Appels systèmes

François Trahay

Contents

Qu'est ce qu'un système d'exploitation ?

Rôles d'un système d'exploitation:

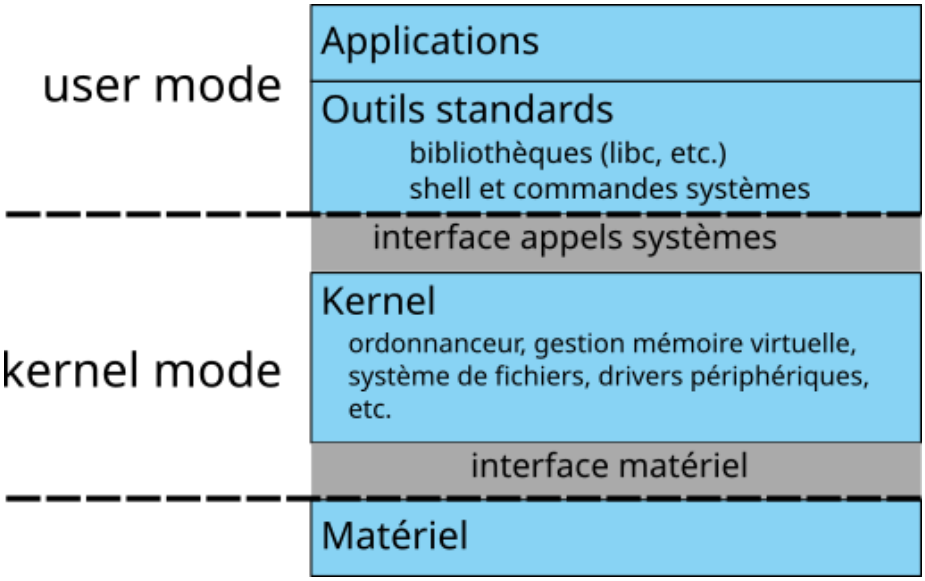
- Abstraire le matériel pour le programmeur
  - Cacher la complexité du matériel
  - Fournir une interface virtuelle de la machine
- Protéger
  - Protection entre utilisateurs (droits d'accès aux fichiers, espaces mémoires des processus séparés)
  - Protection du matériel
- Partager les ressources
  - Partage du CPU (ordonnancement des processus)
  - Accès concurrents à un périphérique

User mode vs. Kernel mode

Cloisonnement entre le mode utilisateur et le mode noyau

- *User mode*

  - certaines instructions sont interdites
  - pas d'accès aux périphériques
  - accès à l'espace mémoire virtuel du processus
- *Kernel mode*
  - accès aux périphériques
  - accès à la mémoire physique



Comment passer en mode noyau ?

2 méthodes:

- interruption
  - interruption logicielle
    - Générée par le processeur en exécutant une instruction
    - division par zéro, accès mémoire illicite
  - interruption générée par le matériel (IRQ)
    - "Je viens de recevoir un message." – la carte réseau
    - "J'ai fini de copier les données sur le disque dur." – le moteur DMA
- appel système
  - l'utilisateur demande à l'OS un service

Lorsqu'une interruption est reçue, le processeur suspend l'exécution du thread, bascule en mode noyau, et appelle la routine traitant l'interruption. Lorsque

la routine se termine, le processeur rebascule en mode utilisateur est reprend l'exécution du thread.

Un appel système consiste à appeler une fonction exécuté en mode noyau. Le passage du mode utilisateur au mode noyau peut se faire en générant une interruption logicielle particulière (par exemple sur les processeurs ARM ou les processeurs x86 32 bits), ou en exécutant une instruction particulière (par exemple, l'instruction `syscall` sur les processeurs x86 64 bits) qui a un effet équivalent. Le kernel exécute alors la fonction correspondant au numéro de l'appel système demandé. Lorsque l'appel système se termine, on sort du traitant d'interruption, et le processeur rebascule en mode utilisateur.

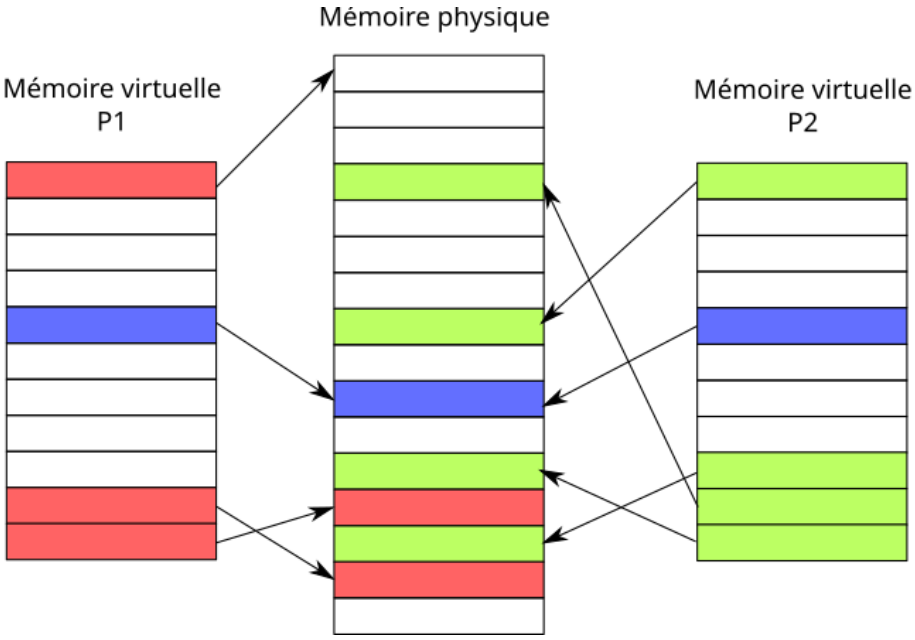
Observer les appels systèmes

La commande `strace` intercepte et affiche les appels systèmes d'un programme:

```
$ strace echo "coucou"
execve("/bin/echo", ["echo", "coucou"], [/* 54 vars */]) = 0
brk(NULL)                               = 0x25d2000
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f619cc01000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
open("tls/x86_64/libc.so.6", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
open("tls/libc.so.6", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
open("x86_64/libc.so.6", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
open("libc.so.6", O_RDONLY|O_CLOEXEC)   = -1 ENOENT (No such file or directory)
open("/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
[...]
fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 2), ...}) = 0
write(1, "coucou\n", 7coucou
)                 = 7
close(1)                                = 0
close(2)                                = 0
exit_group(0)                           = ?
+++ exited with 0 +++
```

Gestion de la mémoire

- mémoire virtuelle des processus découpées en *pages*
- mémoire physique (RAM) découpée en *cadres de pages*
- *pages* projetées sur des *cadres de pages*



Certains cadres de pages sont référencés par plusieurs processus. Cela est possible par exemple si les processus ne font que des accès en lecture à la page. Il s'agit typiquement du code des bibliothèques partagées (`libc.so` par exemple) qui peuvent être chargées par plusieurs processus.

Primitives de synchronisation: les sémaphores

Sémaphore:

- distributeur de "*jetons*"
- 2 opérations:
  - **P** (*"Puis-je"*): prendre un jeton (et attendre si pas de jeton)
  - **V** (*"Vas-y"*): ajouter un jeton (et débloquer un processus)
- Exemple d'utilisation: exclusion mutuelle entre processus

Sémaphore: mise en oeuvre

- Création: `sem_open("/CLE", O_CREAT, S_IRWXU, nb_jetons);`
  - retourne un `sem_t*`
  - CLE est une chaîne commençant par `/`
- Ouverture: `sem_open("/CLE", 0);`
  - retourne un `sem_t`
- Destruction : `sem_unlink(const char *name)`

- Opération **P** : `sem_wait(sem_t* sem)`
- Opération **V** : `sem_post(sem_t* sem)`

L'opération `sem_wait` est bloquante tant qu'il n'y a pas de jeton. L'opération `sem_post` est, elle, non-bloquante, et permet de débloquer un des processus en attente d'un jeton.

Voici un exemple d'utilisation d'un sémaphore. Le progamme `exemple_sem_init` crée un sémaphore, qui peut ensuite être utilisé par le programme `exemple_sem`. Le programme `exemple_sem_unlink` détruit un sémaphore.

```c
#include <stdio.h>
#include <stdlib.h>
#include <semaphore.h>
#include <fcntl.h>
#include <sys/stat.h>

int main(int argc, char**argv) {
  sem_t *sem;

  if (argc != 3) {
    fprintf(stderr, "USAGE = %s cle valeur\n", argv[0]);
    exit(EXIT_FAILURE);
  }

  char*cle=argv[1];
  int valeur = atoi(argv[2]);

  /* Creation et initialisation du semaphore */
  sem = sem_open(cle, O_CREAT, S_IRWXU, valeur);
  if (sem == SEM_FAILED) {
    perror("sem_open failed");
    exit(EXIT_FAILURE);
  }

  printf("Initialisation OK\n");
  int sval = -1;
  /* recupere le nombre de jetons */
  if(sem_getvalue(sem, &sval) <0) {
    perror("sem_getvalue failed");
    exit(EXIT_FAILURE);
  }
  printf("sval = %d\n", sval);

  return EXIT_SUCCESS;
}
```

```c
#include <stdio.h>
```

```c
#include <unistd.h>
#include <stdlib.h>
#include <semaphore.h>

int main(int argc, char**argv) {
  sem_t *sem;

  if (argc != 2) {
    fprintf(stderr, "USAGE = %s cle\n", argv[0]);
    exit(EXIT_FAILURE);
  }

  char*cle=argv[1];

  /* Creation et initialisation du semaphore */
  sem = sem_open(cle, 0);
  if (sem == SEM_FAILED) {
    perror("sem_open");
    exit(EXIT_FAILURE);
  }

  printf("Ouverture OK\n");

  printf("On prend un jeton...\n");
  sem_wait(sem);
  printf("Jeton obtenu.\n");
  sleep(5);

  printf("On relache le jeton\n");
  sem_post(sem);
  printf("Jeton relache\n");

  return EXIT_SUCCESS;
}
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <semaphore.h>
#include <fcntl.h>
#include <sys/stat.h>

int main(int argc, char**argv) {
  sem_t *sem;

  if (argc != 2) {
    fprintf(stderr, "USAGE = %s cle\n", argv[0]);
```

```
    exit(EXIT_FAILURE);
  }

  char*cle=argv[1];

  if(sem_unlink(cle) < 0){
    perror("sem unlink failed");
    abort();
  }

  return EXIT_SUCCESS;
}
```