1 ère année Master : RSD **Année : 2021/2022**

Cours : Programmation réseaux et systèmes

Chapitre 1 : Processus.

fppt.com

Introduction

Les ordinateurs dotés de systèmes d'exploitation modernes peuvent exécuter « en même temps » plusieurs programmes pour plusieurs utilisateurs différents : ces machines sont dites multi-tâches (voir chapitre sur les processus) et multi-utilisateurs.

Un processus (ou process en anglais) est un ensemble d'instructions se déroulant séquentiellement sur le processeur : par exemple, un de vos programmes en C ou le terminal de commande (shell en anglais) qui interprète les commandes entrées au clavier.

fppt.com

Programmes et processus

Il est très important de différencier la notion de programme et la notion de processus. La définition exacte d'un processus, c'est programme en cours d'exécution auquel est associé un environnement processeur et un environnement mémoire. En effet, au cours de son exécution, les instructions d'un programme modifient les valeurs des registres (le compteur ordinal, le registre d'état...) ainsi que le contenu de la pile.
Un programme, c'est une suite d'instructions (notion **statique**), tandis qu'un processus, c'est l'image du contenu des registres et de la mémoire centrale (notion **dynamique**).

fppt.com

Lorsque l'on crée un processus (nous verrons comment faire dans la suite du chapitre), on utilise une fonction qui permet de dupliquer le processus appelant. On distingue alors les deux processus par leur PID. Le processus appelant est alors nommé **processus père** et le nouveau processus **processus fils**. Quant on s'occupe du processus fils, le PID du processus père est noté **PPID** (Parent PID).

Lorsque l'on crée un processus ,on utilise une fonction qui permet de dupliquer le processus appelant. On distingue alors les deux processus par leur PID. Le processus appelant est alors nommé **processus père** et le nouveau processus **processus fils**. Quant on s'occupe du processus fils, le PID du processus père est noté **PPID** (Parent PID).

fppt.com

UID

Les systèmes basés sur Unix sont particulièrement axés sur le côté multi-utilisateur. Ainsi, il existe de très nombreuses sécurités sur les permissions nécessaires pour exécuter telle ou telle action.

C'est pour cela que chaque utilisateur possède un identifiant, sous forme numérique, nommé **UID** (User IDentifier).

fppt.com

GID

Chaque utilisateur du système appartient à un ou plusieurs groupes. Ces derniers sont définis dans le fichier /etc/groups. Un processus fait donc également partie des groupes de l'utilisateur qui l'a lancé. Comme nous l'avons vu avec les UID, un processus dispose donc de plusieurs **GID** (Group IDentifier) **réel**, **effectif**, **sauvé**, ainsi que de GID supplémentaires si l'utilisateur qui a lancé le processus appartient à plusieurs groupes.

Connaître le GID d'un processus n'est pas capital, c'est pourquoi nous ne nous arrêterons pas sur ce point.

fppt.com

Les états d'un processus

Un processus peut avoir plusieurs états :

exécution (**R** pour running) : le processus est en cours d'exécution ;

sommeil (**S** pour sleeping) : dans un multitâche coopératif, quand il rend la main ; ou dans un multitâche préemptif, quand il est interrompu au bout d'un quantum de temps ;

arrêt (**T** pour stopped) : le processus a été temporairement arrêté par un signal. Il ne s'exécute plus et ne réagira qu'à un signal de redémarrage ;

zombie (**Z** pour … zombie) : le processus s'est terminé, mais son père n'a pas encore lu son code de retour.

fppt.com

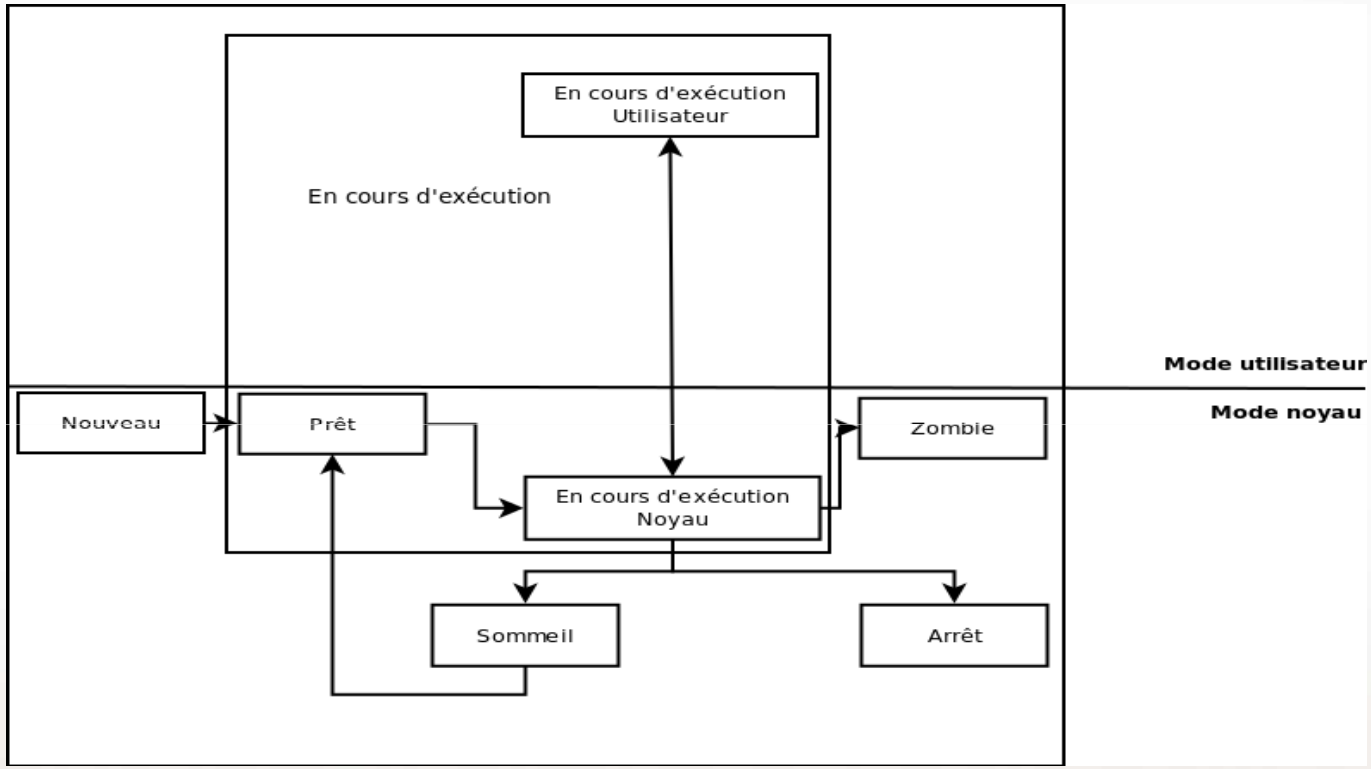

Figure 1 :Les états d'un processus

fppt.com

Implémentation des processus

Pour implémenter les processus, le système d'exploitation utilise un tableau de structure, appelé **table des processus**. Cette dernière comprend une entrée par processus, allouée dynamiquement, correspondant au processus associé à ce programme : c'est le **bloc de contrôle du processus** (Process Control Block, souvent abrégé **PCB**).

fppt.com

Ce bloc contient, entres autres, les informations suivantes :

le PID, le PPID, l'UID et le GID du processus ;

l'état du processus ;

les fichiers ouverts par le processus ;

le répertoire courant du processus ;

le terminal attaché au processus ;

les signaux reçus par le processus ;

le contexte processeur et mémoire du processus (c'est-à-dire l'état des registres et des données mémoires du processus).

Grâce à ces informations stockées dans la table des processus, un processus bloqué pourra redémarrer ultérieurement avec les mêmes caractéristiques.

fppt.com

Création d'un nouveau processus

La fonction fork

La fonction fork permet à un programme en cours d’exécution de créer un nouveau processus.

Le processus d’origine est appelé processus père, et il garde son PID, et le nouveau processus créé s’appelle processus fils, et possède un nouveau PID.

fppt.com

Le processus père et le processus fils ont le même code source, mais la valeur retournée par fork permet de savoir si on est dans le processus père ou fils. Ceci permet de faire deux choses différentes dans le processus père et dans le processus fils (en utilisant un if et un else ou un switch), même si les deux processus on le même code source.

La fonction fork retourne -1 en cas d'erreur, retourne 0 dans le processus fils, et retourne le PID du fils dans le processus père. Ceci permet au père de connaître le PID de son fils.

fppt.com

Exemple

```
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
int main(void)
{
pid_t pid_fils;
pid_fils = fork();
if (pid_fils == -1)
{
printf("Erreur de création du nouveau processus");
exit (1);
} if (pid_fils == 0)
{
printf("Nous sommes dans le fils\n");
printf("Le PID du fils est \%d\n", getpid());
printf("Le PID de mon père (PPID) est \%d", getppid());
}
```


fppt.com

```
else
{
printf("Nous sommes dans le père\n");
printf("Le PID du fils est \%d\n", pid_fils);
printf("Le PID du père est \%d\n", getpid());
}
return 0;
}
```


fppt.com

Terminaison d'un processus

Lorsque le processus fils se termine (soit en sortant du main soit par un appel à exit) avant le processus père, le processus fils ne disparaît pas complètement, mais devient un zombie. Pour permettre à un processus fils à l'état de zombie de disparaître complètement, le processus père peut appeler l'instruction suivante qui se trouve dans la bibliothèque sys/wait.h :

wait(NULL);

fppt.com

La fonction wait est fréquemment utilisée pour permettre au processus père d'attendre la fin de ses fils avant de se terminer lui-même, par exemple pour récupérer le résultat produit par un fils.

Il est possible de mettre le processus père en attente de la fin d'un processus fils particulier par l'instruction

waitpid(pid_fils, NULL, 0);

Le paramètre de la fonction wait, et de deuxième paramètre de waitpid est un passage par adresse d'un entier qui donne des informations sur le statut du fils lorsqu'il se termine : terminaison normale, avortement (par exemple par ctrl-C) ou processus temporairement stoppé).

fppt.com

```
#include <unistd.h>
#include <sys/wait.h>
#include <errno.h>
pid_t pid_fils
int main(void)
{ int status;
pid_fils=fork() ;
if (pid_fils ==-1) {
printf("Problème dans fork()\n");
exit(errno); }
if (pid_fils == 0)
{
printf("Je suis le fils");
puts("Je retourne le code 3");
exit(3);}
else
{
printf("Je suis le père");
printf("Je récupère le code de retour");
wait(&status);}
printf("code de sortie du fils %d : %d\n",pid_fils, WEXITSTATUS(status));}
}
return 0;
}
```


fppt.com

Référence

Rémy Malgouyres Programmation Système En C sous Linux (Debian et Ubuntu)

fppt.com