Fonctionnement des systemes d'exploitation

Michel Billaud (michel.billaud@u-bordeaux.fr, michel.billaud@laposte.net)

20 juillet 2020

Table des matières

1 Introduction aux systèmes d'exploitation

Un système d'exploitation est un logiciel qui agit comme intermédiaire entre

- le matériel d'un ordinateur (processeur, mémoire, disque, réseau, …),
- les programmes que fait tourner l'utilisateur.

1.1 Fonctions d'un système d'exploitation

Le système d'exploitation fournit aux programmes d'application un environnement dans lesquels ils peuvent s'exécuter de façon sûre (les programmes sont protégés les uns des autres), commode (le système s'occupe des opérations de bas niveau) et efficace (les ressources sont gérées correctement).

Le système d'exploitation fournit une API (Application Programming Interface), une bibliothèque d'appels par lesquels les programmes d'application lui demandent d'exécuter une action.

Par exemple, le programme C suivant

```c
#include <stdio.h>
int main() {
   printf("Hello, world\n");
   return 0;
}
```

l'appel de `printf` demande l'écriture d'une chaîne sur la "sortie standard" associée au processus, ce qui conduira (peut-être) le système d'exploitation à demander au "gestionnaire d'interface graphique" d'afficher des caractères dans une fenêtre, ce qui passera par des demandes d'accès au pilote de la carte graphique.

Ensuite, en retournant la valeur 0, la fonction `main` signale au système la fin du programme, en fournissant le code de retour qui par convention signifie une fin normale.

Précision : `printf` est définie dans la *bibliothèque standard* **du langage** C. Elle fait appel à la fonction `write` qui fait partie de l'API du noyau du système d'exploitation. De même, le `return` du `main` entraine l'exécution de l'appel système `_exit()`.

1.2 Tâches et processus

On appelle **processus**, ou **tâche** un programme en cours d'exécution sous le contrôle du système d'exploitation.

Ce cours s'intéresse aux systèmes **multi-tâches** en temps partagé qui font tourner plusieurs processus présents en mémoire en même temps. De nos jours, ces systèmes multi-tâches sont présents dans tous les ordinateurs grands et petits (smartphones).[1]

Le système d'exploitation gère les ressources matérielles

- temps accordé aux processus,
- l'espace mémoire accordé aux processus,
- périphériques d'entrée-sortie,
- périphériques de stockage,
- etc.

Cette gestion a été rendue possible par l'introduction successive de divers mécanismes matériels :

- interruptions,
- fonctionnement du processeur dans divers modes
- mécanismes de gestion de la mémoire
- (TODO)

1.3 Problématique des systèmes multi-tâches

Un système multi-tâches "fait tourner" plusieurs processus à la fois, même sur une machine qui n'a qu'un seul processeur, grâce la technique de "temps partagé" (*time slicing*)

Pour cela, le système "fait travailler" une tâche pendant un petit laps de temps[2], puis en fait travailler une autre, etc. Le temps étant ainsi partagé, le travail des tâches avance régulièrement en donnant (à notre échelle) une *illusion* d'avancement simultané.

(Plus de détails plus loin)

Il se pose alors plusieurs problèmes :

- empêcher les processus d'utiliser directement les périphériques
- partager équitablement le temps
- partager la mémoire disponible entre les processus, pour éviter que l'un accède à l'espace mémoire de l'autre

2 Protection : modes du processeur

Dans une machine conçue pour être utilisée avec un système d'exploitation muti-tâches, on doit empêcher les programmes d'application d'accéder directement aux périphériques et à d'autres emplacements mémoire.

Ainsi le système d'exploitation est un intermédiaire obligé pour les actions sur le matériel. Il vérifie que ce qu'on lui demande de faire est acceptable ou non.

2.1 Modes normal et privilégié

Pour cela, les processeurs ont (au moins) deux modes

- un **mode normal** pour l'exécution des programmes d'application,
- un **mode privilégié** pour l'exécution du système.

Certaines instructions (accès aux périphériques par exemple) ne peuvent être exécutées qu'en mode privilégié. En mode normal, toute tentative d'exécuter de telles instructions provoquent une interruption (voir plus loin) "instruction illégale". De même, en mode normal, le processeur ne peut accéder qu'à une partie de la mémoire,la partie réservée au processus qui s'exécute.

2.2 Déroulement d'un appel.

Pour faire un appel, le processus demandeur va exécuter une instruction particulière (SYSCALL/SYSENTER/int 0x80/…) qui va

- dérouter l'exécution vers le point d'entrée du système d'exploitation
- sauver le compteur d'instructions et divers registres sur la pile (qui permettra de savoir où revenir, et dans quel état),
- passer en mode privilégié.

1. Les systèmes mono-tâches existent dans les petits dispositifs d'informatique embarquée qui ne font qu'une chose à la fois. Par exemple une centrale météo qui relève la température, la pression, etc. et transmet les données périodiquement à un serveur.
2. Valeurs typiques : 20-100 ms.

Préalablement, il aura mis dans certains registres le numéro de l'appel voulu, et les paramètres nécessaires.

Le système d'exploitation va vérifier ces données, et, comme il est en mode privilégié, effectuer les actions qui nécessitent d'accéder à la fois aux périphériques et aux données présentes dans l'espace mémoire du processus demandeur.

Quand l'action est terminée, le système d'exploitation provoque le retour en mode normal, à l'instruction qui suit celle de l'appel.

3 Fonctionnement asynchrone, le partage du temps entre processus

3.1 États d'un processus

Situation banale : vous avez plusieurs fenêtre ouvertes à l'écran. Dans un navigateur web, vous cliquez sur un lien, qui tarde à répondre. Vous en profitez pour continuer à faire autre chose dans une autre fenêtre.

Que s'est-il passé ?

- En cliquant sur le lien, le navigateur a envoyé (via la couche réseau) une requête HTTP, et il s'est mis en attente d'une réponse.
- Mais il ne passe pas son temps à surveiller activement l'arrivée de cette réponse.
- Au contraire, le navigateur (ou du moins le processus qui s'occupe de l'onglet) est **bloqué** jusqu'à l'arrivée de la réponse (ou l'expiration d'un délai jugé trop long).

Maintenant, le système peut activer une des autres tâches présentes.

Que se passe-t-il ensuite ?

- Lorsque la carte réseau reçoit un paquet, elle signale au système d'exploitation qu'une trame est arrivée est arrivé.
- le système d'exploitation (la pile TCP/IP) examine cette trame, en extrait un paquet IP, constate que c'est un paquet TCP, et qu'il fait partie d'une connexion ouverte par le navigateur.
- le contenu du paquet est donc transmis au navigateur web, qui est débloqué, et pourra alors traiter la réponse.

3.1.1 États d'un processus

On voit donc 3 états possibles pour les processus

- état **actif** : le processeur est en train d'exécuter une des instructions de la tâche,
- état **bloqué** : la tâche ne peut pas être activée, elle attend un événement pour pouvoir continuer,
- état **prêt** : la tâche n'est pas bloquée, elle pourrait être activée.

3.2 Interruptions

3.2.1 Historique

Les interruptions sont un mécanisme matériel qui a été introduit pour gérer efficacement les "situations exceptionnelles".

- Sur l'UNIVAC 1 (1951), un débordement arithmétique lançait automatiquement l'exécution de l'instruction qui se trouvait en mémoire à l'adresse 0000, au lieu de la suivante. C'est une "exception" [3]
- Sur l'IBM 650 (1954), une "Machine Error" provoque l'exécution de l'instruction qui est à l'adresse 8000, si l'interrupteur "Machine Erreur" n'est pas basculé. C'est la notion de "masquage".
- En 1954, le NBS DYSEAC avait deux compteurs de programme. La réception d'un signal venant d'un périphérique d'entrée sortie faisait passer sur le second compteur.

Référence : https://people.cs.clemson.edu/~mark/interrupts.html

3.2.2 Principe

Les interruptions évitent d'avoir du code qui fait de l'**attente active**, c'est-à-dire qui boucle en sur-

3. On distingue - les **exceptions** qui sont causées par le déroulement anormal d'une instruction (division par zero, etc), - les **interruptions logicielles** déclenchées par une instruction spécifique (`int nnn` sur x86) dans un programme, - les **interruptions matérielles** provenant d'un signal extérieur, ou d'un *timer* interne

veillant les signaux émis par les périphériques dont on attend une réponse (requêtes asynchrones).

Au contraire : le signal qui indique l'arrivée d'une réponse **interrompt** le déroulement normal du programme présent, pour aller exécuter une séquence d'instructions qui se trouve à un endroit pré-déterminé (et que l'on appelle **routine de traitement** de l'interruption).

Pendant ce déroutement, la valeur du compteur de programme est mémorisée (stockée sur une pile, en compagnie du registre d'état et de quelques autres) pour pouvoir, à la fin de la routine de traitement de l'interruption, reprendre l'exécution à l'endroit où elle en était.

3.2.3 Masquage, niveaux, état du processeur

Pour éviter de mélanger le traitement de deux interruptions, une solution simple est d'avoir un indicateur qui dit si le processus accepte d'être interrompu, ou si l'interruption doit être mise en attente.

Plus généralement, il peut exister plusieurs niveaux d'interruption : lorsque le processeur est au niveau N, il ne peut être interrompu que par une interruption de niveau plus élevé.

C'est la notion de **masque d'interruption**. qui est stocké généralement dans le "registre d'état".

3.3 Transitions d'état d'un processus

4 Le partage de l'espace