phead = &(bufhead.subject[50]);
pwhead = &(bthead.subject[50]);
for(i = 0; i < LHEAD; i++) *pwhead-- = *phead-- ;
```

```
btind.nbre = bindex.nbre;
for(i = 0; i < 3; i++) btind.date[i] = bindex.date[i];
btind.pthead = bindex.pthead;


getbody();
printf("Ok to send (y/n) ? ");
if((c = getchar()) != '#') while(getchar() != '#');
if(c == 'y')
        {
        datnum();

        /* sujet = Reply to message 'num de message lu' (date d'envoi) */

        for(j = 0; j < 17; j++) bufhead.subject[j] = suj1[j];
        num = bthead.nber;
        j = 21;          /* indice du numero ds SUBJECT */
        for(i = 0; i < 5; i++)
                {
                bufhead.subject[j--] = num % 10 + '0';
                num = num / 10 ;
                }
        for(i = 17; i < 22 && bufhead.subject[i] == '0'; i++);
        j = 17;
        for(; i < 22; i++) bufhead.subject[j++] = bufhead.subject[i];
        bufhead.subject[j++] = '(';
        ptime = ctime(bthead.datheur) ;
        ptime =+ 4;
        for(i = 0; i < 12; i++)
                                bufhead.subject[j++] = *ptime++;
        bufhead.subject[j++] = ')';
        bufhead.subject[j] = '#';

        /* destinataire de la reponse = emetteur du message */

        for(i = 0;(dest[i] = bthead.sender[i]) != ':' ; i++);
        dest[i] = '#' ;
        sendlst(dest);
        }
printf("#Save the body in a work file (y/n) ?");
```

```
if((c = getchar()) != '#') while(getchar() != '#');
if(c == 'y')     {
                 seek(fdwrk, 0, 0);
                 save(fdwrk, bufhead.lmess);
                 }
close(fdwrk);
unlink(work);


/* restorer header et index */

phead = &(bufhead.subject[50]);
pwhead = &(bthead.subject[50]);
for(i = 0; i < LHEAD; i++) *phead-- = *pwhead--;

bindex.nbre = btind.nbre;
for(i = 0;i < 3; i++) bindex.date[i] = btind.date[i];
bindex.pthead = btind.pthead;

}
```

```
#


/*
*
*
*
*       Fonctions annexes :
*                       - creer une liste
*                       - detruire une liste
*                       - executer une commande de UNIX
*                       - se declarer membre du systeme de courrier
*                       - declarer ne plus etre membre du courrier
*                       - consulter le repertoire des utilisateurs
*
*
*       +
*
*       Routines de composition des noms de fichiers
*
*
*
*/
```

```
#define LMEMB   43
#define ECHO    010


/* signaux utiles pour executer la routine SHELL  */

#define SIGHUP  1
#define SIGINTR 2
#define SIGQUIT 3
#define SIGGP   19



/*
*
*
*
*       L'utilisateur veut-il faire une recherche selon :
*               - le login name
*               - le nom
*               - generale
*
*
*/

getuser()
        {
        extern char *ptline;

        if(compar(ptline,"login:") == 0) userlog();
        else
        if(compar(ptline,"name:") == 0) userpatro();
        else
        if(compar(ptline,"#:") == 0) userall();
        }

/*
*
*
*       Afficher le repertoire
*
```

```
*
*/

userall()
        {
        extern struct member {
                char log[9];
                char name[25];
                char pass[8];
                char nl;
                } bmemb;

        extern int fdl;

        int fdu;                        /* fd de USER.LST */

                /* attendre que le verrou soit ouvert */
        while((fdl = open("/ss/mail/user.lock",0)) > 0)close(fdl);
                /* fermer le verrou ... en priant pour qu'un autre
                   utilisateur ne vienne pas le fermer entre les 2
                   instructions (cette derniere instruction n'est pas
                    encore implementee )
                */
        fdl = creat("/ss/mail/user.lock",0777);

        fdu = open("/ss/mail/user.lst",0);

                /*
                  Pour chaque article du repertoire :
                        - afficher login name et nom a l'ecran
                */

        while(read(fdu, &bmemb, LMEMB) == LMEMB)
                typember();

        close(fdu);
        close(fdl);
        unlink("/ss/mail/user.lock");           /* ouvrir le verrou */
        }
```

```
/*
*
*
*       Recherche des entrees dont le login = login ds la ligne de
*       commande
*
*
*/


userlog()
        {
        extern char *ptline;
        extern struct member {
                char log[9];
                char name[25];
                char pass[8];
                char nl;
                } bmemb;

        extern int fdl;

        int fdu;                        /* fd de USER.LST */

        char *ptwrk;                    /* pointeur de travail ds LINE */

        ptwrk = ptline;
        while(*ptline != '#')ptline++;
        *ptline = ':';

        while((fdl = open("/ss/mail/user.lock",0)) > 0)close(fdl);
        fdl = creat("/ss/mail/user.lock",0777);

        fdu = open("/ss/mail/user.lst",0);

        while(read(fdu,&bmemb,LMEMB) == LMEMB)
                {
                        /* si login name tape correspond a login name ds
                                USER.LST */
                if(compar(bmemb.log,ptwrk) == 0) typember();
                }
```

```
        close(fdu);
        close(fdl);
        unlink("/ss/mail/user.lock");
        }


/*
*
*
*       Recherche des articles dont le nom correspond au nom tape
*       par l'utilisateur
*
*
*/

userpatro()
        {
        extern struct member {
                char log[9];
                char name[25];
                char pass[8];
                char nl;
                } bmemb;

        extern char *ptline;
        extern int fdl;

        int fdu;                        /* fd de USER.LST */
        char *ptwrk;                    /* pointeur de travail ds LINE */

        ptwrk = ptline;
        while(*ptline != '#') ptline++;
        *ptline = ':';

        while((fdl = open("/ss/mail/user.lock",0)) > 0)close(fdl);
        fdl = creat("/ss/mail/user.lock",0777);

        fdu = open("/ss/mail/user.lst",0);

        while(read(fdu,&bmemb,LMEMB) == LMEMB)
```

```
                if(compar(bmemb.name,ptwrk) == 0) typember();

        close(fdu);
        close(fdl);
        unlink("/ss/mail/user.lock");
        }


/*
*
*
*       Afficher - bmemb.log
*                - bmemb.name
*
*
*/

typember()
        {
        extern struct member {
                char log[9];
                char name[25];
                char pass[8];
                char nl;
                } bmemb;

        int i;

        for(i = 0; bmemb.log[i] != ':'; i++) putchar(bmemb.log[i]) ;
        printf("  :  ");
        for(i = 0; bmemb.name[i] != ':'; i++) putchar(bmemb.name[i]);
        putchar('#');
        }
```

```
/*
*
*
*       Creer une liste de destinataires
*
*
*/

creatlst()
        {
        extern char *ptline;
        extern char namlst[];           /* prefixe du nom de liste */
        char logname[9];        /* nom de login tape par l'utilisateur */
        char c;                 /* char lu au terminal */

        int fdl;
        int i;
        int eol;

        extern char *ptline;

        while(*ptline == ' ')ptline++;
        for(i = 9; (*ptline != '.') && (*ptline != '#') && (i < 17); i++)
                        namlst[i] = *ptline++;


        /* Le nom de la liste doit avoir une longueur < 8 char et ne pas
          contenir de '.';
        */

        if(i == 17 || *ptline == '.')
                        {
                        if(i == 17) while(getchar() != '#');
                        printf("Incorrect list name .#");
                        return;
                        }

        /* ajouter au nom donne par l'utilisateur le suffixe ".LST" */

        namlst[i++] = '.';
```

```
namlst[i++] = 'l';
namlst[i++] = 's';
namlst[i++] = 't';
namlst[i] = '0';

/* si une telle liste existe deja ds le systeme de courrier :
        signaler l'erreur
*/

if((fdl = open(namlst,1)) > 0)
                        {
                        printf("Allready such a list in the mail .#");
                        close(fdl);
                        return;
                        }

fdl = creat(namlst,0777);
printf("Login names ?#");
eol = 0;
while(! eol)
        {
        /* L'utilisateur tape une suite de login name separes par CR.
           Pas de verification d'existence .          .
        */

        for(i = 0; i < 9 && (c = getchar()) != '#'; i++)
                        logname[i] = c;
        if(i == 9)
                        {
                        while(getchar() != '#');
                        printf("To long login name . List deleted .#");
                        close(fdl);
                        unlink(namlst);
                        return;
                        }

                /* si premier caractere tape est un '.' :
                        fin de liste ;
                   sinon inserer le login name tape ds NAMLST ;
                */
```

```
                if(logname[0] == '.') eol = 1;
                else
                                        {
                                        logname[i] = ':';
                                        write(fd1,logname,9);
                                        }
                }
        close(fd1);
        }


/*
*
*
*       Detruire une liste de destinataires
*
*
*/

deletlst()
        {
        extern char *ptline;
        extern char namlst[];
        char c;                         /* char lu au terminal */

        int i;

        extern char *ptline;

        while(*ptline == ' ')ptline++;
        for(i = 9; (*ptline != '.') && (*ptline != '#') && (i < 17); i++)
                                namlst[i] = *ptline++;
        if(i == 17 || *ptline == '.')
                                {
                                printf("Incorrect list name .#");
                                return;
                                }
        namlst[i++] = '.';
        namlst[i++] = 'l';
        namlst[i++] = 's';
        namlst[i++] = 't';
```

```
        namlst[i] = '0';
        if(unlink(namlst) < 0) printf("No such list.#");
        }




/*
*
*
*       Appel du SHELL
*
*
*/

shell()
        {
        int savint, savhup, savquit, savgp;
        int pid, rpid, retcode;

        printf("%% ");
        if((pid = fork()) == 0)
                {
                signal(SIGHUP, 0);
                signal(SIGINTR, 0);
                signal(SIGQUIT, 0);
                signal(SIGGP, 0);
                execl("/bin/sh", "sh", "-t", 0);
                exit();
                }
        savhup = signal(SIGHUP,1);
        savint = signal(SIGINTR, 1);
        savquit = signal(SIGQUIT, 1);
        savgp = signal(SIGGP, 1);

        while((rpid = wait(&retcode)) != pid && rpid != -1);
        signal(SIGHUP, savhup);
        signal(SIGINTR, savint);
        signal(SIGQUIT, savquit);
```

```
        signal(SIGGP, savgp);
        }




/*
*
*
*       - detruire les fichiers de message de l'utilisateur
*
*       - detruire l'entree relative a l'utilisateur ds USER.LST
*
*
*/


delmber()
        {
        extern char login[];
        extern char wbody[];
        extern char abody[];
        extern char wfbody[];
        extern char afbody[];
        extern char whead[];
        extern char ahead[];
        extern char wfhead[];
        extern char afhead[];
        extern char aindex[];
        extern char windex[];
        extern char conf[];

        duser();


/*      Destruction des fichiers de message en attente et du fichier des conf */
```

```
        makewait(login);

        unlink(conf);
        unlink(wbody);
        unlink(wfbody);
        unlink(whead);
        unlink(wfhead);
        unlink(windex);


/*      Destruction des fichiers de message archive */

        makearch(login);

        unlink(ahead);
        unlink(afhead);
        unlink(aindex);
        }

/*
*
*
*       detruire l'entree relative al'uitlisateur ds USER.LST
*
*
*/

duser()
        {
        extern char login[];
        struct temp                                  /* buffer pour le fichier USER.LST */
                {
                char l[9];                  /*login name */
                char n[25];                 /* nom */
                char p[8];                  /* mot de passe */
                char nl;                    /* CR */
                } btemp;

        int fd1;                            /* file descriptor de USER.LST */
        int fd2;                            /* file descriptor du fichier de travail */
        int fd3;                            /* file descriptor du verrou */
```

```
while((fd3 = open("/ss/mail/user.lock",0)) > 0) /* mettre le verrou */
                                        close(fd3);
fd3 = creat("/ss/mail/user.lock",0777);

fd2 = creat("/ss/mail/user.wrk",0777);                    /* creer un fichier de
                                                          travail pour copier
                                                         la nouvelle liste */
fd1 = open("/ss/mail/user.lst",2);


while(read(fd1,&btemp,LMEMB) == LMEMB)
        {
        if(compar(btemp.1,login) != 0)
                                        write(fd2,&btemp,LMEMB);
        }

close(fd1);
close(fd2);
unlink("/ss/mail/user.lst");
link("/ss/mail/user.wrk","/ss/mail/user.lst");
unlink("/ss/mail/user.wrk");                /* Et on a copie le fichier de travail
                                        dans USER.LST */
close(fd3);
unlink("/ss/mail/user.lock");               /* defaire le verrou */
}
```

```
/*
*
*
*       NEWMBER permet l'insertion d'un nouvel utilisateur dans le courrier .
*       Les operations a remplir pour ce faire sont les suivantes :
*               - creation et initialisation des fichiers de message et
*                 de confirmation
*               - inserer le membre dans le repertoire des utilisateurs
*
*
```

```
*/



newmber()
        {
        extern char wbody[];
        extern char wfbody[];
        extern char whead[];
        extern char wfhead[];
        extern char windex[];
        extern char conf[];
        extern char ahead[];
        extern char afhead[];
        extern char aindex[];
        extern char login[];

        char c;                         /* caractere lu au terminal */

        int izero;
        int i;
        int fd;                         /* file descriptor du fichier courant */

        char czero;

        if(! insmber()) return;
        izero = 0;
        czero = 0;

/* creation du fichier de message en attente et du fichier CONF */

        makewait(login);
        close(creat(conf,0777));
        close(creat(wbody,0777));
        close(creat(whead,0777));
        fd = creat(windex,0777);
        write(fd,&izero,2);
        close(fd);
```

```
        fd = creat(wfhead,0777);
        for(i = 0;i < 100;i++)
                write(fd,&czero,1);
        close(fd);

        fd = creat(wfbody,0777);
        for(i = 0;i < 100;i++)
                write(fd,&czero,1);
        close(fd);

/* creation des fichiers de messages archives */

        makearch(login);
        close(creat(ahead,0777));
        fd = creat(afhead,0777);
        for(i = 0;i < 100;i++)
                write(fd,&czero,1);

        fd = creat(aindex,0777);
        write(fd,&izero,2);
        close(fd);
        }



/*
*
*
*       Insertion du nouveau membre dans le repertoire .
*
*
*/

insmber()
        {
        extern char login[];

        extern struct member {
                char log[9];
                char name[25];
```

```
        char pass[8];
        char nl;
        } bmemb;
struct temp                                 /* buffer pour le fichier USER.LST */
        {
        char l[9];                  /*login name */
        char n[25];                 /* nom */
        char p[8];                  /* mot de passe */
        char nl;                    /* CR */
        } btemp;


struct {
        char ispeed, ospeed;
        char erase, kill;
        int flag;
        } term , *pterm;                    /* caracteristiques du teminal */

int i;
int insert;                         /* nouveau mbre est insere ? */
int ret;                            /* valeur retournee par compar */
int all;                            /* valeur retournee a newmber */
int fd1;                            /* file descriptor de USER.LST */
int fd2;                            /* file descriptor du fichier de travail */
int fd3;                            /* file descriptor du verrou */

char c;                             /* caractere lu au terminal */
char passwrk[50];                   /* mot de passe a encrypter */
char *ptpass;                       /* pointe vers le mot de passe a encrypter */
char *crypt();                      /* parametre retourne par CRYPT est un pointeur */


for(i = 0; (bmemb.log[i] = login[i]) != ':'; i++);

printf("Your name , please (< 25 char) ? ");
i = 0;
while((c = getchar()) != '#')
        {
        if(i < 25) bmemb.name[i++] = c;
        else
```

```
                        {
                        while(getchar() != '#');      /* vider le buffer */
                        printf("More than 25 char .#You name , please ? ");
                        i = 0;
                        }
        }
bmemb.name[i] = ':';

printf("Your password , please ? ");

pterm = &term;
gtty(0,pterm);
pterm->flag =& ~ECHO ;
stty(0,pterm);              /* ne pas afficher le caractere tape */

i = 0;
while((passwrk[i] = getchar()) != '#')i++;
passwrk[i] = '0';
ptpass = passwrk;
ptpass = crypt(ptpass);          /* encrypter le mot de passe */
for(i = 0;i < 8;i++) bmemb.pass[i] = *ptpass++;
bmemb.nl = '#';

pterm->flag =| ECHO ;
stty(0,pterm);                   /* remettre le TTY dans son etat
                                    initial */

while((fd3 = open("/ss/mail/user.lock",0)) > 0) /* mettre le verrou */
                                 close(fd3);
fd3 = creat("/ss/mail/user.lock",0777);

fd2 = creat("/ss/mail/user.wrk",0777);                  /* creer un fichier de travail
                                                        pour l'insertion */
fd1 = open("/ss/mail/user.lst",2);
insert = 0;                      /* nouveau membre pas encore insere */

/*
* Pour chaque article lu dans USER.LST :
*       - si l'insertion n'a pas encore eu lieu et que le login name
*         du nouveau membre est plus petit que le login name lu :
*               - inserer le nouveau membre dans le fichier de travail
```

```
*       - ecrire le contenu du buffer dans le fichier de travail
* Si l'insertion n'a pas eu lieu , ecrire le nouveau membre dans le
* fichier de travail .
*/

while(read(fd1,&btemp,LMEMB) == LMEMB)
        {
        if((ret = compar(bmemb.log,btemp.l)) == 0)
                {
                printf("You are allready in mail system .#");
                insert++;
                all = 0;
                }
        if(ret < 0 && !insert)
                {
                write(fd2,&bmemb,LMEMB);
                insert++;
                all = 1;
                }
        write(fd2,&btemp,LMEMB);
        }
if(! insert)    { write(fd2,&bmemb,LMEMB);
                  all = 1;
                }

close(fd1);
close(fd2);
unlink("/ss/mail/user.lst");
link("/ss/mail/user.wrk","/ss/mail/user.lst");
unlink("/ss/mail/user.wrk");                    /* Et on a copie le fichier de travail
                                        dans USER.LST */
close(fd3);
unlink("/ss/mail/user.lock");                   /* defaire le verrou */
return(all);
}
```

```
        /*
        *
        *
        *       Routines de composition des noms de fichiers
        *
        *
        */



/*
*       Composition des noms de fichier de message en attente .
*/

makewait(name)
char name[];                /* prefixe du nom de fichier */
        {
        extern char wbody[];
        extern char wfbody[];
        extern char whead[];
        extern char wfhead[];
        extern char windex[];
        extern char conf[];
        extern char lock[];

        int i;
        int j;

        for(j = 9; j < 15; j++)
                {
                wbody[j] = 'x';
                wfbody[j] = 'x';
                whead[j] = 'x';
                wfhead[j] = 'x';
                windex[j] = 'x';
                conf[j] = 'x';
                lock[j] = 'x';
                }
        j = 9;
        for(i = 0;i < 6 && name[i] != ':';i++)
                {
```

```
                wbody[j] = name[i];
                wfbody[j] = name[i];
                whead[j] = name[i];
                wfhead[j] = name[i];
                windex[j] = name[i];
                conf[j] = name[i];
                lock[j++] = name[i];
                }
        }



/*
*       Composition des noms de fichier de messages archives .
*/

makearch(name)
char name[];                            /* prefixe du nom de fichier */
        {
        extern char abody[];
        extern char afbody[];
        extern char ahead[];
        extern char afhead[];
        extern char aindex[];
        extern char lock[];

        int i;
        int j;

        for(j = 9; j < 15; j++)
                {
                abody[j] = 'x';
                afbody[j] = 'x';
                ahead[j] = 'x';
                afhead[j] = 'x';
                aindex[j] = 'x';
                lock[j] = 'x';
                }
        j = 9;
        for(i = 0;i < 6 && name[i] != ':';i++)
```

```
                {
                abody[j] = name[i];
                afbody[j] = name[i];
                ahead[j] = name[i];
                afhead[j] = name[i];
                aindex[j] = name[i];
                lock[j++] = name[i];
                }
        }


/*      Composition d'un nom de fichier temporaire .  */

maketemp()
        {
        extern char work[];
        extern char wrkind[];

        int i,pid,d;

        pid = getpid();
        for(i=15;i>=11;--i)
                {
                d = (pid&07) + '0';
                work[i] = d;
                wrkind[i] = d;
                pid =>> 3;
                }
        }
```

```
/*
*
*
*
*       Modules de gestion des fichiers de messages
*
*
*
*/
```

```
#define LBLOC    100              /* longueur d'un bloc dans un fichier body */
#define ZERO     00
#define LHEAD    72               /* longueur d'un header */
#define LINDEX   7                /* longueur d'un index */
#define LTAB     100              /* longueur de tab */


/* variables pour les routines SEEKFREE et MAKEFREE */

char tab[LTAB];          /* buffer pour un fichier fbody ou fhead */

/* le ieme element de ce vecteur permet de tester si le ieme bit
   d'un byte est a 0 ou a 1 */

char key1[] {
              0200,
              0100,
              040,
              020,
              010,
              04,
              02,
              01
             };

/* le i eme element de ce vecteur permet de mettre le i eme bit d'un
   byte a 0 */

char key2[] {
              0177,
              0277,
              0337,
              0357,
              0367,
              0373,
              0375,
              0376
             };
```

```
/*
*
*
*       reserver "nbloc" blocs consecutifs dans le fichier "file";
*       SEEKFREE retourne comme valeur l'indice du premier bloc libre
*       par rapport au debut du fichier .
*
*
*/

seekfree(file,nbloc)
char *file;
int nbloc;
        {
        int n;                          /* nombre de bytes lus dans file */
        int fd;                         /* file descriptor de "file" */
        int i;                          /* numero du byte courant */
        int j;
        int ifree;
        int byte;                       /* numero du byte dans tab */
        int bit;                        /* numero du bit dans "byte" */
        int indice;                     /* numero du premier bloc libre dans "file" */

        if(nbloc == 0) return(-2);      /* stupide de reserver 0 bloc */
        fd = open(file,2);
        n = read(fd,tab,LTAB);
        ifree = 0;
        i = 0;

/* explication de l'algorithme (1 bit <==> 1 bloc) :

        tant que "nbloc" blocs n'ont pas ete reserves :
                tester si le bit courant est a 0:
                si oui :
                        si le nbre de blocs actuellement reserve = 0 :
                                byte = numero du byte courant(i)
                                bit = numero du bit courant(j)
                        ajouter 1 au nombre de blocs reserves(ifree)
```

```
                si non :
                        nbre de blocs reserves = 0
*/

        while((i < n) && (ifree < nbloc))
                {
                j = 0;
                while((j < 8) && (ifree < nbloc))
                        {
                        if((tab[i] & key1[j]) == 0)
                                {
                                if(ifree == 0) { byte = i;
                                                 bit = j;
                                               }
                                ifree++;
                                }

                        else
                                ifree = 0;
                        j++;
                        }
                i++;
                }


        if(ifree != nbloc) return(-1);   /* plus de place !!! */
        indice = byte*8 + bit;           /* petit exemple :
                                            si byte = 3 et bit = 1 ,
                                            le premier bloc libre est le 25eme bloc */


        /* mettre a 1 les bits representant les blocs venant d'etre reserves */

                for(i=0; i<ifree; i++)
                {
                tab[byte] =| key1[bit++];
                if(bit == 8) {
                             byte++;
                             bit = 0;
                             }
```

```
                }
        seek(fd,0,0);
        write(fd,tab,n);
        close(fd);
        return(indice);
        }



/*
*
*
*       liberer "nbloc" blocs dans le fichier "file" a partir de la
*       position "indice"
*
*
*/

makefree(file,nbloc,indice)
char file[];
int nbloc,indice;
        {
        int n;                          /* nbre de bytes lus dans tab */
        int fd;                         /* file descriptor de "file" */
        int byte;
        int bit;
        int i;

        fd = open(file,2);
        n = read(fd,tab,LTAB);
        byte = indice / 8;
        bit = indice % 8;


        /* a partir du bit "bit dans le byte "byte" ,mettre les "nbloc"
           bits a 0
        */

        for(i=0;i<nbloc;i++)
```

```
                {
                tab[byte] =& key2[bit++];
                if(bit == 8) {
                                byte++;
                                bit = 0;
                             }
                }
        seek(fd,0,0);
        write(fd,tab,n);
        close(fd);
        }


/*
*
*
*       Indiquer "nbloc" blocs comme etant occupes a partir de
*       la position "indice" .
*
*
*/



makebusy(file,nbloc,indice)
char file[];
int nbloc,indice;
        {
        int n;                  /* nbre de bytes lus dans tab */
        int fd;                 /* file descriptor de "file" */
        int byte;
        int bit;
        int i;

        fd = open(file,2);
        n = read(fd,tab,LTAB);
        byte = indice / 8;
        bit = indice % 8;


        /* a partir du bit "bit dans le byte "byte" ,mettre les "nbloc"
```

```
            bits a 1
        */

        for(i=0;i<nbloc;i++)
                {
                tab[byte] =| key1[bit++];
                if(bit == 8) {
                                byte++;
                                bit = 0;
                             }
                }
        seek(fd,0,0);
        write(fd,tab,n);
        close(fd);
        }



/*
*
*
*       Ecrire le message(corps + header + index) courant dans les fichiers
*       de messages dont le prefixe est "name" . Si wora = 'w' , l'ecriture
*       se fera dans les fichiers de messages en attente ; si wora = 'a' ,
*       l'ecriture se fera dans les fichiers de messages archives .
*
*
*/

writmess(name,wora,fdo)
char name[];
char wora;
int fdo;                        /* file descriptor du fichier contenant le
                                   corps du message a transferer */
        {
        extern char lock[];
        extern char afbody[];
        extern char abody[];
        extern char wbody[];
        extern struct header {
                int ptbody;             /* pointeur vers le corps */
```

```
        int lmess;              /* longueur du corps */
        int nber;               /* numero du message */
        int datheur[2];         /* date & heure d'envoi */
        char sender[9];         /* emetteur du message */
        char ioru;              /* important ou urgent ? */
        char secret;            /* secret ? */
        char subject[51];       /* sujet du message */
         } bufhead;

char bufbody[100];
int nbloc;
int fd;                 /* file descriptor du fichier "lock" */
int ret;                /* valeur retournee par WRITBODY */

if(wora == 'w')     { makewait(name);
                      while((fd = open(lock,0)) > 0)close(fd);
                      fd = creat(lock,0777);
                      if((ret = writbody(wora,fdo)) <= 0)
                         {
                         close(fd);
                         unlink(lock);    /* ouvrir le verrou */
                         return(ret);
                         }
                    }
else
                {
                   makearch(name);
                   nbloc = bufhead.lmess/LBLOC + 1;
                   makebusy(afbody,nbloc,bufhead.ptbody);
                }
writhead(wora);
writind(wora);
if(wora != 'a')
                {
                close(fd);
                unlink(lock);           /* liberer le verrou */
                }
return(1);
}
```

```
/*
*
*
*       Ecrire le corps dans un fichier body .
*
*
*/

writbody(wora,fdo)
char wora;
int fdo;                        /* file descriptor du fichier contenant le message */
        {
        extern char wbody[];
        extern char abody[];
        extern char wfbody[];
        extern char afbody[];
        extern struct header
                {
                int ptbody;             /* pointeur vers le corps */
                int lmess;              /* longueur du corps */
                int nber;               /* numero du message */
                int datheur[2];         /* date & heure d'envoi */
                char sender[9];         /* emetteur du message */
                char ioru;              /* important ou urgent ? */
                char secret;            /* secret ? */
                char subject[51];       /* sujet du message */
                 } bufhead;
        extern int fbd;

        int fd1;                /* file descriptor du fichier body */
        int nbloc;              /* nbre de bloc dans le corps */
        int ret;                /* valeur retournee par SEEKFREE */
        int i;
        int n;
        int disp;
        char bufbody[LBLOC];

        nbloc = bufhead.lmess / LBLOC + 1;
        if((fd1 = open(wbody,1)) < 0) return(0);
        bufhead.ptbody = seekfree(wfbody,nbloc);
        if(bufhead.ptbody < 0) return(bufhead.ptbody);
```

```
        seek(fd1,bufhead.ptbody*LBLOC,0);

        /*  transferer un bloc du fichier dont le file descriptor est fdo
           dans le fichier body "nbloc" fois .
        */

        for(i=0;i<nbloc;i++)
                {
                        read(fdo,bufbody,LBLOC);
                        write(fd1,bufbody,LBLOC);
                }
        close(fd1);
        return(1);
        }


/*
*
*
*       Ecriture d'un header dans un fichier whead ou ahead selon la
*       valeur de wora .
*
*
*/

writhead(wora)
char wora;
        {
        extern char ahead[];
        extern char afhead[];
        extern char wfhead[];
        extern char whead[];

        extern struct header {
                int ptbody;             /* pointeur vers le corps */
                int lmess;              /* longueur du corps */
                int nber;               /* numero du message */
                int datheur[2];         /* date & heure d'envoi */
                char sender[9];         /* emetteur du message */
                char ioru;              /* important ou urgent ? */
```

```
                char secret;                    /* secret ? */
                char subject[51];               /* sujet du message */
                 } bufhead;
        extern struct index {
                int nbre;                       /* numero du message */
                char date[3];                   /* date sous la forme aammjj */
                int pthead;                     /* pointeur vers le header */
             } bindex;
        int fd;                         /* file descriptor du fichier head */
        int disp;

        if(wora == 'w')
                        {
                         fd = open(whead,1);
                         bindex.pthead = seekfree(wfhead,1);
                        }
        else
                        {
                         fd = open(ahead,1);
                         bindex.pthead = seekfree(afhead,1);
                        }
        disp = bindex.pthead * LHEAD;    /* deplacement reel = indice * longueur */
        seek(fd,disp,0);
        write(fd,&bufhead,LHEAD);
        close(fd);
        }



/*
*
*
*       Ecriture de l'index dans un fichier windex ou aindex selon la
*       valeur de wora .
*
*
*/

writind(wora)
char wora;
        {
```

```
extern struct header {
        int ptbody;             /* pointeur vers le corps */
        int lmess;              /* longueur du corps */
        int nber;               /* numero du message */
        int datheur[2];         /* date & heure d'envoi */
        char sender[9];         /* emetteur du message */
        char ioru;              /* important ou urgent ? */
        char secret;            /* secret ? */
        char subject[51];       /* sujet du message */
         } bufhead;
extern struct index {
        int nbre;               /* numero du message */
        char date[3];           /* date sous la forme aammjj */
        int pthead;             /* pointeur vers le header */
     } bindex;
extern char aindex[];
extern char windex[];
extern char afhead[];
extern char wfhead[];
extern char wrkind[];

int fd1;                /* file descriptor du ficheir index */
int fd2;                /* file descriptor du fichier de travail */
int mod;                /* indique si le nouvel article est insere ou non */
int n;
int disp;
int i;
int cpt;                /* buffer pour le compteur du nbre de destructions */
int *pt;

struct temp {                   /* buffer pour le fichier index */
            int nb;             /* numero de message */
            char d[3];          /* date d'envoi */
            int p;              /* pointeur vers le header */
          }btemp;


if(wora == 'w')
            fd1 = open(windex,2);
else
            fd1 = open(aindex,2);
```

```
bindex.nbre = bufhead.nber;
pt = localtime(bufhead.datheur);          /* mettre la date sous forme : */
bindex.date[0] = *(pt+5);                    /* aa */
bindex.date[1] = *(pt+4);                    /* mm */
bindex.date[2] = *(pt+3);                    /* jj */
maketemp();              /* composer un nom de fichier de travail */
fd2 = creat(wrkind,0777);
read(fd1,&cpt,2);
write(fd2,&cpt,2);
mod = 0;                 /* initialisation : nouvel article non encore insere */

/* Tant qu'il n'y a pas eof :
        lire un article;
        si le numero lu est plus grand que le numero de l'index
        a inserer et que le nouvel article n'est pas encore insere :
                ecrire le nouvel index dans le fichier de travail;
                l'indice du nouvel index = indice courant;
        ecrire l'article lu dans le fichier de travail;
        incrementer l'indice de l'index courant;
*/

while((n = read(fd1,&btemp,LINDEX)) == LINDEX)
        {
        if((mod == 0) && (btemp.nb > bindex.nbre))
                {
                write(fd2,&bindex,LINDEX);
                mod = 1;
                }
        write(fd2,&btemp,LINDEX);
        }

/* si le numero du nouvel index est plus grand que le plus grand : */

if(mod == 0)    write(fd2,&bindex,LINDEX);
close(fd1);
close(fd2);

/* copier le fichier de travail dans le fichier index */

if(wora == 'w'){
                unlink(windex);
```

```
                        link(wrkind,windex);
                        }
        else
                        {
                        unlink(aindex);
                        link(wrkind,aindex);
                        }
        unlink(wrkind);
        }



/*
*
*
*       Destruction d'un message dans les fichiers body , head , index
*       courants .
*
*
*/

delmess(wora)
char wora;                          /* en attente ou archive ? */
        {
        extern struct header {
                int ptbody;                 /* pointeur vers le corps */
                int lmess;                  /* longueur du corps */
                int nber;                   /* numero du message */
                int datheur[2];             /* date & heure d'envoi */
                char sender[9];             /* emetteur du message */
                char ioru;                  /* important ou urgent ? */
                char secret;                /* secret ? */
                char subject[51];           /* sujet du message */
                 } bufhead;
        extern struct index {
                int nbre;                   /* numero du message */
                char date[3];               /* date sous la forme aammjj */
                int pthead;                 /* pointeur vers le header */
             } bindex;
        extern char wfhead[];
```

```
        extern char afhead[];
        extern char wfbody[];
        extern char afbody[];
        extern int fhd;

        int nbloc;                  /* nbre de bloc dans le corps */
        int wrknum;

        delindex(wora);
        seek(fhd,-LHEAD,1);
        wrknum = bufhead.nber;
        bufhead.nber = ZERO ;
        write(fhd,&bufhead,LHEAD);
        bufhead.nber = wrknum;
        nbloc = bufhead.lmess / LBLOC + 1;
        if(wora == 'w')
                        {
                        makefree(wfhead,1,bindex.pthead);
                        makefree(wfbody,nbloc,bufhead.ptbody);
                        }
        else
                        {
                        makefree(afhead,1,bindex.pthead);
                        makefree(afbody,nbloc,bufhead.ptbody);
                        }
        }


delindex(wora)
char wora;
        {
        extern fdi;
        extern char windex[];
        extern char aindex[];
        extern char work[];


        extern struct header {
                int ptbody;             /* pointeur vers le corps */
                int lmess;              /* longueur du corps */
```

```
        int nber;               /* numero du message */
        int datheur[2];         /* date & heure d'envoi */
        char sender[9];         /* emetteur du message */
        char ioru;              /* important ou urgent ? */
        char secret;            /* secret ? */
        char subject[51];       /* sujet du message */
         } bufhead;
extern struct index {
        int nbre;               /* numero du message */
        char date[3];           /* date sous la forme aammjj */
        int pthead;             /* pointeur vers le header */
     } bindex;
int mod;                /* indique si destruction a eu lieu ou non */
int cpt;                /*buffer pour le compteur du nbre de delete */
int fd2;                /* file descriptor du fichier de travail */
int iread;              /* indice de l'article courant */
int index;              /* indice de l'article a detruire */
int iwrite;             /* si tassement , indice de l'article suivant
                           l'article detruit */
int used;               /* si = 1 : le fichier est consulte actuellement
                           sinon : = 0 */
int wrknum;             /* pour sauver bindex.nbre */
int wrkpt;              /* pour sauver bindex.pthead */

if(bufhead.nber == bindex.nbre) used = 1;
else
                                used = 0;

mod = 0;
iread = 1;

seek(fdi,0,0);
read(fdi,&cpt,2);

/* Lire le fichier d'index jusqu'a ce que bufhead.nber == bindex.nbre
     Mettre a 0 le numero de l'index trouve
        Incrementer le cpt de destructions
*/

while((read(fdi,&bindex,LINDEX) == LINDEX) && (mod == 0))
        {
```

```
        if(bufhead.nber == bindex.nbre)
                {
                wrkpt = bindex.pthead;
                seek(fdi,-LINDEX,1);
                wrknum = bindex.nbre;
                bindex.nbre = ZERO;
                write(fdi,&bindex,LINDEX);
                bindex.nbre = wrknum;
                mod = 1;
                index = iread;
                cpt++;
                }
        iread++;
        }

/* si plus de 10 destructions , retasser le fichier */

if(cpt > 10)
        {
        /* creer un fichier de travail pour recopier les index dont
           le numero n'est pas = 0 */

        maketemp();
        fd2 = creat(work,0777);

        /* mettre a 0 le cpt du nbre de destructions */

        cpt = 0;
        iwrite = 0;
        write(fd2,&cpt,2);
        seek(fdi,2,0);

        /* Pour chaque article lu ds le fichier d'index :
                - si bindex.nbre != 0 : recopier bindex ds le
                  fichier de travail */

        while(read(fdi,&bindex,LINDEX) == LINDEX)
                {
                if(bindex.nbre != ZERO) write(fd2,&bindex,LINDEX);
                if(index > 0) iwrite++;
                index--;
```

```
                }
        close(fdi);
        close(fd2);


        /* copier le fichier de travail ds le fichier d'index */

        if(wora =='w') {
                        unlink(windex);
                        link(work,windex);
                        unlink(work);
                        fdi = open(windex,2);
                        }
        else
                        {
                        unlink(aindex);
                        link(work,aindex);
                        unlink(work);
                        fdi = open(aindex,2);
                        }

        /* si le fichier d'index etait utilise , se positionner sur
          l'index suivant l'index detruit */

        if(used) seek(fdi,2 + (iwrite*LINDEX),0);
        }

/* nbre de destructions > 10 */

else
        {
        seek(fdi,0,0);
        write(fdi,&cpt,2);
        if(used) seek(fdi,index*LINDEX,1);
        }
bindex.pthead = wrkpt;
}
```

```
/*
*       Lecture d'un article dans le fichier index dont le file descriptor
*       est fd .
*/

readindex(fd)
int fd;
        {

        extern struct index {
                int nbre;                       /* numero du message */
                char date[3];                   /* date sous la forme aammjj */
                int pthead;                     /* pointeur vers le header */
             } bindex;
        if(read(fd,&bindex,LINDEX) == LINDEX) return(1);
        return(0);                      /* eof */
        }



/*
*       Lecture sequentielle d'un article dans le fichier head dont le
*       file descriptor est fd .
*/

readhead(fd)
int fd;
        {

        extern struct header {
                int ptbody;                     /* pointeur vers le corps */
                int lmess;                      /* longueur du corps */
                int nber;                       /* numero du message */
                int datheur[2];                 /* date & heure d'envoi */
                char sender[9];                 /* emetteur du message */
                char ioru;                      /* important ou urgent ? */
                char secret;                    /* secret ? */
                char subject[51];               /* sujet du message */
                 } bufhead;
        if(read(fd,&bufhead,LHEAD) == LHEAD) return(1);
```

```
        return(0);                  /* EOF */
        }



/*
*       Lecture directe dans le fichier head dont le file descriptor est fd .
*       L'acces se fait grace au pointeur se trouvant dans l'index correspondant .
*/

readirhead(fd)
int fd;
        {

        extern struct index {
                 int nbre;                  /* numero du message */
                 char date[3];              /* date sous la forme aammjj */
                 int pthead;                /* pointeur vers le header */
              } bindex;
        extern struct header {
                 int ptbody;                /* pointeur vers le corps */
                 int lmess;                 /* longueur du corps */
                 int nber;                  /* numero du message */
                 int datheur[2];            /* date & heure d'envoi */
                 char sender[9];            /* emetteur du message */
                 char ioru;                 /* important ou urgent ? */
                 char secret;               /* secret ? */
                 char subject[51];          /* sujet du message */
                  } bufhead;
        seek(fd,bindex.pthead*LHEAD,0);
        read(fd,&bufhead,LHEAD);
        }
```