Systèmes d'exploitation

Olivier Roques

2016-2017

Table des matières

1 Introduction

1.1 Modules de base

La machine de *Von Neumann* modélise la plupart des ordinateurs actuels. Elle est constituée de deux parties :

- La partie automate et calcul s'appelle *processeur* ou *microprocesseur*.
- La partie mémoire s'appelle la *mémoire RAM*.

Ces deux modules communiquent par l'intermédiaire de fils, l'ensemble de ces fils est appelé *bus*. Pour communiquer avec l'extérieur, on ajoute à la partie processeur/mémoire des moyens de communication reliés au bus par des *unités d'échange* (UE).

La mémoire unique contient toutes les informations dont a besoin le processeur, c'est-à-dire les données et les instructions. Toutes ces informations sont codées en binaire, rien ne permet de distinguer en mémoire une donnée d'une instruction. Le processeur contient lui aussi des éléments de mémorisation : les registres. Ces éléments jouent un rôle capital, citons :

- le compteur ordinal qui mémorise quelle instruction il va falloir effectuer ;
- le registre de pile qui indique le début de la zone mémoire réservée à la pile.

1.2 Rôle d'un système d'exploitation

Un système d'exploitation a plusieurs rôles :

1. Assurer l'indépendance vis-à-vis des aspects matériels, c'est-à-dire construire une machine virtuelle sur la machine physique.
2. Organiser et optimiser l'utilisation des ressources (matérielles et logicielles).
3. Proposer une interface utilisateur souple et puissante.

Le système est lui-même décomposé en plusieurs couches, en particulier pour des raisons de sécurité. UNIX est organisé en deux couches. La couche la plus proche du matériel s'appelle le *noyau* (kernel). Le noyau, cœur du système, résident en mémoire, offre les mécanismes de base : gestion des interruptions, gestion des tâches. Les fonctions de plus haut niveau, gestion de la mémoire, des fichiers se trouvent dans les couches supérieures du système.

1.3 Caractéristiques

Dans un *système temps partagé*, c'est le système d'exploitation qui impose sa base de temps aux utilisateurs. L'interactivité avec la machine est conduite par celle-ci : on utilise les ressources (disque, clavier, écran) quand le système d'exploitation le décide. L'objectif du système est le dialogue avec tous les utilisateurs, si ceux-ci sont nombreux l'allocation des ressources doit se faire de façon équitable.

Dans un *système temps réel*, c'est l'environnement qui impose sa base de temps au système : il doit donner le contrôle à la tâche qui est vitale à un instant donné. Sa contrainte principale sera de délivrer les résultats attendus avant une date limite appelée échéance, le non-respect de l'échéance pouvant mettre en danger l'installation contrôlée par la machine.

2 Gestion des processus

2.1 Définitions

Définition 2.1. Un *programme* est une suite figée d'instructions, un ensemble statique.

Définition 2.2. Un *processus* est l'instance d'exécution d'un programme dans un certain contexte pour un ensemble particulier de données. Plusieurs processus peuvent ainsi exécuter parallèlement le même programme.

Définition 2.3. Par *contexte d'un processus*, on entend toutes les informations relatives à l'exécution d'un processus, telles que :

- Les registres : le compteur ordinal, le pointeur de pile, le registre de drapeaux.
- Les ressources utilisées et les quotas associés : fichiers ouverts, espace mémoire alloué.

Définition 2.4. Une *ressource* est une quantité limitée d'un matériel ou d'un service qui doit être partagée. Elle peut prendre différentes formes : mémoire, processeur(s), périphériques, flux d'informations, date et heure, fichiers etc.

2.2 Partage des ressources

Le partage des ressources entre tâches concurrentes se fait sous les contraintes suivantes :

- les offrir dans un délai raisonnable (éviter la famine) ;
- optimiser l'utilisation de ces ressources ;
- exclusion mutuelle sur les ressources non partageables ;
- prévenir les blocages.

Le processeur est alloué aux travaux par unités de temps appelées *quantums*. Un processus peut prendre 3 états :

1. **État actif** ou élu : le processus utilise le processeur.
2. **État prêt** ou éligible : le processus pourrait utiliser le processeur si il était libre (et si c'était son tour).
3. **État en attente** ou bloqué : le processus attend une ressource.

Le changement d'état d'un processus peut être provoqué par :

- un autre processus (qui lui a envoyé un signal, par exemple) ;
- le processus lui-même (appel à une fonction d'entrée-sortie bloquante par exemple) ;
- une interruption (fin de quantum, terminaison d'entrée-sortie par exemple).

Le contexte d'un processus doit être sauvegardé quand le processus est désactivé ou bloqué et doit être restauré lorsqu'il est réactivé.

2.3 Processus sous UNIX

La création d'un processus sous Unix est réalisée par l'appel système **fork** :

- Duplication des zones mémoires attribuées au père, le père et le fils ne partagent pas de mémoire.
- Le fils hérite de l'environnement fichiers de son père, les fichiers déjà ouverts par le père sont vus par le fils. Ces fichiers sont partagés.

fork renvoie 0 pour le processus fils et l'identifiant du processus fils crée pour le processus père.

A l'exécution de la fonction **exec**, la région de code est remplacée par le fichier désigné et la pile et les données sont réinitialisées.

2.4 Politiques d'allocation du processeur

Dans un système multitâches, le système d'exploitation doit gérer l'allocation du processeur aux processus. On parle d'*ordonnancement des processus*.

Définition 2.5. Le *temps de réponse* est le temps qui s'écoule entre la soumission d'une requête et la première réponse obtenue. On utilise en général un *temps moyen de réponse* calculé pour tous les processus mis en jeu pendant une période d'observation donnée.

Définition 2.6. Le *temps de service* est le temps qui s'écoule entre le moment où un processus devient prêt à s'exécuter et le moment où il finit de s'exécuter (temps d'accès mémoire + temps d'attente dans la file des processus éligibles + temps d'exécution dans l'unité centrale + temps d'attente + temps d'exécution dans les périphériques d'entrée/sortie).

Définition 2.7. Le *temps d'attente* est le temps passé dans la file des processus éligibles.

Définition 2.8. Le *rendement* d'un système est le nombre de processus exécutés par unité de temps.

Il existe plusieurs politiques d'ordonnancement des processus :

- **First Come First Served (FCFS)** : L'ordre d'exécution est identique à l'ordre d'arrivée dans la file des processus éligibles.
- **Shortest Job First (SJF)** : On exécute d'abord les processus qui ont le temps d'exécution le plus court.
- **Priorité** : On associe une priorité à chaque processus. Les processus sont exécutés par ordre de priorité.
- **Algorithme du tourniquet (Round Robin)** : Le temps est découpé en tranches ou quantum de temps. Un quantum de temps est alloué à chaque processus de la file des processus éligibles à tour de rôle.
- **Tourniquet par priorité avec priorité dynamique** (Algorithme implanté sous UNIX) : Si un processus est arrêté par un événement (entrée-sortie, fonctions wait, pause etc), il est réactivé avec une priorité qui dépend du type de cet événement. La priorité des processus prêts est directement liée à la consommation de temps CPU et au temps qui s'est écoulé depuis leur dernier accès au processeur.

2.5 Synchronisation des processus

Un système d'exploitation dispose de ressources (imprimantes, disques, mémoire, fichiers, base de données etc), que les processus peuvent vouloir partager. Ils sont alors en situation de concurrence vis-à-vis des ressources. Il faut synchroniser leurs actions sur les ressources partagées.

La partie de programme dans laquelle se font des accès à une ressource partagée s'appelle une *section critique*. Dans la plupart des cas l'accès à cette section critique devra se faire en exclusion mutuelle, ce qui implique de rendre indivisible ou atomique la séquence d'instructions composant la section critique. Un outil de synchronisation logiciel est le sémaphore.

Définition 2.9. Un *sémaphore* S associé à une ressource R est un objet composé :

- d'un compteur c qui indique le nombre de processus en attente sur R ;
- d'une file d'attente F où sont chaînés les processus en attente sur la ressource R.

Un sémaphore est accessible par les opérations atomiques suivantes :

- **Initialisation** : $\mathsf{Init}(\mathsf{S}, \mathsf{val})$ où val est le nombre d'éléments de la ressource. On a alors $c = val$ et F vide.
- **P** pour demander un élément de la ressource : $\mathsf{P}(\mathsf{S})$ décrément c. Si $c < 0$ alors le processus est mis dans F et passe à l'état bloqué, sinon un élément de ressource lui est alloué.
- **V** libère un élément de la ressource : $\mathsf{V}(\mathsf{S})$ incrémente c. Si $c \geq 0$ alors on sort un processus de F, il passe dans la file des prêts et on peut lui allouer un élément de ressource.

Définition 2.10. Un *verrou* est un sémaphore dont le compteur est initialisé à 1.

Plusieurs modèles d'accès à des ressources partagées existent :

- **Producteur/Consommateur** : un producteur dépose des messages un par un dans un tampon ; un consommateur retire ces messages un par un. Le producteur et le consommateur doivent synchroniser leurs actions.
- **Lecteur et écrivain** : plusieurs consultations (lectures) en parallèle sont possibles, mais à partir du moment où une mise à jour (une écriture) est en cours, tous les autres accès (en lecture ou en écriture) sont interdits.
- **Rendez-vous** : deux processus P_A et P_B exécutent respectivement les instructions I_1 et J_1 avant d'exécuter les instructions I_2 et J_2.

3 Interruptions et signaux

Nous présentons ici les moyens qui permettent de gérer les événements qui arrivent parallèlement au déroulement d'un processus. La prise en compte matérielle de ces événements est faite sous forme de variation d'état d'une des lignes du bus, appelée *ligne d'interruption*. Le processeur détecte cette variation, le système d'exploitation la prend en charge et la répercute vers le processus concerné. Sous Unix l'outil utilisé pour « remonter » les interruptions vers un processus s'appelle *signal*.

3.1 Interruptions

Les interruptions modifient le cycle de fonctionnement du processeur :

1. Avant d'exécuter une instruction, vérifier si une interruption n'est pas arrivée.
2. Si oui, la traiter :
   (a) l'instruction en cours se termine ;
   (b) l'adresse de l'instruction suivante et le contenu des registres sont sauvegardés dans une pile spécifique : la pile d'interruption
   (c) les interruptions de niveau inférieur ou égal sont masquées ;
   (d) le processeur consulte une table qui contient l'adresse de la procédure à exécuter suivant le type d'interruption qu'il a reçu.

Les interruptions sont plus prioritaires que n'importe lequel des processus.

3.2 Signaux

Définition 3.1. Un *signal* sert à indiquer l'occurrence d'un événement, cet événement peut être d'origine :

- **Interne**, c'est-à-dire provoqué par le processus lui-même : division par zéro, gestion d'une alarme etc.
- **Externe** : provoqué par un autre processus ou le système.

Dans le cas des événements d'origine matérielle, les signaux sont le moyen dont dispose le système pour remonter les interruptions du matériel vers les processus concernés.

L'envoi d'un signal à un processus est matérialisé sous forme d'un bit positionné par le système dans un tableau associé à ce processus. Un émetteur ne peut pas savoir si le processus destinataire a reçu ou non le signal qu'il a émis. On utilise la commande **kill** -signal pid pour émettre un signal.

Un processus actif traite le signal reçu immédiatement. Un processus bloqué traite le signal lors de son passage à l'état actif ou bien sort de l'état bloquant si le signal est plus prioritaire que l'évènement attendu.

Pour chaque signal, un processus peut :

- ignorer ce signal si le système l'y autorise ;
- se contenter du traitement par défaut (la fin du processus destinataire dans la plupart des cas) ;
- exécuter une fonction spécifique (si le système l'y autorise) grâce à la commande **signal**(NumSig, action).

4 Processus et fichiers

4.1 Accès aux ressources

Pour accéder aux différentes ressources dont il a besoin, l'utilisateur en donne le nom, il n'a pas besoin de connaître leur localisation matérielle.

Pour faire des opérations sur les fichiers, il faut ouvrir le fichier en donnant son nom (fonction **open** en C par exemple). Le système associe alors un descripteur au nom du fichier. Ce descripteur peut être un simple numéro.

Sous UNIX, le mécanisme d'accès aux fichiers repose sur l'utilisation de tables :

- Une table centrale et unique (*System Open File Table*), vue par tous les processus. Elle contient une entrée par fichier ouvert.
  - Chaque ouverture du fichier ajoute une entrée dans cette table.
  - Chaque entrée de cette table contient le nom du fichier et la valeur courante du pointeur dans celui-ci.
- Une table par processus. Ces tables contiennent une entrée par fichier ouvert par le processus qui contient un pointeur vers une entrée de la table centrale.

Un processus hérite (par duplication) de la table du processus père.

Sous UNIX, on peut diriger les entrées/sorties avec les symboles > et <. Pour la commande **ls** -l > ls.out :

1. création d'un fichier ls.out ;
2. attribution à ce fichier d'une entrée dans la table générale des fichiers ouverts ;
3. modification du contenu de la première entrée libre de la table du processus : cette entrée pointe maintenant vers l'entrée attribué à ls.out dans la table générale.

4.2 Partage de fichiers

Le partage de fichiers peut se faire de deux façons :

- en utilisant deux pointeurs différents pour parcourir le fichier partagé ;
- en mettant en œuvre un pointeur unique.

Pour gérer la concurrence sur les fichiers partagés (modèle lecteurs/écrivains), Unix permet la synchronisation en mettant à disposition des verrous spécifiques : **lockf** ou **flock**.

Les accès à un fichier déclaré comme *tube* (*pipe* en anglais) sont gérés par Unix suivant le schéma producteur/consommateur. Les **read** et **write** sur ce fichier seront en fait des **consommer** et **déposer** :

- un **read** est bloquant si le tube est vide ;
- un **write** est bloquant si le tube est plein.

La synchronisation est alors assurée par le système.

On crée un *tube standard* par la fonction **pipe**. Il est détruit une fois que tous les processus qui l'utilisent sont terminés. Plusieurs producteurs/consommateurs sont possibles sur le tube. Il n'est accessible que par les descendants d'un ancêtre commun.

Les tubes dits "*nommés*" sont accessibles pour peu qu'on connaisse son nom et qu'on ait les droits d'accès adéquats. Un tube nommé n'est pas détruit quand le processus qui l'a crée se termine. Ils sont crées par les fonctions **mknod** ou **mkfifo**.

5 Gestion de la mémoire

5.1 Définitions

Définition 5.1. La *mémoire* est la juxtaposition de cellules (ou mots-mémoire ou cases-mémoire) à m bits. Chaque cellule peut coder 2^m informations différentes et sont repérées par leur numéro ou *adresse*.

Définition 5.2. Le *gestionnaire de mémoire* gère l'allocation de l'espace mémoire au système et aux processus utilisateurs.

Définition 5.3. Un *octet* est une cellule à 8 bits.

Définition 5.4. Un *mot* est une cellule à 16 32 ou 64 bits.

La gestion de la mémoire a plusieurs caractéristiques :

- **Protection** : un processus ne doit pas pouvoir accéder à l'espace d'adresssage d'un autre.
- **Partage** : s'ils le demandent, plusieurs processus peuvent partager une zone mémoire commune.
- **Réallocation** : les adresses vues par le processus (*adresses relatives*) sont différentes des adresses d'implantation (*adresses absolues*).
- **Organisation logique** : notion de partitions, segmentation, pagination, mémoire virtuelle etc.
- **Organisation physique** : présence d'une hiérarchie de mémoire, ici du plus lent et plus volumineux au plus petit et plus rapide :
  1. disque (de l'ordre du giga-octets) ;
  2. mémoire RAM (méga-octets) ;
  3. cache matériel secondaire (kilo ou méga-octets) ;
  4. cache matériel primaire (dizaine de kilo-octets) ;
  5. registres.

5.2 Allocation contiguë de la mémoire

L'espace mémoire est alloué dynamiquement, de façon contiguë lors de l'implantation des processus en mémoire : on ne sait pas découper l'espace d'adressage d'un programme en plusieurs parties disjointes. Si plusieurs partitions sont susceptibles d'accueillir le processus à charger (partitions libres), plusieurs politiques de placement sont possibles :

- **First Fit** : on place le processus dans la première partition libre dont la taille est suffisante.
- **Best Fit** : on place le processus dans la partition dont la taille est la plus proche de la sienne.
- **Worst Fit** : on place le processus dans la partition dont la taille est la plus grande.

Dans le cas où il n'existe aucune partition libre assez grande, on peut faire du *garbage collecting* (ou ramasse-miettes) qui regroupe les fragments de partitions libres en une partition libre plus large. La *règle de Knuth* affirme de plus qu'à l'équilibre, environ un bloc sur trois est libre.

5.3 Pagination et mémoire virtuelle

La mémoire est une ressource de taille finie, le système d'exploitation va en donner une représentation "logique", la virtualiser, pour qu'elle apparaisse comme une ressource disponible sans limitation.

La première étape de cette virtualisation est la *pagination*. La mémoire est divisée en sous-ensembles de même taille, appelés *pages physiques*. Dans un espace mémoire paginé, les adresses sont structurées en paires : (P_{phy}, D) où P_{phy} est le numéro de page et D est le déplacement, en octets, dans cette page.

L'espace mémoire alloué aux processus est lui aussi divisé en pages, appelées *pages logiques*, numérotées de 0 à N_{proc} pour chaque processus. De même que précédemment, les adresses sont structurées en paires.

Dans un espace paginé, aux algorithmes de placement se substituent des algorithmes de remplacement : les pages logiques sont chargées au fur et à mesure des besoins sur les pages physiques libres, même si ces dernières ne sont pas contiguës. Une *table de pages* est employée pour retrouver une information se trouvant dans une page logique numérotée N_{PL} qui a été chargée sur une page physique N_{phy}. Il y a une table de pages par processus. Si cette table est nommée **TP**, **TP[**N_{PL}**]** donne le numéro de page physique.

Mais la pagination autorise l'exécution d'une application même si la taille de l'espace disponible en mémoire physique est inférieure à celle requise par l'application. L'espace d'adressage d'un processus étant potentiellement plus grand que l'espace d'adressage de la mémoire physique, on parle donc d'*espace d'adressage virtuel* et d'*adresses virtuelles* pour les processus. Finalement, l'accès à une information en mémoire se fait comme suit :

1. Découpage de son adresse en numéro de page virtuelle (N_{PV}) et déplacement.
2. Accès à l'entrée **TP[**N_{PV}**]** de la table de pages du processus. On examine l'*indicateur de validité* :
   (a) Si ce dernier est à 1, passage de page logique en page physique, accès à l'information par traduction de l'adresse logique en adresse physique.
   (b) Sinon, il y a *défaut de page* : on remplace une des pages virtuelles N_{PR} par N_{PV}, on recopie N_{PR} sur le disque et on met à jour **TP[**N_{PV}**]** et **TP[**N_{PR}**]**.
3. On accède à l'information par traduction d'adresse.

Définition 5.5. Un *défaut de page* se produit lorsqu'un processus a besoin de référencer une information qui ne se trouve pas en mémoire. La page logique sur laquelle se trouve cette information devra donc être chargée ou rechargée sur une page physique.

La mémoire virtuelle permet d'exécuter :

- simultanément plusieurs processus dont la somme des espaces d'adressage est supérieure à la taille de la mémoire physiques ;
- une application dont la taille est supérieure à la taille de la mémoire physique.

La mémoire virtuelle donne l'illusion à l'utilisateur (et au processus) qu'il dispose d'un espace d'adressage illimité.

Pour passer de l'adresse virtuelle à l'adresse physique :

1. L'adresse d'une information est divisée en deux champs : numéro de page virtuelle et déplacement dans cette page (ici appelés p et w).
2. Le contenu de l'entrée p de la table de pages donne le numéro de page physique p' où est chargée p. Dans cette entrée figurent également les droits d'accès à la page en lecture, écriture et destruction, ainsi que des indications nécessaires à la pagination. Ces indications sont données par des bits, citons le bit V qui indique si p' est bien un numéro de page en mémoire, ou le bit M qui indique si la page a été modifiée.
3. Pour trouver l'information cherchée on concatène la partie déplacement dans la page au numéro de page physique trouvé.

5.4 Stratégies de remplacement

Ces stratégies permettent de choisir quelle page virtuelle doit être remplacée par la page virtuelle courante. Plusieurs algorithmes de remplacement existent, les plus utilisés sont :

- **Least Recently Used (LRU)** : la page la moins récemment utilisée est remplacée. Cet algorithme implique que l'on conserve une trace des dates d'accès aux pages.
- **Least Frequently Used (LFU)** : la page la moins fréquemment utilisée est remplacée. Cela implique que l'on conserve une trace du nombre d'accès aux pages.
- **First In First Out (FIFO)** : la page la plus ancienne est remplacée. Cela implique que l'on conserve une trace de l'ordre de chargement.