Préparation à l'examen de *Systèmes d'exploitation*

Wéry Benoît

27 novembre 2017

Chapitre 1

Définitions de concepts

1. *Définissez ce qu'est le système d'exploitation d'un système informatique. En particulier, identifiez ses buts et les abstractions du système informatique qu'il permet d'opérer. Présentez également la structure d'un système informatique et où il s'y situe.*

   Le système d'exploitation (Operating System = OS) est une **couche logicielle intermédiaire** entre l'utilisateur et le hardware.

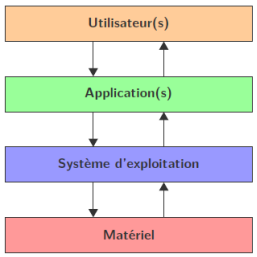


FIGURE 1.1 – Vue en couches du système informatique [?]

Ses principaux objectifs sont :

(a) **Pratique** : permettre une utilisation facile du système informatique (SI) pour l'utilisateur

(b) **Efficace** : optimiser l'utilisation du hardware

(c) **Evolution** : ajouter de nouvelles fonctions systèmes

Parmi les *abstractions* que l'OS propose, on retrouve les suivantes :

Matériel	Abstraction
CPU et mémoire	Processus
Stockage sur disque	Fichier
Interface réseau	Socket, RPC, RMI
Affichage	Librairies de dessin, fenêtres

FIGURE 1.2 – Abstractions proposées par l'OS [?]

L'OS fournit un environnement pour utiliser les ressources (matériel, logiciel et données) du système informatique.

Exemples d'OS : Windows10, macOS, Android, Ubuntu, Debian, Solaris,...

Remarques :

(a) L'OS facilite l'exploitation du hardware MAIS il n'est pas indispensable au fonctionnement du SI.

2. *Quelles sont les principales étapes qui sont franchies lors du démarrage d'un système informatique ? Citez-les et identifiez la fonction de chacune des étapes, c'est-à-dire le rôle des actions réalisées pour le fonctionnement du système informatique.*

(a) Le "programme initial" (boostrap), stocké dans la ROM, initialise plusieurs aspects du système : registres CPU, contrôleur périphérique, contenu de la mémoire,...

(b) Ensuite, il localise et charge dans la RAM le noyau de l'OS (càd le kernel)

(c) Le kernel exécute des processus systèmes (démons), tel que "init", qui eux-même en lancent d'autres,...

(d) Lorsque tous les démons ont été lancés, le démarrage est complété et l'OS se met en attente d'un évènement

3. *Citez et expliquez des exemples de services offerts par un système d'exploitation. Comment peut-on les classer selon le type destinataire du service ? Citez et expliquez également les trois aspects sur lesquels il joue le rôle de coordinateur.*

L'OS peut être vu comme un *fournisseurs de services* pour l'utilisateur et les programmes. Parmi ces services, on retrouve :

(a) L'EXÉCUTION DE PROGRAMMES : l'OS charge en mémoire et exécute un programme

(b) Les OPÉRATIONS D'ENTRÉE/SORTIE : gestion des périphériques, interactions utilisateur

(c) La MANIPULATION DU SYSTÈME FICHIERS : il gère la lecture et l'écriture des fichiers, les permissions d'accès, les propriétaires,...

(d) La COMMUNICATION ENTRE PROCESSUS : l'OS définit quel processus a accès à quelle partie de mémoire,...

(e) La DÉTECTION D'ERREUR : répère les erreurs au niveau du CPU, de la mémoire, des périph, du réseau,... et tente des les corriger

(f) L'ALLOCATION DES RESSOURCES : distribution des ressources (CPU, mémoire,...) entre les différents processus exécutés

(g) La COMPTABILISATION

(h) La PROTECTION et la SÉCURITÉ : authentification utilisateurs,...

Ces services peuvent être regroupés en deux catégories selon le destinataire :

(a) Services pour l'UTILISATEUR

(b) Services pour le SYSTÈME

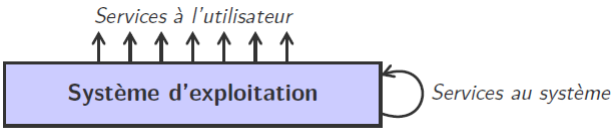


FIGURE 1.3 – Catégories de services de l'OS

L'OS joue également un rôle de **coordinateur** sur trois aspects :

(a) PROTECTION : il s'assure que les jobs n'interfèrent pas les uns avec les autres

(b) COMMUNICATION : il doit permettre aux jobs de pouvoir intéragir entre eux

(c) GESTION DES RESSOURCES : il facilite le partage des ressources (CPU,mémoire,...) entre les jobs

4. *Qu'est-ce-que la multiprogrammation ? Quelles conséquences cette tendance a-t-elle eue sur le développement des systèmes d'exploitation ?*

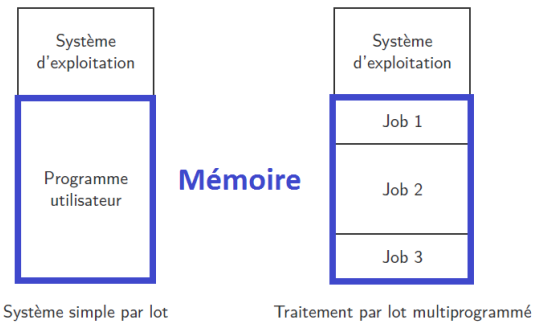


FIGURE 1.4 – Répartition des programmes en multiprogrammation [**?**]

Dans le cas de la multiprogrammation, plusieurs jobs sont mis en mémoire simultanément. Chaque job possède **son propre espace mémoire** qui n'est pas accessible par les autres (Rem : dans certains cas, plusieurs jobs peuvent partager une même zone mémoire, si cela est demandé).

Contrairement au système simple, si un job est en attente d'une réponse, alors l'OS peut choisir d'en lancer un autre. Ainsi, on donne l'*illusion* de faire tourner plusieurs jobs en même temps. On parle de **concurrence**, car le CPU est *multiplexé* entre plusieurs tâches qui sont exécutées les unes après les autres et non pas en même temps (parallélisme).

Cette approche a eu pour conséquence la nécessité de gérer les éléments suivants :

(a) l'**attribution des ressources**

(b) la **gestion de la mémoire** allouée aux tâches

(c) la **planification** de l'utilisation du CPU, càd des tâches à exécuter

A vérifier... Il y a un partage des *ressources* et du *temps-calcul* qui se fait à deux niveaux :

(a) D'une part, *plusieurs batch de jobs* via la multiprogrammation

(b) D'autre part, *plusieurs utilisateurs* avec le time-sharing

5. *Citez et expliquez les deux modes d'exécution possible d'un programme. Qu'est-ce-qu'un appel système et comment se déroule l'exécution d'un tel appel ?*

Un programme peut être utilisé dans deux modes d'exécution :

(a) mode NOYAU : code destiné à l'OS (ces instructions ont plus de privilèges). Dans ce mode, l'OS a un accès complet à l'ensemble du matériel et peut exécuter n'importe qu'elle instruction (que la machine est capable de traiter)

(b) mode UTILISATEUR : code destiné à l'utilisateur (ex : les programmes/softwares). Dans ce mode, les logiciels n'ont accès qu'à une partie des instructions possibles.

Un **appel système** permet d'utiliser un service de l'OS, il s'agit d'une tâche très *bas niveau*. Les appels systèmes disponibles dépendent de l'OS et sont utilisés par des **programmes systèmes**.

Afin de ne pas ré-écrire soi-même les séquences des nombreux appels systèmes d'un service, on utilise une **API** (Application Programming Interface -> ! ! interface au sens de "contrat", classe purement abstraite) via une *librairie*

Lors de l'exécution d'un appel système, le programme passe du mode utilisateur au mode noyau. Le passage d'un mode à l'autre est géré par l'*interface des appels systèmes* qui intercepte ces appels sys. dans l'API.

Ainsi, le programme utilisateur n'a pas connaissance du détails des différents appels sys. Il respecte simplement l'utilisation de l'API (noms fonctions, passage des bons paramètres,...) et comprend la valeur de retour.

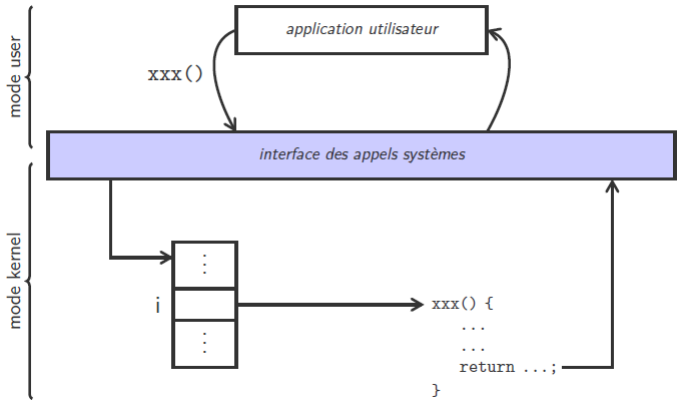


FIGURE 1.5 – Changement de mode via l'*interface des appels systèmes* [**?**]

6. *Définissez le concept de processus et détaillez les différentes opérations que l'on peut réaliser depuis sa création jusque sa terminaison. Citez et expliquez quels sont les différents états possibles d'un processus.*

Un processus est un **programme en cours d'exécution**. Pour rappel, un programme peut se retrouver dans deux états :

(a) Passif : stocké sur le disque dur et inactif
(b) Actif : lancé par l'OS, il est alors placé en mémoire RAM (on parle ici de processus)

Plusieurs processus peuvent être associés à un même programme. Ex : L'OS, qui est un programme, possède un processus nommé le "dispatcher".

Un processus est une **abstraction** qui représente une activité, il est représenté par un **bloc mémoire structuré** dans la RAM. L'OS attribue à chaque processus une zone mémoire, qui lui est réservée, dans laquelle sont stockées les informations suivantes :

(a) zone texte : code du programme
(b) pile : données temporaires
(c) tas : mémoire dynamiquement allouée
(d) zone de données : variables globales

Un processus peut être créé à différents moments :

(a) Initialisation du système
(b) Aooel système
(c) Requête utilisateur
(d) Initialisation d'un travail en traitement par lots

OPERATIONS...

Les différents états... Dans les versions les plus complètes, on distingu jusqu'à 7 états différents que peut prendre un processus.
— NEW : en cours de création
— RUNNING : instructions en cours d'exécution
— WAITING : en attente d'un évènement
— READY : prêt et en attente du CPU
— TERMINATED : exécution terminée
— WAITING/SUSPENDED : swappé sur le disque dur et en attente d'un évènement
— READY/SUSPENDED : swappé sur le disque dur et prêt à être exécuté

Lorsqu'un nouveau processus est créé, il est placé dans la file des processus READY, en attendant d'être choisi par l'OS pour son exécution. Lorsque son temps imparti est terminé, soit le processus a pu être fini, dans quel cas il est passé à l'état TERMINATED, soit il n'est pas fini et est alors replacé dans la file d'attente READY.

Si un processus doit obtenir une valeur, un évènement, ... il est retiré du CPU et placé en état WAITING, dans une nouvelle file, jusqu'à ce qu'il obtienne sa réponse. Après cela, il pourra alors retourné dans l'état READY.

Lorsqu'il n'y a plus assez de place en mémoire RAM, l'OS peut décider de déplacer certains processus vers la mémoire du disque dur. Cette transistion s'appelle le *swapping* et amène le proc. dans un état SUSPENDED. AU moment de son activation, il sera remplacé dans la RAM.

7. *Comment et où le système d'exploitation maintient-il de l'information par rapport aux processus ? Quel genre d'informations garde-t-il et pourquoi ?*

   L'OS maintient en mémoire une table des processus qui contient un **bloc de contrôle** pour chaque processus. Ces blocs stockent des informations spécifiques telles que :
   — l'état du processus, son identification (PID) et un compteur ordinal (retient la position courante du processus à la fin de sa dernière exécution)
   — les valeurs des registres du CPU (pour sauvegarde)
   — la zone allouée dans la RAM (les adresses limites)
   — une liste des fichiers ouverts par le processus
   — ...
   Lorsqu'un processus est terminé, l'OS met à jour le bloc de contôle qui lui est associé. Lorsqu'il doit charger le processus pour exécution, il utilise les données courantes du bloc.

8. *Quelles sont les différences entre parallélisme et concurrence ? Comment les obtient-on dans un système informatique donné ? Le hardware disponible est-il un facteur déterminant dans le choix entre parallélisme et concurrence lors du développement d'un programme ?*

   Parallélisme... plusieurs tâches sont effectuées simultanément sur les coeurs différents

   Concurrence... le CPU *donne l'illusion* de faire tourner plusieurs tâches en leur attribuant les ressources tour-à-tour

   Pour pratiquer du parallélisme, il faut nécessairement avoir un CPU multicoeurs ou plusieurs CPU physiques ! !

   Remarque... On distingue deux niveaux de parallélisme : para. *de données* (répartition sur plusieurs coeurs des données à traîter par une même tache) et para. *de tâches* (exécution de différentes tâches sur plusieurs coeurs).

9. *Citez et caractérisez les deux types de threads en fonction du niveau où le support des threads est fourni. Citez, expliquez et comparez les différents mappings que l'on peut réaliser entre ces deux types de thread.*

   / ! / ! à compléter

   Un thread est "découpage" d'un processus (c'est en quelque sorte "un processus dans un processus"), une unité d'exécution liée au processus et qui chargée d'en exécuter une partie. Plusieurs threads issus d'un même processus partagent certaines ressources (le code, les données et les fichiers ouverts) et possèdent des informations qui leurs sont propres (un identifiant, des registres et un pile). IMAGE...

   On distingue les threads au niveau **utilisateur** et **noyau**.

1) Threads utilisateurs Le noyau gère les processus et ne se préoccupe pas de l'existence des threads. Donc, lorsqu'il alloue le processeur à un processus, le temps est réparti entre les différents threads et cet ordonnancement est la responsabilité de l'application elle-même (utilisation de foncions telles que : slepp(), suspend(), reusme(), stop()... présentent dans des librairies spécifiques).

Ce système est donc totalement indépendant de l'OS sur lequel il tourne. Par contre, il ne peut pas être utilisé en multiprocesseurs et un appel système d'un thread peut bloquer tout le processus. IMAGES DOCUMENT ANNEXE

2) Theads noyau L'OS gère lui-même l'ordonnancement des threads, au même titre que les processus, en leur allouant du temps CPU. Dans ce cas ci, si un thead est bloqué, ce n'est pas tout le processus qui est bloqué puisque d'autres threads (de ce même processus) peuvent être exécutés. Contrairement au threads utilisateurs, les threads d'un processus peuvent être répartis sur différents coeurs (multi-threading) MAIS, le changement de contexte nécessite de passer par le noyau...

10. *Définissez ce qu'est et ce que fait l'ordonnanceur de processus, et le dispatcher. Quand ces derniers sont-il exécutés ? Qu'est-ce-que la préemption et quels sont ses avantages et inconvénients ?*

L'**ordonnanceur court terme** est un processus OS qui **choisit** le processus de la *ready queue* qui sera envoyé sur le CPU. Il alterne donc les processus qui sont maintenus en mémoire pour qu'il ait acces aux ressources chacun à leur tour.

On distingue 4 instants où l'ordonnanceur doit prendre une décision :

(a) Lorsqu'un p. passe de l'état Running à Waiting
(b) Lorsqu'un p. passe de l'état Running à Ready
(c) Lorsqu'un p. passe de l'état Waiting à Ready
(d) Lorsqu'un p. se termine

Le **dispatcher** est un processus OS qui **donne** le contrôle du CPU à un processus. Il doit...

(a) Initialiser tout ce qu'il faut pour lancer le p. (celui choisi par l'ordonnanceur). Pour cela, il regarde dans la *table des processus* le *bloc de contrôle* de ce processus.
(b) Il bascule en mode utilisateur (changement de 1 bit)
(c) Il se positionne à la bonne adresse mémoire pour relancer le p.

Chaque dispatch engendre un **temps de latence** (temps "perdu" pour l"utilisateur), il est donc important que ce dispatcher soit le plus rapide possible.

Rappel préemption... un système/OS est dit préemptif s'il peut interompre à tout instant une tâche en cours d'exécution. On distingue ainsi les ordonnanceurs préemptifs (un processus garde le contrôle CPU jusqu'à ce qu'il le relâche) des ordonnanceurs non-préemptifs (un p. peut être retiré à tout moment du CPU -> ex : interruption, niveaux de priorités,...)

Dans le dernier cas, si un processus peut être interrompu avant son arrêt "normal" (tâche terminée, changement état volontaire), il faut mettre un place des mécanismes, par exemple : dans le cas de données partagées, il faut montrer que la donnée utilisée n'est plus à jour,...

/ ! / ! / ! Dans un cas préemptif, dès qu'un processus arrive, l'OS doit reprendre la main pour calculer la priorité de ce p. et prendre une décision (algo. ordonnancement). Ces "pauses" diminuent le temps en mode "utilisateur", il faut donc que le travail soit fait le plus vite possible.

Plusieurs critères existent afin d'évaluer la qualité d'un ordonnanceur :

(a) l'UTILISATION CPU : le pourcentage de temps où le CPU est occupé
(b) le DÉBIT de processus : nombre de processus terminés par unité de temps
(c) le TEMPS DE ROTATION : temps total écoulé pour l'exécution d'un processus
(d) le TEMPS D'ATTENTE : somme des temps d'attente en ready queue
(e) TEMPS DE RÉPONSE : temps entre la soumission du p. et sa première réponse

En fonction, du contexte on cherchera à optimiser certains critères soit par rapport à une valeur moyenne soit par rapport aux valeurs extremes (max et min).

Exemples d'algorithmes d'ordonnancement... *First-Come First-Served* (FCFS), *Shortest-Job-First* (SJF), *Shortest-Reaming-Time-First*, *Round-Robin* (RR)... Il est possible de mettre en place un système de "files de processus" (en fonciton du type de p.) qui ont chacunes leur propre algo.

11. *Définissez ce qu'est une section critique et dans quelle situation l'existence d'une telle section peut poser problème (illustrez avec un exemple). Citez et expliquez une solution que l'on peut apporter pour protéger de telles sections en synchronisant des processsus.*

Une section critique est une ressource (donnée en mémoire, fichier,...) partagée entre plusieurs processus

Une telle section pose problème en situation de concurrence pour des **opérations atomiques**. Une op. atomique est une opération qui NE PEUT PAS ETRE INTERROMPUE

Exemple... Soit deux processus, qui utilise un buffer. Le premier (producteur) ajoute un élément au buffer tant qu'il n'est pas plein et le second (consommateur) retire un élément s'il y en a un. Les deux programmes partagent une donnée *counter* qui définit le nombre d'éléments présent actuellement dans le buffer.

Pour mettre cette donné à jour, les programmes utilisent des OPÉRATIONS DE HAUT NIVEAU du type "counter++" et "counter−" qui elles-mêmes regroupent plusieurs INSTRUCTIONS BAS NIVEAU. De telles opérations sont atomiques. En effet, si l'un des processus est interrompu par l'autre alors qu'il se trouvait entre deux instructions (autrement dit, l'opération counter++/− nétait pas finie), le second va manipuler une valeur de counter qui n'est pas à jour. En fonction de l'ordre des instructions bas niveaux entre les deux processus, on pourrait donc se retrouver avec une valeur erronnée pour la donnée partagée.

Afin de résoudre un problème de section critique, trois conditions doivent être respectées (simultanément) :

(a) **Exclusion mutuelle** : Si un processus est dans sa section critique, aucun autre processus ne peut y accéder en même temps. Dès lors, un processus doit demander l'autorisation à l'Os avant de pouvoir accéder à une section critique. IMAGE STRUCTURE PROCESSUS

(b) **Progression** : lorsque plusieurs processus demandent l'accès à une section critique (qui n'est pas occupée), l'OS doit choisir qui va pouvoir y entrer.

(c) **Attente limitée** : si un processus demande l'accès à une section critique, il devra tôt ou tard être accepté. Une demande ne peut pas être reportée à vie par l'OS.

Ainsi, dans l'exemple si dessus, si le producteur commence son opération "counter++", càd qu'il entre dans section critique puisque la valeur "counter" est partagée, le second processus ne pourra accéder à la valeur de counter tant que les instructions de bas niveau de l'opération ne seront pas toutes finies.

12. *Définissez ce qu'est un deadlock et dans quelles situations il peut se produire (illustrez avec un exemple). Comment peut-on détecter, prévenir et se remettre d'un deadlock ?*

Un deadlock est une situation dans laquelle un ensemble de processus attend des ressources (partagées) qui sont respectivement utilisées par les uns et les autres. Chaque p. est alors en mode WAITING, formant ainsi un blocage.

Pour qu'une telle situation ait lieu, il faut que les quatres conditions suivantes soient satisfaites **simultanément** :

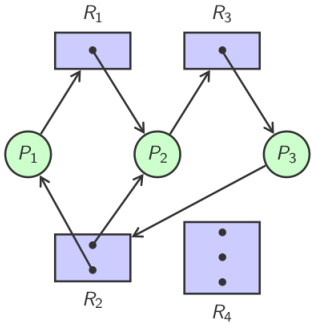
(a) Exclusion mutuelle : au moins une ressource ne peut pas être partagée

(b) Hold and wait : un p. détient des ressources et en attend d'autres qui ne sont pas disponibles

(c) Pas de préemption : si le système n'est pas préemptif, une ressource ne peut pas être volontairement récupérée par l'OS

(d) Attente circulaire : ensemble de pro. qui s'attendent les uns les autres pour récupérer des ressources

Un *graphe d'allocation des ressources système* permet d'illustrer facilement la consommation de ressources à un istant t.



Dans l'exemple ci-dessus, le pro P2 détient une instance de la ressource R2 et il souhaite accéder la re R3. Cette re est utilisée par P3 qui, lui, attend qu'une instance de R2 soit libérée.

Une analyse d'un tel graphe permet de tirer certaines conclusions...

- Si les re n'ont qu'une instance alors un cycle = deadlock
- Si certaines re ont plusieurs instances alors on ne peut rien affirmer. Dans l'exemple précédent, les deux instances de R2 sont consommées. Comme P1 attend la re R1 détenue par R2 (qui est actuellement bloqué), aucune instance ne devrait pouvoir se libérer = deadlock.

Dans une autre configuration, si une des instances avait pu être libérée alors le deadlock aurait été levé tout seul après un certains temps.

Gestion de deadlock... l'OS a trois possiblités pour gérer un deadlock :

(a) PRÉVENTION : l'OS s'arrange pour que le système n'atteigne JAMAIS un état de deadlock (technique particulièrement difficile à mettre en place dans le cas des syst. "grands publics")(Ex d'algo d'évitement : "banquier", "demande de ressources", "état sain")

(b) DÉTECTION : l'OS détecte une situation de deadlock et tente de la résoudre

(c) IGNORANCE : l'OS suppose qu'un deadlock n'arrivera jamais (ex : dans le cas de Linux et Windows, c'est aux développeurs de mettre les solutions en place dans leur applications)

(Qqes algos, description état safe/unsafe, ...)

13. *Citez et expliquez les différents moyens que l'on peut mettre en oeuvre pour faire communiquer entre eux des processus coopératifs ? Quels sont les avantages et inconvénients de ces deux moyens de communication ?*

14. *Expliquez ce qu'est le principe d'adressage ? Comment l'unité mémoire intervient-elle ? Définissez et expliquez le principe de liaison des adresses.*

Adressage : façon dont sont accédées les données stockées en mémoire. La mémoire est un grand tableau d'octets qui sont identifiés par une adresse unique.

Lors de l'exécution d'une instruction par le CPU, des données sont **chargées depuis** la mémoire ou **stockées vers** la mémoire. Pour accéder à la bonne zone de la mémoire, il faut alors préciser son *adresse mémoire*.

L'unité de gestion mémoire (MMU)

On distingue deux types d'espaces d'adresses :

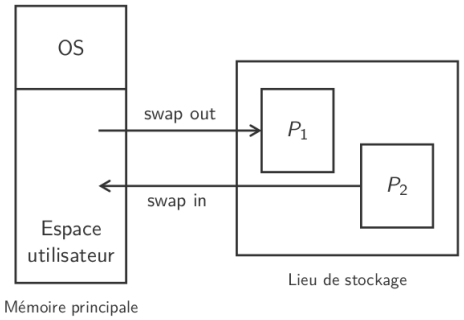
(a) Adresse PHYSIQUE : adresse de la donnée dans la mémoire physique (RAM ou disque)

(b) Adresse LOGIQUE (ou VIRUTELLE) : adresse de la donnée au sein de la mémoire virtuelle propre au processus.

L'unité mémoire, elle voit défiler un flux d'adresses et elle est chargée de faire la conversion entre les deux types ci-dessus. ***Rem :*** elle ne fait que répondre aux demandes, sans interpréter les adresses fournies. L'aspect sécurité est implémenté au niveau hardware et n'est donc pas de sa responsabilité.

15. *En quoi consiste le swapping ? Citez et expliquez les différentes formes de swapping qu'il existe, et comparez-les.*

Le *swapping* consiste à déplacer, temporairement, hors de la mémoire RAM un processus afin d'y libérer de l'espace. Le processus de l'OS qui est en charge des swaps est le **dispatcher**.



On distingue principalement deux formes de swapping :

(a) Le swap d'un PROCESSUS ENTIER : l'entièreté du processus est déplacé vers l'espace de stockage de masse (disque dur)

(b) Le swap d'une PARTIE DE PROCESSUS : certaines parties du processus, dites "pages", sont déplacées. De cette façon, le p. peut encore fonctionner avec les pages restantes en RAM. On peut parle de *pagging*.

Il est a noter que ce déplacement de données nécessite un certains temps qui dépend de la vitesse de transfert (temps perdu point de vue utilisateur). Dans un ordinateur bien configuré et avec suffisamment de RAM, cette opération de swapping ne devrait jamais avoir lieu avec un usage normal.

De plus, si un p. est swappé alors qu'il est attente d'une E/S, lorsque celle-ci sera disponible il devra soit être swappé immédiatement en mémoire RAM soit le résultat de E/S devra être stocké en RAM en attendant le retour du p. (perte de place)

16. *Citez, expliquez et comparez les différentes stratégies d'allocation de la mémoire que l'on peut mettre en place. Quels sont les risques de fragmentation liés à ces stratégies ?*

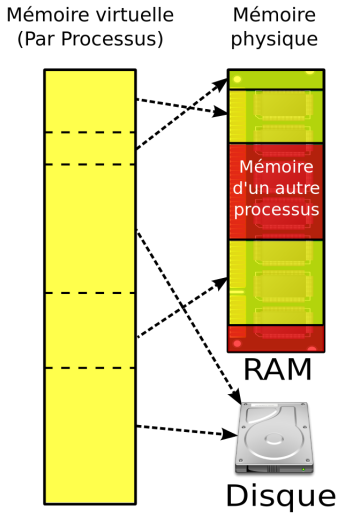
17. *Expliquez le principe de la segmentation. En quoi est-il plus proche de la pensée du programmeur ? Comment les adresses sont-elles construites lorsqu'on utilise la segmentation ?*

18. *Expliquez le principe de la pagination. Quels sont ses avantages par rapport à la segmentation ? Comment les adresses sont-elles construites lorsqu'on utilise la pagination ? Comment peut-on améliorer les performances de la pagination à l'aide d'une mémoire spécialisée ?*

La *pagination* est une technique d'allocation de la mémoire aux processus qui consiste à découper la mémoire logique (= mémoire virtuelle propre à chaque processus) en **pages** de même taille et à découper la mémoire physique (= RAM) en **cadres** de taille fixe. Un cadre est alors rempli par une page et une table des pages (propre à chaque p) permet d'associer les pages aux cadres.

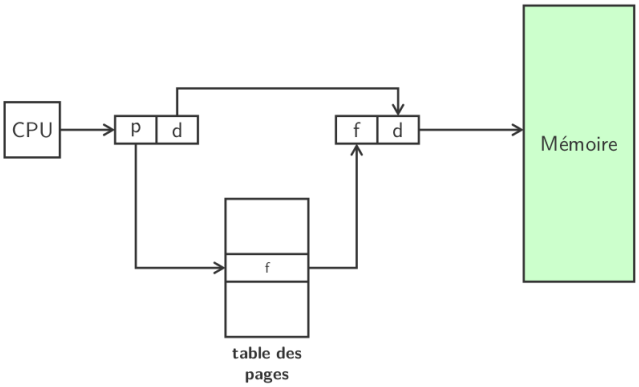
En pratique, l'entièreté d'un processus n'est jamais requise au même moment, stocker l'ensemble de ses données en RAM représente donc une perte de place conséquente. En utilisant la technique de pagination, on divise la mémoire du processus et on ne charge en RAM **que ce**

qui est nécessaire à son fonctionnement à un instant t. Les pages restantes sont alors stockées sur la mémoire de masse et chargées dynamiquement à la demande.



Calcul des adresses

Pour identifier une donnée au sein de la mémoire virtuelle, on utilise son **adresse logique** qui est composée d'un couple <*numéro de page*, *décalage*>



Lorsque le CPU émet une adresse virtuelle (logique), la table des pages fait correspondre le numero de la page avec le cadre (adresse physique) dans lequel elle est stockée et le décalage permet de se positionner au sein cette zone de mémoire.

19. *Citez et expliquez les différentes structures que l'on peut utiliser pour stocker la table des pages. Quels sont les avantages et inconvénients de ces structures ? Dans quelles situations sont-elles plus adaptées ?*

20. *Définissez et expliquez les différences et les liens entre mémoire physique et mémoire virtuelle. Comment le système d'exploitation fait-il le lien entre ces deux mémoires, pour les différents*

processus ?

21. *Définissez et expliquez le concept de défaut de page. Quand un défaut de page se produit-il ? En quoi altèrent-ils les performances du système ?*

22. *Définissez, expliquez et comparez les différentes stratégies que l'on peut mettre en place pour allouer les cadres.*

23. *Définissez la notion d'écroulement et expliquez en quoi elle rend un système instable. Comment peut-on détecter et se protéger d'un écroulement ?*

24. *Définissez la notion de fichier et expliquez comment le système d'exploitation les gère. Citez et expliquez des exemples d'opérations qu'il est possible de réaliser sur des fichiers.*

25. *Expliquez comment un disque peut être structuré en partition/volume/systèmes de fichiers. Citez, expliquez et comparez différentes manières de concevoir une structure en répertoires.*

26. *Définissez ce qu'est un système de fichiers et les informations que le système d'exploitation retient pour l'organiser. En quoi consiste le montage d'un système de fichiers et qu'est-ce-que le système de fichiers virtuel ?*

27. *Citez, expliquez et comparez les différentes méthodes d'allocation utilisées pour stocker les fichiers sur le disque.*

28. *Citez et expliquez les deux types de formatage d'un disque qu'il est possible de réaliser. Quels sont les buts précis de chacun de ces deux formatages ?*

29. *Citez, expliquez et comparez les différentes structures RAID qu'il est possible de mettre en oeuvre. Dans quel cas concret utiliseriez-vous l'une ou l'autre structure ?*

30. *Définissez ce qu'est un driver de périphérique. Où ce dernier intervient-il et avec quels autres composants interagit-il afin de permettre à un utilisateur d'exploiter un périphérique.*

31. *Citez et expliquez la notion de machine virtuelle et quelles sont les deux types de machines virtuelles qu'il est possible de mettre en oeuvre.*

32. *Quelles sont les différences entre moniteur de machines virtuelles et système d'exploitation ? En particulier, quelles sont les opérations qu'ils doivent tous les deux faire et celles qui les distinguent.*

33. *Quelles sont les similitudes et différences entre les deux types d'hyperviseur. Quels sont les avantages et inconvénients de ces deux types d'hyperviseur ?*

34. *Quelles sont les différences entre le kernel Linux, un système Linux et une distribution Linux. Mettez votre comparaison en parallèle avec les trois composants principaux de Linux qui sont en accord avec Unix.*

35. *Définissez et expliquez ce que sont les modules kernel. En particulier, expliquez à quoi ils servent, en quoi ils permettent d'obtenir un kernel minimal et en quoi ils facilitent le développement de drivers.*

36. *La création d'un nouveau processus en Linux passe par deux appels systèmes. Citez-les et expliquez leur rôle. Quels avantages apporte un tel choix de design en comparaison à Windows où un nouveau processus est directement créé avec un unique appel à l'appel système CreateProcess.*

37. *Expliquez le modèle CIA+ en détaillant chacun des objectifs visés.*

38. *Donnez et expliquez trois principes de conception pour le développement de mécanismes de protection.*

39. *Expliquez comment construire un certificat de clé publique, et comment on peut vérifier l'authenticité de la clé.*

40. *Expliquez ce qu'apporte l'utilisation du « sel »dans le stockage de mots de passe.*

41. *Expliquez le contrôle d'accès aux fichiers Unix classique, ainsi que son extension en « Access List ».*

Chapitre 2

Raisonnement

1. *Citez, expliquez et comparez différentes approches possibles pour structurer le code d'un système d'exploitation. Quels sont les avantages et inconvénients de ces différentes approches en fonction du système informatique à opérer ? Argumentez.*

2. *Définissez ce que sont les threads et comparez-les par rapport aux processus. Que permettent-ils de plus, par rapport à un système d'exploitation ne proposant que des processus ? Confondre les threads et les processus en un seul type d'entité comme le fait Linux est-il intéressant ? Argumentez.*

3. *Citez, expliquez et comparez différentes stratégies d'ordonnancement que l'on peut mettre en oeuvre (FCFS, SJF, priorité, RR, files multi-niveaux). Comment peut-on les comparer ? Y a-t-il un meilleur algorithme dans l'absolu ? Argumentez.*

4. *L'acquisition de ressources par des processus peut conduire le système dans des états de deadlock. Il existe plusieurs stratégies permettant d'éviter ou de se remettre d'un deadlock. Quelles sont-elles et en quoi elles sont plus ou moins adaptées selon le type de système informatique ? Argumentez.*

5. *L'utilisation de threads en lieu et place de processus facilite-t-il la communication entre ces entités ? Est-on limités aux mêmes mécanismes de communication ? Où y gagne-t-on et que perd-on ? Argumentez.*

6. *Citez, expliquez et comparez différentes stratégies de remplacement des pages que l'on peut mettre en oeuvre (FIFO, optimal, LRU, approximation de LRU, LFU, MFU). Comment peut-on les comparer ? Y a-t-il un meilleur algorithme dans l'absolu ? Argumentez.*

7. *Un système d'exploitation peut adopter plusieurs attitudes différentes suite à l'apparition d'un*

deadlock. Citez-les et expliquez en quoi elles sont plus adaptées en fonction du système d'exploitation à opérer. Argumentez.

8. *Citez, expliquez et comparez différentes stratégies d'ordonnancement de disque l'on peut mettre en oeuvre (FCFS, SSTF, SCAN, C-SCAN, LOOK, C-LOOK). Comment peut-on les comparer ? Y a-t-il un meilleur algorithme dans l'absolu ? Argumentez.*

9. *Une virtualisation est réussie si le système d'exploitation est incapable de savoir s'il tourne sur une machine physique ou sur une virtuelle. Expliquez comment un système d'exploitation pourrait se rendre compte qu'il est trompé et ce qu'on peut mettre en oeuvre pour le tromper.*

10. *Dans plusieurs situations (ordonnancement de processus, remplacement de pages, ordonnancement d'opérations disque), plusieurs algorithmes sont possibles. Y a-t-il à chaque fois un algorithme qui est le meilleur dans tous les cas ? Comment peut-on comparer ces algorithmes ? Sont-ils bon ou mauvais selon le système informatique à opérer ? Comment choisir du mieux possible l'algorithme à utiliser étant donné un système d'exploitation ? Argumentez.*

11. *Vous devez installer un serveur web sécurisé, sur base d'un OS Linux. Comment procédez-vous ? Détaillez les grandes lignes de votre stratégie de hardening.*