Les Processus

Les processus encore appelés tâches ou job sont l'unité d'exécution et d'ordonnancement d'un système d'exploitation et constitue son activité de base.

1. Expliquer les deux expressions unité d'exécution : signifie que la plus petite entité exécutable gérée par le système d'exploitation est le processus. **Unité d'ordonnancement** : signifie que fréquence avec laquelle le système d'exploitation alloue le processeur à chaque fois à un processus entier.

2. quelle est la différence entre un programme et un processus ? Un processus est dynamique alors qu'un programme est statique

3. Comment le système d'exploitation gère-t-il ces notions ? Les programmes et les processus de la manière suivante : -- Les programmes sont des fichiers exécutables donc ils sont gérés comme tout autre fichier. Le système d'exploitation charge les programmes en mémoire, les décode, initialise le compteur ordinal et déclenche son exécution. -- Les processus sont enregistrés dans une table en mémoire à leur création ayant une entrée pour chaque processus présent tout au long de son exécution. L'activité du processus est suivie dans cette table. A la fin de son exécution l'entrée du processus dans la table est supprimée.

4. dans un contexte de multiprogrammation, dans quel état un processus se trouve-t-il la plupart du temps ? En multiprogrammation, un processus passe la plupart de son temps à l'état prêt. Ceci parce que le processeur passe un temps non négligeable dans les activités de commutation de contexte, alors chaque processus aura en moyenne 1/n du temps d'exécution totale.

5. Lorsqu'un processus n'est pas en cours d'exécution, que deviennent les valeurs de ses registres matériels au moment de sa dernière exécution ?Les valeurs de ses registres matériels sont enregistrées dans le mot d'état du processus et dans la table des processus. Parce que le processus pour s'exécuter après aura besoin de son contexte contenu dans les valeurs des registres.

6. Une application sur un réseau offre des services à des utilisateurs sur un réseau. L'application consiste en un programme serveur sur une machine, qui tourne indéfiniment en attente de clients. Lorsqu'un client sollicite le serveur, celui-ci engendre un fils, qui prend en charge les requêtes de ce client. Le père retourne à son attente perpétuelle (c'est pourquoi on l'appelle démon).

a. expliquer pourquoi il est important que chaque client dialogue avec un processus serveur autonome au lieu que le démon gère toutes les communications. Lorsque le seul démon gère toutes les communications, il peut devenir très lent et donc ne pas réagir à certaines requêtes. De plus le code du démon devient très compliqué à écrire et à tester.

b. quel problème peut-on néanmoins avoir avec ce schéma d'exécution ? Lorsqu'on dédie un processus autonome pour chaque client, la consommation des ressources systèmes devient très vite importante au point de pouvoir bloquer le système.

c. la machine serveurs tourne sous linux. Que partagent les différentes instances du serveur ? Ils partagent en commun le code source, ceci pour la simple raison qu'ils exécutent tous le même programme.

- **Gérer la mémoire :**

La gestion de la mémoire vise 3 objectifs essentiels : ---allocation de la ressource critique mémoire aux processus en compétition en cherchant à maximiser son

utilisation ---donner une abstraction convenable aux processus et aux compilateurs --- réaliser une isolation complète des différents processus.

2. pour le cas du système Linux, dire en expliquant lequel des 3 objectifs ci-dessous est visé par les concepts ci-après ?

a. copie à l'écriture : objectif 1 puisque la copie à l'écriture permet d'économiser la mémoire en partageant la mémoire lorsque plusieurs processus utilisent les mêmes données et ne dupliquent que les zones privées modifiées.

b. mémoire virtuelle : objectif 2 puisque, la mémoire virtuelle donne l'impression au programme d'avoir un espace mémoire très grand

c. Swap :objectif 1 puisque le swap permet au système de libérer l'espace en mémoire physique au profit des processus immédiatement

d. pagination à la demande :objectif 1 puisque les pages ne sont crées que lorsqu'un processus fait référence à une adresse qui n'est ni en mémoire ni sur le swap.

3. pourquoi la mémoire est-elle une ressource critique ?

Parce que les processus ont besoin chacun d'une zone de mémoire privée et qu'ils ne se soucient pas de connaitre l'existence des autres processus

4. qu'est-ce que l'espace d'adressage d'un processus ? Ensemble des adresses (données et instructions) de ce processus. **Que contient-il ?** Le code et les données du processus. **Peut –il évoluer ?** Oui **Comment ?** Notamment si le processus demande ou libère des zones d'allocation dynamiques.

5. soit le fragment de programme pascal ci-dessous

Var k : array [1..128] of array [1…128] of integer; for k:=1 to 128 do k[i][j] :=0;

Les entiers sont codes sur un mot (32 bits) et on dispose d'une mémoire virtuelle avec des pages de 128. La mémoire physique n'a qu'un seul cadre de page. Combien de défauts de page ce calcul génère-til ?Et la matrice A est stockée.

A chaque pas i de la boucle externe correspondant à une ligne, le programme parcourt toutes les colonnes de la matrice sur cette ligne. Donc si la matrice est enregistrée ligne par ligneet que le cadre de page contient 128mots, toute une ligne sera en mémoire physique et on pourra accéder à tous les 128 éléments de cette ligne de la matrice. On aura donc un défaut de page seulement quand il faudra passer o la ligne suivante. Ce calcul génère par conséquent 128 défauts de page, en considérant que le chargement de la première ligne est dut o un défaut de page.

6. Qu'est-ce que l'espace d'adressage d'un processus ?Ensemble des adresses (données et instructions) de ce processus. **Que contient-il ?**Il contient le code et les données du processus. **Peut-il évoluer ?** Il peut évoluer notamment si le processus demande ou libère des zones d'allocation dynamiques.

Système de fichier

Les mémoires secondaires sont typiquement tout ce qui est en dehors de la << mémoire principale >>. Elles ne permettent pas l'exécution direct des instructions, ou l'utilisation des données par les instructions machines du type chargement / rangement. Elles sont de grandes capacités, moins chères, les données y sont persistantes, mais lentes.

On demande de répondre aux questions suivantes en rapport avec le système de fichiers :

1. A votre avis pourquoi les mémoires secondaires sont-elles lentes ? Les mémoires

secondaires sont lentes parce que d'une part, la lecture se fait par des procédés mécaniques (rotation ou translation de bras) qui sont très lents, et d'autre part parce qu'elles sont éloignées du processeur, ce qui fait plus de distances à parcourir pour les signaux.

2. Comment la solution générique du cache est-elle utilisée pour ce problème ? Pour savoir comment la solution générique du cache est utilisée pour ce problème, Commençons tout d'abord par définir ce qu'on appelle mémoire cache : La cache est une mémoire à accès rapide et de plus petite taille, située entre le processeur et la RAM, son ultime avantage est d'accélérer les opérations. Par ailleurs, la cache est une mémoire qui stocke de façon temporaire les données susceptibles d'être utilisées ou réclamées pour éviter de les rechercher en mémoire d'origine.

3. Donner deux fonctions du système d'exploitation relatives a la gestion des disques. Donnons deux fonctions du système d'exploitation relatives à la gestion des disques :---Cacher la complexité du matériel : initialisation, architecture physique, …---L'abstraction de fichiers : considère les disques comme un dépôt de fichier avec une certaine organisation.

4. Au lieu de typer les fichiers, les vielles versions de MA OS enregistraient le nom du programme créateur dans le nom de fichier. Pourquoi ? Les vielles versions de MA OS enregistraient le nom du programme créateur dans le nom de fichier au lieu de typer les fichiers dans la mesure de permettre la manipulation du fichier par ce programme. La plus part des fichiers ne connaissant pas la notion de type de fichier, seul le programme créateur en connaît la signification

5. Discuter des inconvénients de cette stratégie. Discutons des inconvénients de cette stratégie ; L'inconvénient de cette stratégie est que l'on ne peut pas utiliser un autre programme (même similaire) pour la manipulation du fichier.

6. Dans la plus part des systèmes modernes, un répertoire est juste un fichier spécial contenant des métadonnées, en particulier la liste des fichiers. Cette liste n'est généralement pas triée, alors les programmes d'affichage doivent les faires eux-mêmes. Dire, sous Linux comment le système réalise l'appel.

fd = open(''/ceci/est/un/fichier'',O_RDWR),

où O_RDWR signifie une demande d'accès en lecture et écriture. Disons, sous Linux

comment le système réalise l'appel : Ce système vérifie d'abord si l'utilisateur a les droits pour les opérations qu'il sollicite, ensuite ouvre le fichier s'il existe, Sinon il le crée :

Dans le cas ou le fichier n'existe pas, alors il y a :---Accès à l'inode du répertoire racine/ ; ---Vérifications des droits en ouverture pour l'UID du processus en cours ; ---Recherche du nom du fichier dans cet Inode ;---Accès à lnode associé à ceci.

Dans le cas ou le fichier existe :---Le système vérifie les droits en lecture et écriture sinon il fait la vérification des droits en écriture sur le répertoire un (01), puis création d'une inode et création d'un autre répertoire dans un (01) pour fichier ;---Positionnement de l'index dans le fichier

7. Dans quelles conditions cet appel échoue-t-il ? Cet appel échoue dès que l'une des vérifications mentionnée ci-haut échoue

1Quelles est la déférence entre un système batch et un système en temps partagé ?
Un système batch doit attendre la fin d'une tâche avant d'en commencer une autre alors qu'un système en temps partagé partage le temps processeur en changeant de contexte lors des entrées/sorties ou à la fin d'un quantum de temps.

2 Quelles est la différence entre un programme et un processus ?
Un programme est statique alors qu'un processus est dynamique. Un processus est un "programme en cours d'exécution".

3 Qu'est-ce que le bloc de contexte ?
C'est une structure de données associée `a chaque processus, dans laquelle le système d'exploitation mémorise les informations utiles `a la gestion du processus (identifiant, état, registres, mémoire allouée, table des pages, emplacement et taille de la pile, les E/S utilis´ees...).

4 Quelles sont les di□érentes zones mémoires associées `à un processus ? Quelle est la fonction de chacune d'entre elles ?
- Pile (LIFO) : stocke les variables temporaires (variables locales, paramètres de fonctions, ...)--- Tas : utilisé pour les variables allouées dynamiquement (malloc)--- Données statiques : stocke les variables statiques (espace m´emoire connu `a la compilation)--- Code : contient le programme `a exécuter--- Registres : "contexte de l'exécution du processus" : CO,SP, etc....

5 Qu'est-ce qu'un thread ?
C'est un processus "allégé", dont la conception est à la charge du programmeur, qui permet un parallélisme efficace grâce à un partage des ressources d'une même tâche.

6 Quelles sont les zones mémoires partagées entre les threads d'une même tâche, et celles qui ne le sont pas ?
- Partagés : tas, données statiques, code -- Non partagés : Pile, registres

7 Soit une ressource multiple composée de 3 lecteurs/graveurs de CD, et un programme de duplication de CDs. Ce programme nécessite l'utilisation de 2 lecteurs/graveurs pour pouvoir fonctionner. Le programme peut se résumer au pseudo-code suivant :

- Lgr1 ← getLecteurGraveur() ; // demande de réservation du premier lecteur/graveur
- Lgr2 ← getLecteurGraveur() ; // demande de réservation du second lecteur/graveur
- copie(Lgr1,Lgr2) ; // copie du contenu du lecteur/graveur Lgr1 vers Lgr2
- free(Lgr1); // libération du premier lecteur/graveur
- free(Lgr2); // libération du second lecteur/graveur

Où est la section critique de ce programme ? Section critique (section ininterruptible) : a-b
Qu'elle doit être la condition d'entrée dans cette section critique ? Condition d'entrée : il doit rester au moins 2 lecteurs graveurs disponibles.

2 Gestion mémoire

8 Soit un espace mémoire de 800Mo géré par un système de gestion mémoire à partitions.
Supposons que 6 processus de 100Mo soient exécutés, et que les processus 2 et 4 se terminent.
- Quelle est alors la mémoire disponible ? - 400Mo
- Un programme de 300Mo peut-il s'exécuter ? - Non car le plus grand espace contigüe est de 200Mo
- Comment remédier à ce problème ? - Fragmentation → Systèmes `a pagination

9 : Un système utilise des adresses virtuelles sur 48 bits et des adresses physiques sur 32 bits. Les pages font 8ko (les mots mémoires sont de 1 octet).
- Combien y a-t-il d'entrées dans la table des pages ? - Pages de 8ko → 2^{3}□2^{10} = 2^{13}mots. Il reste 48-13=35 bits. Le nombre de pages est donc de 2^{35} = 32Giga pages !
- Quelle solution proposez-vous pour réduire la taille de la table ?
- Faire une table des pages `a 2 niveaux : par ex Un champ de 17 bits pour le premier niveau (table de taille 217), Et un champ de 18 bits pour le deuxième niveau (217 tables de taille 218).
10 Voici le schéma du début d'une table des pages (pages de 64ko) d'un processus :

Numéro de page	Accès	Adresse physique
0	RWX	12340000
1	RX	12350000
2	RW	12360000
3	R	10000000
4	–	Absente
5	RX	44440000
6	R	20000000
7	–	Absente
8	–	Absente
9	X	40000000
A	RW	43210000
B	–	Absente
C	–	Absente
D	•	Absente
E	R	43220000
F	–	Absente

Traduisez les adresses suivantes (en donnant l'erreur s'il y en a une et le type d'accès) :
1. 00001232h
2. 00055554h
3. 000A2240h
4. 000B3240h
5. 000C2330h
6. 00040000h
7. 00050560h
8. 00070220h
9. 00080020h

10. 00030320h

1. 12341232h - RWX
2. 44445554h - RX
3. 43212240h - RW
4. Faute de page
5. Faute de page
6. Faute de page
7. 44440560h - RX
8. Faute de page
9. Faute de page
10. 10000320h - R

EXERCICE : Ils reviennent … comme promis

La commande linux Kill qui permet d'envoyer un signal a un processus du système, connaissant sonPID. Une version plus sympathique est killall quine vous oblige pas a connaitre le **PID**, mais seulement la ligne de commande du processus. Dans les deux cas, il faut donner le code signal après l'option **–S**. soit le programme **C** suivant qui enregistre l'heure au moment ou il est lancé, s'endort ensuite, et calcul la durée de son sommeil a son réveil et l'affiche :

```
main(){
int i ; time_t t1, t2
(void) time(&t1) ;
kill (getpid(),SIGSTOP) ;
(void)time(&2);
printf("\n %d secondes \n",(int) t3-t1) ;
}
```

Soit le script ci-dessous :

```
# ! /bin/sh
./temps &
for source in `ls *.c´
do
       echo accorder pour $source ?
       read reponse
        if [ reponse = oui ]
        then
              chmod +r $source
      fi
done
Killall     -s  SIGCONT   temps
```

1. Donnons la signification de chacune des trois commande de la boucle for
Dans la boucle for, nous avons les commandes echo, read , chmod.
---Echo "accorder pour" $source : affiche " accorder pour " et ajoute à cela les paramètres qui lui sont passés sur la sortie principale---Read reponse: lit une ligne sur l'entrée standard et l'affecte à la variable reponse.---Chmod : change le droit d'accès

2. La commande en entrée « ls * .c »indique à ls de lister tous les fichiers dont le nom se termine par « .c ». Dans source in ` ls * .**c ´ indique** à ls de lister tous les fichiers dont le nom se termine par « .c » et de les ranger dans la variable source.
3. La commande en sortie « chmod +r source » autorise le droit d'accès en lecture du fichier se trouvant dans la variable source.
4. Le script lance le programme en c nommé temps et enregistre le temps du début de et son sommeil et le réveille à la demande de l'utilisateur.

5. Avec la commande kill, on réalise ceci :

```
# ! /bin/sh
./temps &
for source in `ls *.c´
do
       echo accorder pour $source ?
       read reponse
        if [ reponse = oui ]
        then
              chmod +r $source
      fi
done
kill  -s   ( getpid ( ), SIGCONT)
```

Petit problème :

Un système combine la segmentation (chaque division logique du programme est logée dans un segment distinct), la pagination, et le support matériel pour la mémoire. Lorsqu'on objet est référencé en mémoire, on vérifie la nature de l'objet (instruction, donnée) et la compatibilité de l'opération (lecture / écriture) avec le segment correspondant. L'adresse physique de l'objet est enfin calculée par le mécanisme de gestion des pages.
1. Rappeler le rôle du matériel dans une telle organisation de mémoire. Le

matériel dans une telle organisation de mémoire a pour rôle de : ---Calculer les adresses physiques,---Vérifier les cohérences des accès mémoires,---Vérifier la cohérence des opérations (lecture / écriture).

2. Dire en quoi consiste une erreur de violation mémoire dans ce système. Dans ce système, une erreur de violation mémoire consiste à accéder à un espace mémoire non dédié (c'est-à-dire lorsque l'adresse générée par la mémoire est hors de la limite réservée au segment).

3. Dire en quoi consiste une erreur de défaut de page dans ce système. Dans un tel système, une erreur de défaut de page consiste en une demande de page non référencée en mémoire (c'est-à-dire une instruction d'une page chargée en mémoire fait référence d'une donnée ne faisant pas partir de la page).

4. Dire, pour les questions 2 et 3, comment le système gère le déroutement.

Si nous considérons, dans le sens général du terme un « déroutement » comme étant une situation exceptionnelle dans laquelle une erreur liée à une instruction provoque une interruption, alors nous pouvons ainsi dire de cette définition, que le système gère le déroutement comme suit :

---Il va en mémoire et fait un remplacement de la page non dédiée pour qu'elle soit référencée, ---Il organise la mémoire de façon à rendre cohérent les accès, ---Il arrête le processus en cours d'exécution et envoie un message d'erreur.

5. Quelle sont les actions menées par le système d'exploitation lors du remplacement d'un processus par un autre dans l'UC ? Lors du remplacement d'un processus par un autre dans l'UC, les actions menées par le système sont : ---La sauvegarde de contexte du processus sortant,---Déclenchement de l'ordonnancement (c'est-à-dire la recherche d'un autre processus ayant la plus haute priorité dans la file d'attente).---La charge du processus ayant accès au processeur (c'est-à-dire ce processus a la main du processeur).

6. Si l'espace d'adressage virtuel se compose de 8 segments pouvant chacun atteindre une longueur de octets, et contenant des pages de 4ko. Combien de bits de l'adresse virtuelle spécifient :

a. Le numéro de segment ? Le nombre de bits de l'adresse virtuelle spécifient le numéro de segment : Cherchons les proches du nombre de segment où *n* représente ce nombre de bits. On a : . Donc on a ***3 bits d'adresse virtuelle spécifient le numéro de segment***.

b. Le numéro de page ? Le nombre de bits de l'adresse virtuelle spécifient le numéro de page : Si pour un segment de longueur octets, contenant des pages de octets; alors nous avons la relation suivante : . Ce qui donne 17 bits d'adresse virtuelle spécifient le nombre de page.

c. L'offset au sein de la page ? Le nombre de bits de l'adresse virtuelle spécifient l'offset au sein de la page : De la relation : . D'où l'obtention de ***12 bits d'adresse virtuelle spécifie l'offset au sein de la page.***

Nombre de bits adresse virtuelle entière = Nombre de bits segment + nombre de bit page + nombre de bit offset

d. L'adresse virtuelle entière ? Le nombre de bits de l'adresse virtuelle spécifient l'adresse virtuelle entière: Pour ce nombre, il suffit de faire :

C'est-à-dire on a :***Nombre de bits adresse virtuelle entière = 3+17+12=32bits.***

CC N°1

1- Définitions :

- **Logiciel libre :** Un **logiciel libre** est un logiciel dont l'utilisation, l'étude, la modification et la duplication en vue de sa diffusion sont permises, techniquement et légalement, afin de garantir certaines libertés induites, dont le contrôle du programme par l'utilisateur, et la possibilité de partage entre individus.
- **Open source** : qui signifie code source ouvert sont des logiciels dont la licence respecte des critères précisément établis par l'*Open Source Initiative*, c'est-à-dire la possibilité de libre redistribution, d'accès au code source et aux travaux dérivés.

Discussion

Un logiciel est libre selon la Free Software Foundation s'il confère à son utilisateur quatre libertés :

- la liberté d'exécuter le programme, pour tous les usages, --la liberté d'étudier le fonctionnement du programme et de l'adapter à ses besoins, --la liberté de redistribuer des copies du programme (ce qui implique la possibilité aussi bien de donner que de vendre des copies), --la liberté d'améliorer le programme et de distribuer ces améliorations au public, pour en faire profiter toute la communauté.

Une open source donne la possibilité à l'utilisateur d'adapter le logiciel à ces besoins en modifiant le code source. En même temps il a le droit de redistribuer le logiciel modifié ou les différentes versions crées.
Open Source en référence à l'*Open Source Definition* introduite par l'*Open Source Initiative* (OSI) en 1998, qui souhaitait une autre terminologie pour les logiciels libres, qui se voulait en anglais moins ambiguë et plus adaptée au monde des affaires que *Free Software*.

Le mouvement pour le logiciel libre a défini des règles sur des principes éthiques ; celui pour l'open source (qui en découle) a proposé une traduction fonctionnelle ; cela a déclenché un temps quelques différends relatifs au respect de ces principes. Les défenseurs du logiciel libre considèrent que le logiciel libre est une affaire de philosophie, tandis que les partisans de l'open source rejettent toute philosophie.
Cette initiative a causé une controverse avec Richard Stallman et la *Free Software Foundation* qui regrettaient la mise en avant des principes techniques aux dépens de l'éthique. Richard Stallman explique aussi pourquoi le logiciel libre est meilleur que l'open source et pourquoi l'« open source » passe à côté du problème que soulève le logiciel libre.
S'il persiste des désaccords entre ces mouvements, ils restent très proches et un travail conjoint d'harmonisation fait que les définitions officielles du logiciel libre par la *Free Software Foundation* et de l'open source par l'*Open Source Initiative* renvoient dans la pratique aux mêmes licences, à quelques rares exceptions près (versions 1.*x* de l'*Apple Public Source License* par exemple).

Le terme logiciel libre confère aussi les libertés d'étudier et améliorer un logiciel supposent un accès au code source du logiciel. L'accès au code source est important car les logiciels sont généralement distribués sous une forme compilée en langage

machine, prêts à être exécutés par un ordinateur. Mais le langage machine est très peu lisible et rend l'étude du logiciel excessivement pénible. L'accès au code source a donné lieu à la notion d'*Open Source* (code source ouvert).

2. la réentrance dans un système Linux permet d'économiser la mémoire pour l'exécution des processus.
La réentrance est : l'exécution d'un code pour le compte de plusieurs processus : **donner un exemple de contexte de mise en œuvre sous linux ?** : Bibliothèque partagée

3. la stratégie de copie à l'écriture est une autre technique qui permet à Linux d'optimiser l'utilisation de la mémoire. Dire comment elle fonctionne. C.O.W : on copie le contexte du processus du père pour le fils que si ce dernier demande à modifier les données
L'appel système *vfork(2)* du système d'exploitation Linux crée un nouveau contexte (processus) en dupliquant la table des pages du processus qui fait l'appel (son *père*). La partie de la table des pages marquée en lecture seule (le code) sera dupliquée telle quelle. Les pages qui correspondent aux données sont marquées *copy on write*. Quand Linux devra effectuer une écriture sur une page marquée *copy on write*, il allouera une nouvelle *frame*, recopiera le contenu de la *frame* originale et enfin fera la modification demandée sur cette nouvelle *frame*. Préservant ainsi les données telle qu'elle au cas où un processus aura besoin de les utiliser.
4. pour réaliser l'uniformité des accès aux périphériques et fichiers, Linux met en œuvre le VFS
VFS : Virtual File System système de fichier virtuel qui est ensemble de primitives du SE permettant de manipuler les fichiers indépendamment de leurs types, de leurslocalisations et du matériel utilisé. Il a essentiellement pour rôle de contrôler la cohérence des opérations d'accès aux périphériques et communique avec les pilotes de périphériques pour l'exécution proprement dite des opérations réalisables sur le fichier (nommage, protection...etc).
Il n'est responsableni de la lecture de bloc de données, transformation des enregistrements en suite de caractère, vérification de la syntaxe d'une instruction d'E/S, capture des instructions pour le compte d'un processus

5. comment Linux réalise-t-il la protection des données des utilisateurs d'un système ?
Les droits sur un fichier Linux s'attribuent sur trois « actions » différentes possibles : --le droit de lecture (r) --le droit d'écriture (w) --le droit d'exécution (x).
Seuls *root* et le propriétaire d'un fichier peuvent changer ses permissions d'accès. On écrit côte à côte les droits r, w puis x respectivement pour le propriétaire (u pour user), le groupe (g pour group) et les autres utilisateurs (o pour others).
Les codes u, g et o sont utilisés par les commandes Linux pour établir l'appartenance des fichiers. On écrit respectivement côte à côte (r, w ou x) quand il n'est pas attribué on écrit un « - » par exemple :

```
rwxr-xr--
 \ /\ /\ /
  v  v  v
  |  |  droits des autres utilisateurs (o)
  |  |
  |  droits des utilisateurs appartenant au groupe (g)
```

|
droits du propriétaire (u).

Linux utilise donc 9 bits pour réaliser la protection de données à savoir les trois premiers bits pour l'utilisateur, les trois autres pour le groupe et les trois derniers pour le reste du monde.
On a par exemple
111 111 111 soit r w x r w x r w x : l'utilisateur a le droit de lecture, écriture et exécution ; le groupe ainsi que le reste du monde ont les droits de lecture, écriture et exécution

110 000 101 soit r w - - - - r – x : l'utilisateur a le droit de lecture, écriture mais pas d'exécution ; le groupe n'a pas les droits de lecture, écriture et exécution et le reste du monde a les droits de lecture et exécution mais pas d'écriture.

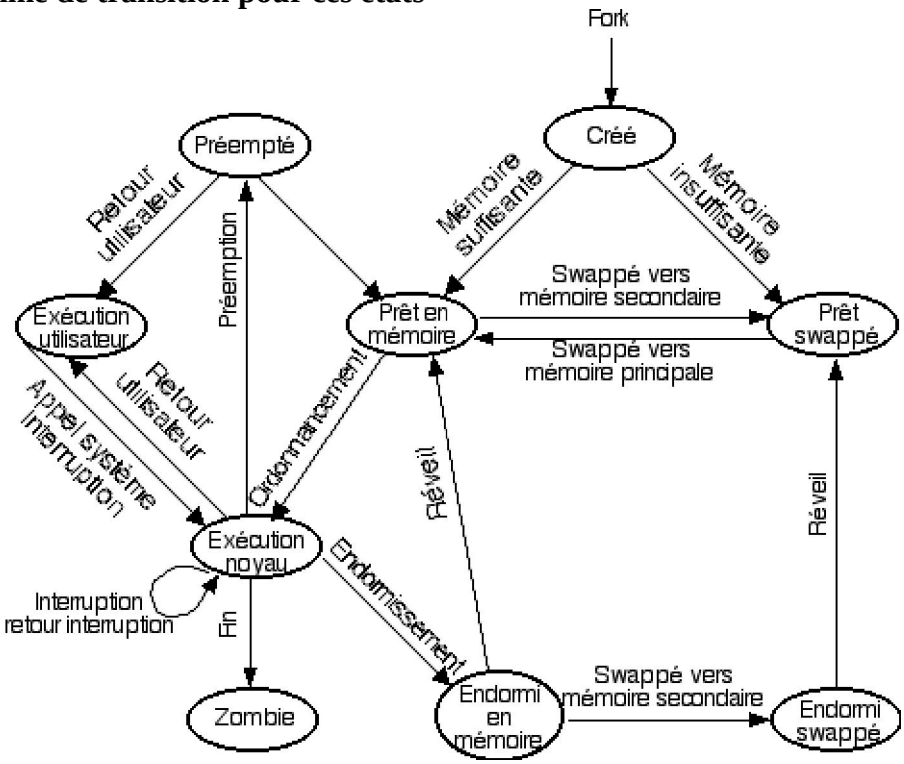
Discussion
Tous les utilisateurs Linux n'ont pas les mêmes droits d'accès à l'ordinateur (ils ne peuvent pas tous faire la même chose), et ceci simplement pour des raisons de sécurité et d'administration. Par exemple, pour éviter tout problème sur Internet, l'utilisateur qui gère le serveur HTTP n'a pas le droit d'exécuter des commandes localement, pour éviter que le serveur ne puisse le faire.

Certains utilisateurs ne peuvent en effet pas s'authentifier sur l'ordinateur et accéder à un interpréteur de commandes. Cela ne veut toutefois pas dire qu'ils ne peuvent rien faire sur l'ordinateur : il leur est possible de lire ou écrire des fichiers mais cela nécessite que le super-utilisateur démarre un programme pour cet utilisateur. Ce mécanisme est généralement utilisé pour les démons : le super utilisateur démarre le démon et pour éviter que ce dernier ne puisse faire tout et n'importe quoi sur la machine, il est par exemple attribué à l'utilisateur bin.
Sur tout système Linux, il y a un super-utilisateur, généralement appelé *root*, qui a tous les pouvoirs. Il peut accéder librement à toutes les ressources de l'ordinateur, y compris à la place d'un autre utilisateur, c'est-à-dire sous son identité. En général, du moins sur les systèmes de production, seul l'administrateur système possède le mot de passe *root*. En clair le système Linux est d'une grande efficacité dans la protection des données.

6. comment Linux réalise-t-il la protection desprogrammes les uns des autres pendant leur exécution ?
La division interne par zone de mémoire virtuelle permet d'établir des sécurités d'accès au niveau du code et des données du programme on distingue : --code qui est en lecture seule mais peut être partagé --données statiques qui sont en lecture seule mais privées. --Données générales qui sont en lecture/écriture mais privées. --La pile qui est en lecture écriture et peut augmenter de taille. --Tas constitué des données dynamiques.
7. qu'est-ce qu'une inode ? Encore i-nœuds, ce sont des structures utilisées par certains systèmes pour stocker les informations relatives à un fichier excepté son nom.
Quel rapport y-a-t-il entre un inode et un nom de fichier ? Le rapport est que lorsque le système stocke les informations relatives à un fichier, ne prend pas en compte son nom car le nom d'un fichier n'est qu'un lien vers celui-ci.
8. qu'est-ce que un module sou Linux ?: c'est un programme que l'on ajoute au

noyau du système (programme système) **utilisation courante** : les pilotes des périphériques

9. les processus sous linux peuvent êytre dans 8 états possibles. Donner le diagramme de transition pour ces états



10.quelle est la taille maximale (code et données) d'un programme sous Linux ? 4 Go. **Pourquoi ?**sur cette architechture, Tmax = 2^{32} = 4Go ; cet espace est répartie en 2 à savoir 1Go pour les données en mode noyau et 3 Go pour l'espace utilisateur. En mémoire virtuel les programmes sont adressés sur 32 bits 3 Go + 1 Go (super utilisateur, mode).

1. Le système dans l'architecture

1. Où se trouve le programme d'amorçage qui charge le système d'exploitation en mémoire lors du démarrage d'un ordinateur ? Peut-il être situé sur le même disque que le système d'exploitation ? Pourquoi ?

Le program d'amorçage se trouve dans un endroit spécifié par le bois, qui peut être un disque, une disquette, une clé USB. Il se trouve généralement dans le premier secteur (plus précisément sur la 1ére piste du disque)

2. Les architectures matérielles actuelles mettent en œuvre des mécanismes pour assister le système d'exploitation. … Dans chaque cas, dire quel est l'apport du matériel, et comment un système s'y prendrait dans une architecture non dotée de telles capacités pour résoudre le problème.

Apport du matériel en mode superviseur (met des drapeaux à jour, qui permettent de dire qu'on a chargé de mode) protégé aussi chaque espace dédié). Apport du matériel aux instructions privilégiéesrécupère les valeurs paramètres et la vérification de la validité des appels. la matériel ne fait rien pour les appels au superviseur, car il n'y a pas de changement du matériel, d'environnement, à ne pas confondre avec mode superviseur.C'est le système qui prend cette tâche et agit aussi en tant que superviseur

3. Quel est le principal avantage de la multiprogrammation ?
La multiprogrammation est appréhendée dans les SE comme étant une politique de gestion de la mémoire selon laquelle plusieurs processus peuvent résider simultanément en mémoire. Elle permet d'augmenter le taux d'allocation du processeur.
En effet, si le premier programme à exécuter possède par exemple une partie interactive, au lieu de passer le temps à attendre la frappe de l'utilisateur, le SE pourra ainsi enchaîner avec l'exécution du deuxième programme puis du troisième et ainsi de suite**.**
Quel est le principal inconvénient qui découle d'un abus de multiprogrammation ? le processeur passe plus de temps à ordonnancer les programmes qu'à les exécuter ; donc il devient lent.

4. Soit à concevoir un système d'exploitation pour un système embraqué (votre congélateur par exemple). Discuter de l'opportunité des tâches traditionnelles vues en cours pour un tel système. En voyez-vous de nouvelles qui seraient intéressantes ? Indispensables ?Réguler la température ; gérer les temps de fonctionnement et d'arrêt ; le SE vérifie que tout est correctet conforme (couche basse), les programmes des utilisateurs (besoins couche haute)
5. Définir interruption. Une interruption est une commutation du mot d'état provoquée par un signal généré par le matériel. Une interruption est un signal qui transfère le contrôle de l'unité centrale à une adresse précise et simultanément sauvegarde l'adresse en cours du programme exécuté. Ainsi, l'exploitation du programme interrompu est temporairement abandonnée, mais elle peut être reprise plus tard, si nécessaire.
En quoi sont-ils utiles pour le fonctionnement d'un système ? Pour permettre d'effectuer les entrées-sorties simultanément aux travaux de l'unité centrale, il faut pouvoir interrompre le traitement lorsqu'une opération périphérique est achevée

6. Considérer les cinq causes de commutation de contexte : interruption, appel au superviseur, appel au sous programme, déroutement, signal.
(a) Lesquels de ces évènements peuvent créer de l'indéterminisme (exécutions différentes du même programme avec les mêmes données dans un programme ? les interruptions, les signaux car ils sont externes à ce programme et donc peuvent l'influencer sans sa volonté
(b) Comparer les en termes synchronie, mode d'exécution, communication processus/traitant de l'évènement. Synchrone : appel au superviseur, appel au sous programme, déroulement qu'on peut prévoir les autres sont asynchronesMode superviseur : interruption, appel superviseur, dérouter mode utilisateur : appel au sous programme signal

(c) Quelles dispositions particulières le système d'exploitation doit-il prendre pour réaliser l'appel de procédure ?
(d) Pourquoi pensez-vous que les appels système placent leurs paramètres dans un emplacement fixe du système avant leur exécution ?

7. Parmi les instructions suivantes, dire celles qui doivent être privilégiées :(e) Réglage du mode en mode superviseur(f) Réamorçage(h) Désactivation des interruptions(i) Ecriture dans le registre instruction

8. Y a-t-il une déférence entre mode superviseur et privilège du super utilisateur ?

9. Un système d'exploitation peut mettre en œuvre un périphérique d'E/S de mémoire. Lesopérations d'E/S sur ce périphérique provoquent la lecture ou l'écriture de l'emplacementmémoire correspondant. Quel est l'inconvénient de ce type de périphérique ?
Si les utilisateurs avaient accès à un périphérique de mémoire, ils pourraient lire ou écrire sur n'importe quel emplacement de mémoire, corrompant ainsi les mécanismes de protection du système d'exploitation. Le système d'exploitation lui-même pourrait être écrasé, offrant aux programmes utilisateurs la possibilité d'exécuter desinstructions en mode superviseur

3. La mémoire

Pagination : La pagination est une technique de gestion de la mémoire selon laquelle l'espace d'adressage d'un processus est divisé en petites unités de taille fixeappelées pages.
Segment : On appelle donc segment cette division indépendante de la mémoire d'un processus. haque segment est caractérisé par son numéro et sa longueur variable.

1. Le système d'exploitation est généralement aidé du matériel pour la gestion de la mémoire : calcul d'adresse effective, protection, segmentation. Dans chacun des schémas de gestion mémoire ci-dessous, décrire brièvement les fonctions réalisées par le système d'exploitation et celle réalisées par le matériel.
Fonctions réalisées par le système d'exploitation
(a) Partition contigüe : lorsque le système est initialisé le SE charge le régistre de base avec l'adresse de l'emplacement de mémoire la plus basse à laquelle peut accéder un programme utilisatyeur
(b) Multiprogrammation à partitions variables : le SE doit tenir à jour les emplacements de mémoire qui sont utilisées et qui sont libre. Lorsque un processus est créer, pour qqu'u processus se termine le SE doit mettre à jour les données d'allocation de mémoire avant d'attribuer un processus à la mémoire.
(c) Système de zones siamoise : le SE doit tenir à jour les emplacements de mémoire qui sont utilisées et qui sont libre. lo Lorsque un processus est créer, le SE doit l'affecter à une unité d'allocation de mémoire en divisant les unités d'allocation comme il convient
(d) Pagination simple (pas de mémoire virtuelle) le SE doit tenir à jour les emplacements de page qui sont sont utilisés et ceux qui sont libres

Fonctions réalisées par le système d'exploitation
(a) Partition contigüechaque fois qu'un emplacement de mémoire est referencé, son adresse est comparé à l'adresse dans le régistre de base. Une adresse inférieure à cette dernière provoque un déroutement d'erreur de mémoire
(b) Multiprogrammation à partitions variables : chaque fois qu'un emplacement de mémoire est référencé, son adresse logique est comparée à l'adresse dans le régistre de taille. Une adresse > à l'adresse ds le régistre de taille provoque un déroulemet de défaut de mémoire
(c) Système de zones siamoise le matériel de ce système fonctionne de la même manière que celui de la multiprogrammation à partition multiple

(d) Pagination simple (pas de mémoire virtuelle) : chaque fois qu'un emplacement de mémoire est referencé, le matériel de pagination translate l'adresse logique dans l'adresse physique.
La partition contigüe utilise un registre de base contenant l'adresse de début de zone, les zones siamoises les partitions variables utilisent un registre de translation et un registre de taille. La pagination utilise les cadres de page.

2. Sur un système doté de 224 octets de mémoire et de partitions, toutes d'une taille de 65536 octets,
Quel est le nombre maximum de bits nécessaires dans une entrée da la table de processus pour enregistrer la partition à laquelle est alloué un processus ?
Combine de bits doit avoir le registre limite ?

3. Sur un système de pagination simple, l'espace logique peut-il être plus grand que l'espace physique ? Peut-il être plus petit ?de la même taille ou plus petite que l'espace d'adressage physique.

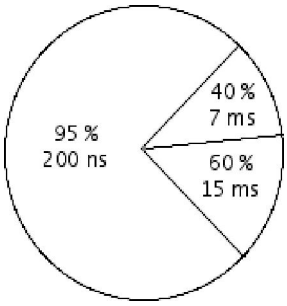
4. Soit la table de pages ci-dessous indiquant pour un processus donné sa cartographie mémoire.

bit Présence	Cadre
0	00101
1	00001
1	11011
1	11010
0	10001
0	10101
0	11000
1	00101
...	...

Si les pages sont de taille de 2^6 octets, dire parmi les adresses virtuelles suivantes, celles qui génèrent un défaut de page.
Pour celle qui ne génère pas de défaut de page, dire quelle est leur adresse physique après translation.
(a) 0000101101001la séparation de l'adresse en numéro de page et offset donne 0000101, 101001. La page 5 est out ce qui donne un défaut.
(b) 000001001001011011010010. la séparation de l'adresse en numéro de page et offset donne 0000010,010010. La page 2 utilise le cadre 11011 auquel doit être ajouté l'offset 010010 donant l'adresse physique 11011010010.
(c) 0000100010101 défaut. La séparation de l'adresse en numéro de page et offset donne 0000100,010101. La page 4 est out ce qui donne un défaut.
(d) 000000111010100001110101. la séparation de l'adresse en numéro de page et offset donne 0000001,110101. . La page 1 utilise le cadre 00001 auquel doit être ajouté l'offset 110101 donant l'adresse physique 00001110101.

5. Sur un système qui a recours à la mémoire paginée à la demande, il faut 200 ns pour satisfaire une requête mémoire si la page est en mémoire. si tel n'est pas le cas, la requête prendre 7 ms si un cadre libre est disponible ou si la page à extraire n'est pas modifiée. Il faut par contre 15 ms si la page à extraire a été modifiée.
Quel est le temps d'accès effectif si le taux de défaut de page est 5 % et que, 60 % du temps, la page à remplacer a été modifiée ? Partir du principe que le système

n'exécute qu'un seul processus et que l'UC est inactive pendant les permutations de pages.
Le temps d'accès effectif est égal à la somme pondérée par la fréquence des temps d'accès correspondant aux différents cas de figure.
Comme le taux de défaut de page est de 5 %, cela signifie que dans 95 % des cas, la page demandée est déjà en mémoire. Dans ce cas, le temps d'accès pondéré est de : **95 % x 200 x 10^{-3} = 0,19 µs**
Dans le cas où la page est en défaut (5 % des cas) et que la page a été modifiée (60 % des cas de page en défaut), le temps d'accès pondéré est de : **5 % x 60 % x 15 x 10^{3} = 450 µs**
Dans les autres cas, cache libre disponible ou page non modifiée, qui représentent 40 % de 5 % des cas, le temps d'accès pondéré est de : 5 % x 40 % x 7 x 10^{3} = 140 µs
Le temps d'accès effectif est donc de : **0,19 + 450 + 140 = 590,19 µs**

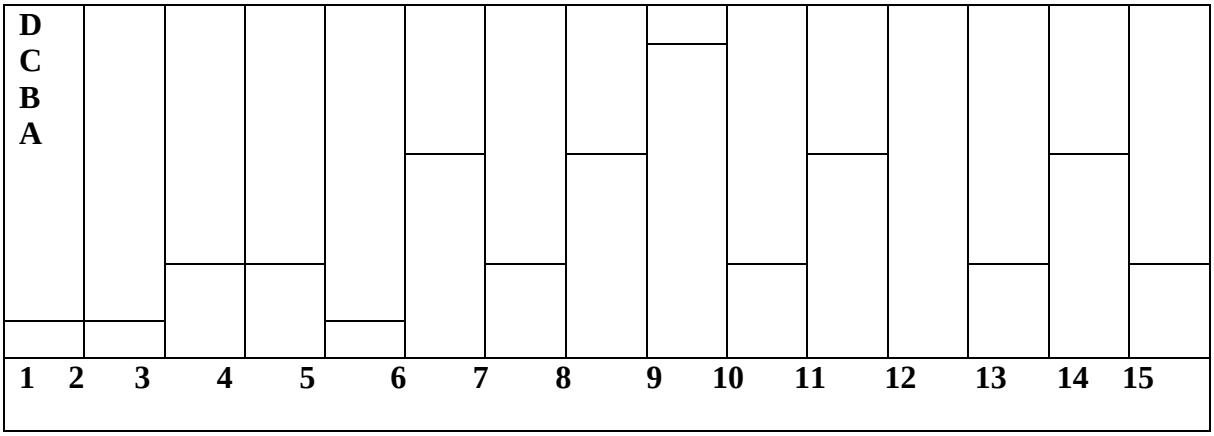


6. Un processus a accès à f cadres de page, tous vides initialement. Si le processus réalise m accès mémoire à p pages distinctes, quel est le nombre minimum de défauts de pages qui va survenir ? Quel est le nombre maximum?

Principe des systèmes

1- commutation de contexte : basculement ou changement de l'environnement du processeur d'une tâche à une autre. **Peut-on en parler dans un système monoprogrammé ?**Oui puisque la notion de tâche n'est pas liée à la multiprogrammation du port et puisque SE coexiste avec le processus en cours d'exécution
2- à quoi sert la mémoire virtuelle dans un système de gestion de mémoire ? Elle permet de faire fonctionner des applications nécessitant plus de mémoire qu'il n'y a de mémoire vive disponible sur le système. En contrepartie cette mémoire est beaucoup plus lente**. En parle-t-on dans un système mono programmé ?**Oui puisque la mémoire physique n'est pas toujours suffisante même si nous avons un programme en mémoire
3. un système fait appel à un algorithme d'ordonnancement avec priorité préemptif

Processus	Date d'arrivée	Un ordonnancement a lieu chaque unité de temps. Deux processus de même priorité sont ordonnancés suivant l'ordre d'arrivée dans le système. Illustrer le schéma d'exécution des processus de la table 1 si beta =-1 et alpha= 17 Temps de traitement
A	0.000	3
B	1.001	6
C	4.001	4
D	6.001	2



Questions

1. donner deux fonctions du système d'exploitation relatives à la gestion des disques : Gestion de la mémoire principale ; Gestion des fichiers

2. section critique : La partie du code d'un processus qui fait référence à une ressource critique ou partie d'un programme où se produit le conflit d'accès pour éviter

3. différence entre un programme et un processus. Un processus est l'activité résultant de l'exécution d'un programme c'est-à-dire tous ce qui est fait quand un programme s'exécute ; donc un programme existe sans processus alors qu'un processus ne serait exister sans programme.

4. commande linux

toppermet de suivre les ressources que le système utilise--**, rm**supprimer un fichier ou un repertoir**,**

ps, permet de connaitre les processus actifs à un moment donné--**fg,-- mount,** permet d'utiliser CdROM et lecteur de disquettes--**cp,** copier un fichier ou répertoire--**cd,** se déplacer dans les répertoires--**chmod,** placer les droits d'utilisation d'un fichier--

mkdir, créer un répertoire--**exit**déconnexion de l'utilisateur courant—

5. Un tube est: Ce sont des canaux permettant la communication entre le père et le fils. Les deux processus père et fils possèdent alors chacun un descripteur en lecture et en écriture sur le tube.

Exclusion mutuelle

Dans certains systèmes…

1. énumérer les causes de pertes de contrôle du processeur interne à un processus en cours : --deux processus sont simultanément en section critique –une hypothèse est faitesur la vitesse ou le nombre de processus --un processus en dehors d'une section critique bloque les autres (l'interblocage) -- un processus attend indéfiniment d'entrer en section critique (la famine)

2. montrer qu'en permettant aux processus utilisateur de masquer les interruptions dans un tel environnement, on peut réaliser l'exclusion mutuelle cette méthode consistera donc à endormir un processus (en masquant ses requêtes d'interruption) qui tentera d'accéder à un verrou fermé en envoyant une requête d'interruptionet de le réveiller plus tard quand la ressource sera disponible. Ainsi il n'entrera pas en section critique et l'on aura réalisé l'exclusion mutuelle.

3. inconvénient d'un tel système : lenteur instabilité du système

2. Les processus

1. Qu'appelle-t-on descripteur de processus,

Chaque processus peut être représenté par un descripteur de processus (aussi appelé *Bloc de contrôle* ou *Vecteur d'état*) qui est une zone de mémoire contenant les informations essentielles concernant le processus.

Le SE tient à jour la table de processus ayant une entrée pour que chaque processus présent. Cette entrée correspond pour chaque processus au descripteur de processus.

Et quelles informations contient-il ? Zone de mémoire contenant les informations essentielles concernant le processus. En premier, nous y trouvons le nom du processus et une indication sur l'état (statu) du processus. Ensuite, la sauvegarde de toutes les informations utiles en cas d'abandon par le processeur. Le PID, le code à exécuter, les priorités, le temps d'exécution, les fichiers ouverts, les emplacements mémoires, les objets IPC

2. A quoi la préemption peut-elle servir dans un système d'exploitation ? Le SE alloue une ressource à un processus. Une fois une ressource allouée, le processus a le droit de l'utiliser jusqu'à ce qu'il libère la ressource ou jusqu'à ce que le SE reprenne la ressource (on parle en ce cas de ressource préemptible, de préemption).

3. Pour chacune des transitions suivantes entre les états des processus, indiquer si la transition est possible. Si c'est le cas, donner un exemple d'évènement qui pourrait en être à l'origine.

- **En exécution - prêt : oui -** processus préempté
- **En exécution - bloqué :** oui - attente d'entrée-sortie
- **En exécution - swappé,** bloqué : non, mais en exécution-bloqué-swappé possible : swap de code
- **Bloqué- en exécution :** non, mais bloqué-prêt-en exécution possible : ressource obtenue

- **En exécution - terminé :** oui - fin de processus

4. Le pourcentage d'attente des E/S, ω, d'un processus est le pourcentage de temps pendant lequel ce processus attend l'achèvement des E/S, lors de l'exécution dans un environnement de monoprogrammation. Sur un système qui a recours à l'ordonnancement à tourniquet avec n processus, tous ayant le même pourcentage d'attente des E/S, quel est pourcentage de temps d'inactivité de l'UC en terme de ω ?
Si le pourcentage d'attente des E/S est ω, la probabilité, à tout moment, du'un processus soit en attente est également ω ; ainsi, si un processus passe 65% de son temps à attendre E/S, la probabilité qu'il à tout moment est 65%.

5. Sur un système recourant à l'ordonnancement à tourniquet, quelle serait la conséquence de l'introduction d'un même processus à deux reprises dans la liste des processus ? Chaque processus prêt dispose d'un quantum de temps pendant lequel il s'exécute. Lorsqu' il a épuisé ce temps ou qu'il se bloque, par exemple sur une entrée-sortie, le processus suivant de la file d'attente est élu et le remplace. Le processus suspendu est mis en queue du tourniquet. Il aura deux fois plus de temps d'accès dans l'UC que les autres.

6. Un système fait appel à un algorithme d'ordonnancement avec priorités préemptif, où les processus au numéro de priorité élevé ont une priorité plus importante. Les processus sont introduits dans le système avec une priorité 0. Lors de l'attente dans la le des processus prêts, la priorité d'un processus change au rythme α. Lors de l'exécution, sa priorité change au rythme β.
(a) Quel algorithme résulte de β > α > 0 ? FIFO – Lorsqu'on arrive à la priorité 0 –si to= 0le processus s'éteint pas il est éxécuté – si t1 > to le processus P1 sera exécuté après le processus Po puisqu'il aura une priorité alpha plus élevée parcequ'il aura plus attendu beta>alpha>0 – ainsi de suite
(b) Quel algorithme résulte de β < α < 0 ? LIFO

7. Sur un système utilisant l'ordonnancement à tourniquet, s représente le temps nécessaire pour réaliser une commutation de processus, q, le quantum de temps à tourniquet et r, le temps moyen d'exécution d'un processus avant de bloquer sur E/S. Donner une formule pour l'efficacité de l'UC (taux d'activité) en fonction de ce qui suit :
(a) $q = \infty$;(b) $q > r$;(c) $s < q < r$;
(d) $s = q < r$; (e) q sensiblement= 0

8. Pour les algorithmes d'ordonnancement non préemptifs, démontrer que l'ordonnancement du travail le plus court d'abord donne le temps d'attente global le plus faible.
Prenons n processus ; temps d'exécution : $\mathbf{t_1<t_2<t_3<\ldots<t_n}$
Si on compare par t_1 ensuite par t_2 : $\mathbf{T_G = (n-1)t_1 + (n-2)t_2 + \ldots + t_{n-1}}$; supposons que **n=2** on a : $\mathbf{T_G = t_1}$; donc si $\mathbf{t_1<t_2}$ on a $\mathbf{T_G}$ est minimal sinon $\mathbf{t_G}$ est maximal
$\mathbf{_{n+1} = n_{t1} + (n-1)t_2 + \ldots + 2t_{n-1} + t_n = ((n-1)t_1 + (n-2)t_2 + \ldots + t_{n-1}) + t_1 + t_2 + \ldots + t_{n-1} + t_n}$
Soient n processus P1, P2,…Pn dans le système avec le temps d'exécution respectifs $\mathbf{t_1, t_2,\ldots t_n}$.
Nous voulons démontrer que l'ordonnancement du travail le plus court d'abord donne

le temps d'attente global le plus faible
Soit O un ordonnancement non préemptif pour ces processus ; évaluons le temps moyen d'attente pour O.
$O=P_1+P_2+P_3+\ldots+P_n$; le temps d'attente **t_{Ai}** pour le processus i sera : **$t_A= t_1+t_2+\ldots+t_{i-1}$** et le temps moyen sera
$= 0+ (t_1) + (t_2 +t_1) +\ldots+ (t_1 + t_2 +\ldots+ t_{n-1}) = ((n-1)t_1 + (n-2)t_2 +\ldots+2t_{n-2} + t_{n-1})/n$
L'ordonnancement au plus court d'abord est celui de **$t_1<t_2<t_3\ldots<t_n$** il faut donc montrer que TM est minimal ssi **$t_1t_2t_3\ldots t_n$**.
Montrons par récurence sur n : n=0, rien à faire ; n=2 : TM1=T1 il est minimal
Supposons que on a : TMn minimal lorsque t1<t2<…<t_n et montrons que T_{Mn+1} est minimal ssi t1<t2<…<tn<tn+1
TMn+1= nt1 + (n-1)t2 + … + 2tn-1) + (t1 + t2 +…+ t_n) donc T_{Mn+1}= T_{Mn} + or TMn est minimal ssi t1<t2<…<tn-1
Il reste à montrer que : si tn>ti i<n et si alors tj>tn
Supposons qu'il existe j / tj>ti**…….**

Synchronisation

1. Dans certains systèmes, une fois qu'un processus obtient le contrôle de l'UC il ne peut le perdre que lorsque l'exécution est transférée au système d'exploitation. Un processus peut empêcher le système d'exploitation de s'exécuter en n'émettant pas d'instruction d'appel au superviseur, en n'exécutant aucune instruction susceptible de provoquer un déroutement.

(a) Montrer qu'en permettant aux processus utilisateur de masquer les interruptions dans un tel environnement on peut réaliser l'exclusion mutuelle. - Lorsque les interruptions ont été masquées l'ordonnancement ne peut plus s'exécuter donc le processus possèdera le processeur exclusivement-ne faire que des interruptions ne provoque pas déroulement –ne pas faire d'appel au superviseur
(b) Enumérer les inconvénients d'un tel système pour la multiprogrammation. Le système peut être bloqué si le processus oublie de masquer les interruptions
(c) Examiner la faisabilité pour un système multiprocesseur. Puisque dans un système multiprocesseur on peut avoir plusieurs processeurs pour cela un processus ne peut masquer les interruptions de son processus, mais en ce temps les processeurs continuent d'évoluer

2. Considérer le morceau de code suivant, définissant un sémaphore. Chacune des fonctions p et v est exécutée en mode superviseur, lorsqu'un processus veut entrer en section critique.

```
struct semaphore{
int count;
process_queue queue; //file d'attente
};
void p(semaphore s){
if(s.count > 0)
s.count = s.count - 1;
else
s.queue.insert() // bloque ce processus
// et l'insère dans la file
}
```

```
void v(semaphore s){
if(s.queue.empty()) // file d'attente vide
s.count = s.count + 1;
else
s.queue.remove(); // libère un processus
// bloqué sur s, s'il y en a.
}
```

- ***Montrer que si le sémaphore est bien initialisé (count), cette structure réalise l'exclusion mutuelle.***

(b) Dans quelles conditions peut-on éviter la famine avec ce sémaphore ?

(c) Discuter les cas où le sémaphore est initialisé à 1 et à n > 1.

Initialisation à 1, assurance mutuelle d'un accès exclusif si P est exécuté avant d'entrer dans une section critique et si V l'est à la sortie

Initialisant à n<, P et V peuvent être utilisés pour autoriser n processus dans leur section critique.

3. Démonter l'algorithme suivant (Algorithme de Peterson) répond aux besoins d'un mécanisme de synchronisation d'accès à une section critique.

```
shared boolean wantIn[2] = false;
shared int turn = 1;
int myPid = 0; // Pour le processus 0.
// Initialisation à 1 pour le processus 1
int otherPid = 1 - myPid;
wantIn[myPid] = true;
turn = otherPid;
while(wantIn[otherPid] && (turn == otherPid))
DoNothing();
// Section critique
wantIn[myPid] = false;
```

4. Démonter l'algorithme suivant ne répond pas aux exigences d'un mécanisme de contrôle d'accès à une section critique.

```
shared int turn = 1;
int myPid = 0; // Pour le processus 0.
// Initialisation à 1 pour le processus 1
int otherPid = 1 - myPid;
wantIn[myPid] = true;
turn = otherPid;
while(turn != myPid)
DoNothing();
// Section critique
trun = otherPid;
```

car la section critique est bloquée indéfiniment en attente du changement de la valeur du sémaphore. Le thread ne se réveille pas lors de l'envoi de la notification

5. Problème du coiffeur endormi : Soit un salon de coiffure comportant n fauteuils pour les clients qui attendent, un fauteuil pour la personne coiffée et un coiffeur. Si un client entre dans le salon et qu'il n'y a aucun fauteuil libre, il quitte le salon. Si le client entre dans le salon et que le coiffeur dort, il le réveille et se fait coiffer. Sinon il s'assoit et attend. Si le coiffeur en a terminé avec un client et qu'il y en a qui attendent, il s'occupe du suivant. Sinon il s'endort sur son fauteuil. A l'aide de sémaphores, écrire les fonctions pour contrôler les actions du coiffeur et du client.

Scenario pour le coiffeur

Le processus coiffeur : --- sert le premier client de la file, s'il y en a un (sinon il se bloque), --- décrémente Attend (accès exclusif), --- libère le siège du client (l'invite à s'asseoir) --- Coiffe

Scenario pour un client

Ce que fait un client : --- Si le nombre de clients en attente est supérieur à N, sort --- incrémente Attend (accès exclusif) - s'ajoute à la file d'attente --- attend de pouvoir s'asseoir dans le siège du coiffeur --- se fait coiffer et sort.

Modélisation avec sémaphores

Les sémaphores utilisés : --- **SP** : gère la file des clients en attente. Indique le nombre de ressources pour le coiffeur, c'est-à-dire les clients. - **SCF** : gère l'accès au fauteuil du coiffeur. Indique si le coiffeur est prêt à travailler ou non (il est la ressource unique des clients). --- SP et SCF gèrent un rendez-vous, -- **SX**: assure l'accès en exclusion mutuelle à la variable partagée Attend.

Attend permet de contrôler la taille maximale de la file d'attente

Ces sémaphores sont initialisés ainsi :

Sémaphore	**Init(SP, 0)**	**Init(SCF, 0);**	**Init(SX, 1);**
Rôle :	Gère les ressources qu'attend le coiffeur	Gère la ressource unique (le fauteuil) qu'attendent des clients	Gère l'exclusion mutuelle sur la ressource partagée par tous coiffeur et clients

Le coiffeur	**Les clients**

Coiffeur(){
 	
while(1){
P(SP);
P(SX);
Attend = Attend - 1;
V(SCF);
V(SX);
Coiffer();
}
 	
}
	Client(){
	P(SX);
	if(Attend < N){
	Attend = Attend + 1;
	V(SP);
	V(SX);
	P(SCF);
	
	SeFaireCoifferEtSortir
	();
	}
	else{
	V(SX);
	Sortir();
	}
	}

```
Shared semaphore cthair=o ;
Shared semaphore waiting=0 ;
Shared semaphore contmuex=1 ;
Shared int count=0 ;
Void barber()
{
While (true)
{
P (cut hair) ;
Givehaircut () ;
}
   {
     Void customer ()
   }
  P ( count mutex) ;
If (cont == n+1 // N chaises plus la chaise du bard
Count== count+1 ;
If (count>1)
    {
     //-----------------------------------
     Prendre une chaise
   //-----------------------------------
   }
   V(counmutex) ;
P(waiting) ;
  Else
```

écrire dans un cahier.
En utilisant les sémaphores (question 2), écrire une solution au problème des lecteurs et rédacteurs qui attribue la priorité aux lecteurs.

```
Shared semaphore cthair=o ;
Shared semaphore waiting=0 ;
Shared semaphore contmuex=1 ;
Void reader ()
{p (rsem);
readCounter = readCount + 1;
if (readCounter ==1)
p(wSem);
readCounter = readCount - 1;
if (readCounter ==0)
V(wSem);
}
Void writers;

{P(wSem);
Write ( );
V(wSem);
}

Sem write(1) ;
Sem mutex(1) ; // Pour Nbl
Int Nbl = 0 ;

Redacteur (){
        P(write) ;
        Ecrire() ;
        V(write) ;
}

Lecteur(){
P(mutex)
Nbl++ ;
If (Nbl==1)
        P(write) ;
V(mutex) ;
Lire() ;
P(mutex) ;
Nbl-- ;
If (Nbl==0)
        V(write) ;
V(mutex) ;
}
```

Mémoire

Un programme contenant un code translatable a été crée, partant du principe qu'il serait chargé à l'adresse 0. Le programme entier fait 64Ko. Dans un code, le

programme fait référence aux adresses suivantes : 0x15300, 0x2154, 0x2341.

a- le programme tt entier pourra til ^stre chargé si l'on dispose d'une mémoire partitionnée de 1Mo au total, et si l'espace libre commence à 0x15300 ? Pourquoi ? La mémoire étant partitionnée chaque programme chargeable sera enregistré dans une zone contigüe. Si le programme est chargé au début de l'espace libre, il devra.. Jusqu'à l'adresse 0x15300 + 0x10000 = 0x25300. Puisque la plus haute adresse de la mémoire est 0xFFFFF, le programme entier sera contenu en mémoire ; il pourra donc être chargé.

b- en supposant que la première page du programme est numéroté 0, à quelle page correspond chacune des adresses 0x0032, 0x0078, 0x2154, 0x2341, 0x6000, 9x9230x 0xC123 ? Les adresses seront ajoutées à l'adresse de début de la zone. On aura donc les ajustements suivants :

adresse	Adresse ajustées
0x0032	**0x4332**
0x01454	**0x5754**
0x2154	**0x6454**
0x2341	**0x6641**
0xA123	**0xE423**

c- si l'algorithme de remplacement de page est LRU, et qu'à l'origine aucune page du programme n'est en mémoire……

d- combien y-a-t-il alors de défauts de pages en tout ? 6 défauts

Périphériques

1- Un disque possède 19456 cylindres, 16 têtes et 63 secteurs par piste. Le disque tourne à 5400 tours par minute. Le temps de positionnement entre les pistes adjacentes est de 2ms. Les secteurs ont une taille de 2048 octets

a- combien de pistes chaque plateau du disque possède-t-il ? Combien y-a-t-il de plateaux en tout ? Le nbre de pistes est égale au nbre de cylindre**.** On a donc 19456 pistes par plateau et 8 plateaux

b- quelle est en mégaoctets la taille théorique de ce disque ? 19456*16*63*2048 = 19152Mo = 38304 Mo

c- ce disque est formaté avec une taille de bloc de 768 octets. Quelle quantité d'information peut-on réellement y stocker ? On aura donc 2 blocs. Par secteur d'où un taux d'utilisation de (768*2)/2048 soit donc : 19152*[(768*2)/2048] = 14362Mo

d- sur combien de secteurs ce disque sera stocké un fichier vidéo de 600M0 ? (600*1024*1024)/(768*2) = 409600 secteurs

e- combien de temps… dans les cas suivants : le fichier occupera 409600/16=25600 secteurs sur chaque plateau

i : sur le même plateau, les secteurs du fichier. Chaque plateau compte 19456 pistes ; on lira 194456 secteurs pistes par piste puis 6144 pistes. D'où un temps de lectures total de [(25600*60)/63*5400] + (25599*0,002) = 55,71 secteurs.

ii : sur les mêmes plateaux stockés sur des secteurs entrelacés de .. chaque piste auront des blocs du fichier pour les 25600 secteurs : 25600/31 = 826 pistes seront sollicités, la dernière n'ayant que 25 secteurs sollicités. Ainsi T=825* (60/5400) + (25/31) * (60/5400) + (825 * 0.002)= 10.825s

2- une application qui affiche de la vidéo sur un moniteur pixélisé rafraichit l'écran 50 fois par seconde.
a- donner la taille de la mémoire 640*480*2= 614400 octet=600Ko
b- quel pourcentage de temps UC est gaspillé si l'application ne peut accéder directement à la mémoire vidéo ? il faut 614400/8 = 76800 instructions par rafraichissement soit 76800*50=3840000 instructions chaque seconde ; en même temps on aurait $1/50*10^{-9}$ = 20000000 instructions. On déduit que le gaspillage est (3840000*100) / 20000000 = 19.2%

Question :

Expliquer la différence entre les termes suivants :

b- état prêt et état bloqué Prêt (Ready)
Processus prêtà s'exécuter quand le processeur lui en donnera l'occasion.
En exécution(Running) Processus en cours d'exécution
Bloqué (Blocked) Processus qui ne peut être exécuté avant qu'un certain événement se soit produit (par exemple attente d'un périphérique d'entrée/sortie). Le processus est en mémoire centrale.
Prêt/Suspendu(Ready/Suspend) Le processus est mis en mémoire secondaire et sera prêt à être exécuter dès qu'il sera chargé en mémoire principale. Le processus peut être suspendu par le système s'il semble provoquer des problèmes (cas d'un utilitaire ou d'un processus en arrière-plan), par l'utilisateur lors d'un débogage ou pour utiliser une ressource particulière, s'il s'agit d'un processus périodique en attente de sa prochaine date de réveil, par un processus parent en vue de la modification ou de la synchronisation de ses enfants.
Bloqué/Suspendu(Blocked/Suspend) Processus bloqué qui a été supprimé de la mémoire principale et mis sur le disque dans une queue de processus suspendus pour laisser la place à de nouveaux processus ou à d'autres processus bloqués (swap de processus). Le processus est en mémoire secondaire et attend l'événement pour lequel il a été mis en état bloqué (Event wait).
Terminé (Exit) Processus qui a été ôté (Release) de la liste des processus exécutables par le système opératoire (fin normal du processus ou arrêt brutal).

c- multiutilisateur plusieurs utilisateurs en même temps ds le système **et multi programmé** ++ pgme peuvent être xécutés simultanément

La pagination est une technique de gestion de la mémoire selon laquelle l'espace d'adressage d'un processus est divisé en petites unités de taille xe2 appelées pages. La mémoire est également découpée en unités physiques de tailles identiques et de même taille que les pages appelées **cadres.**
Pagination à la demande C'est une variante de la pagination. Ici, les pages d'un programme qui sont stockées sur une mémoire ne sont chargées en mémoire principale que lorsqu'elles sont référencées. Une fois en mémoire, ces pages sont accessibles comme dans la pagination simple.

Segmentation : Dans un système paginé, l'espace d'adressage virtuel d'un processus est à une dimension. Or en général, un processus est composé d'un ensemble d'unités logiques : les données initialisées, les données non initialisées, les piles d'exécution. L'idée de la technique de segmentation est d'accorder pour un même processus plusieurs espaces d'adressages virtuels séparés. On appelle donc segment cette division indépendante de la mémoire d'un processus. Chaque segment est caractérisé par son numéro et sa longueur variable.

Pagination segmentée Lorsque le système possède beaucoup de pages, la consultation de la page peut être lente et inecace s'il y a beaucoup de pages non chargées. On la segmente alors par processus. Chaque processus aura donc une table des segments da la table des pages qui le concernent. Une adresse virtuelle sera un triplet : le numéro du segment, le numéro de la page et le déplacement dans la page.

Segmentation paginée : La segmentation peut être combinée avec la pagination. Chaque segment est composé d'un ensemble de pages. Une adresse virtuelle sera un triplet : le numéro du segment, le numero de la page et le déplacement dans la page.

LRU : Least recently used L'algorithme du moins récemment utilisé ou LRU (least recently used) retire la page la moins récemment utilisée.

Un bon algorithme d'ordonnancement doit respecter les critères ci-après :

Équitté : tous les processus doivent avoir accès au processeur Ecacité : le processeur doit toujours être actif Temps de reponse : les travaux inter-actifs doivent l'avoir au minimum Temps de service : les travaux servis doivent être maximiser par unité de temps

L'ordonnancement est dit pré-emptif si le processus peut être retiré d'un processeur avant la fin du processus. Un ordonnancement est dit **non pré-empti**f si un processus qui accède au processeur s'exécute jusqu'à la terminaison avant qu'un autre ne soit accepté.

conditions doivent être vérifiées pour assurer l'exclusion mutuelle : deux processus ne doivent être simultanément en section critique on ne doit faire aucune hypothèse sur la vitesse ou le nombre de processus aucun processus en dehors d'une section critique ne doit bloquer les autres (**l'interblocage**) aucun processus ne doit attendre indéfiniment d'entrer en section critique (**la famine**)

Quelques solutions au problème d'exclusion mutuelle : **Attente active** L'attente active est une approche qui résoud l'exclusion mutuelle en mettant les processus qui veulent accéder à une section critique en attente si cette dernière n'est pas libre **Blocage et Reveil** Cette technique consiste à endormir un processus qui tente d'acceder à un verrou fermé et de le reveiller plus tard quand la ressource sera disponible.

Exercice 2 il est interdit de jouer

Le département de Math-Info a constaté que les étudiants font très souvent autre chose que le TP qui leur est proposé une fois au labo.

Il décide alors de mettre en place un système de surveillance des groupes pendant son passage en salle. Le répertoire **/home/sti3** contient les fichiers contenant chacun la liste des logins des étudiants d'un groupe. Par exemple **/etc/home/sti3/groupe_1** contient les étudiants du groupe 1.

On vous demande alors d'écrire un script qui prend en paramètre le numéro d'un groupe et le programme dans lequel les membres du groupe doivent travailler ; votre script doit alors tuer les autres processus de chacun des membres du groupe (on ne traitera pas le cas des programmes de connexion eux-mêmes, comme **bash** ou **login**).

Vous disposez des commandes suivantes, en plus de votre propre connaissance du shell Linux :

- **pgrep -u user** affiche les identifiants des processus appartenant à l'utilisateur user.
- **pgrep -u user name** affiche l'identifiant du processus de nom name appartenant à l'utilisateur **user**.
- **kill -9 pid** termine le processus de PID pid.
- **grep motif fichier** affiche les lignes du fichier **fichier** contenant motif.
- **cut -c n- ligne** affiche la _n de la ligne **ligne** à partir du caractère numéro **n**.

1. Comment avoir la liste de tous les membres du groupe 3 ?cat /etc/home/sti/groupe_3

2. **Sachant que la commande ps -aux affiche tous les processus présents sur le système, dire comment** combiner (à l'aide de tubes) ps, grep et cut pour obtenir la ligne de commande pour un processus de pid donné. Un exemple d'affichage de ps -aux est (COMMAND est en colonne 64) :USER PID %CPU %MEM VSZ RSS TTY STAT START TIME COMMANDmamadou 4390 0.0 2.4 29396 12460 ? R 08:13 0:02 gnome-terminal : **La commande pour obtenir la ligne de commande pour un processus de pid donné en combinant ps,grep et cut est** :
ps –aux|grep pid|cut -c 64

3. Au fait comment obtient-on la liste des PID des processus non désirés pour un utilisateur donné sachant son login et le PID du seul processus désiré ?pgrep -u login|grep –v "~pid"

4. Ecrire alors le script.

Dans ce script, $1 représente le numéro du groupe
$2 représente le nom du programme désiré
représente le début d'un commentaire.

```
# !/bin|sh
 For login in 'cat /etc/home/sti3/groupe_$1';
# Login prend ses différentes valeurs dans la liste  des membres du groupe $1
do
       For PID_processus_désiré in ' pgrep –u $login $2' ;
#PID_processus_désiré aura comme valeur le pid
       #du processus de nom $2 appartenant à l'utilisateur login
       do
          For PID_processus_ non_désiré_in ' pgrep –u $login|grep –v "~ $PID_processus_désiré" ' ;
      #PID_processus_non_désiré prend ses différentes valeurs dans
      #la liste des pid des processus non désiré pour l'utilisateur login
          do
              Kill -9 $PID_processus_non_désiré
   # On tue les processus non désirés un par un
        done
      done
done
```