Programmation système

OS302

CM : O. Aktouf (Oum-El-Kheir.Aktouf@grenoble-inp.fr)
TD : A. Rochedy (Adrien.Rochedy@grenoble-inp.fr)
TP : A. Baudet (Arthur.Baudet@lcis.grenoble-inp.fr)
A. Rochedy

MAN Linux

Commandes et scripts Linux :

I. Parissis (Ioannis.Parissis@grenoble-inp.fr)

- Encadrement et travail personnel
- Evaluation durant l'examen final d'OS302

Objectifs du cours

- Programmation système sous Unix/Linux
- Etudier les principaux appels système, leur utilité …)

Plan du cours

- Gestion des processus
- Gestion des signaux
- Communication à base de tubes
- Les IPC System V
- Les librairies

Bibliographie

Sur la programmation système :
1. Ch. Blaess, « Programmation système en C sous Linux », Ed.Eyrolles, 2005
2. J.-M. Rifflet, « La programmation sous Unix », 3ème édition, McGraw-Hill, 1993
3. Pages `man` d'Unix

Sur les aspects plus généraux liés aux systèmes d'exploitation :
4. A. Tanenbaum, « Modern Operating Systems », Pearson Prentice Hall, 2009
5. A.Silberschatz, P. Galvin, et G. Gagne « Operating System Concepts », John Wiley & Sons. 2005

Evaluation

- Modalités normales :

E1 : Examen session 1 : Sur machine 1h30, notes personnelles de CM, TD et TP autorisées, calculatrice autorisée

CC : TP

E2 : Examen session 2 : Écrit 1h30 ou oral 30min suivant le nombre de candidats à l'épreuve, notes personnelles de CM, TD et TP autorisées, calculatrice autorisée

- Calcul de la moyenne :

N1 = CC*0,2 + E1*0,8

N2 = CC*0,2 + E2*0,8

Introduction

- Généralités sur les systèmes d'exploitation

- Notion de processus

Généralités sur les systèmes d'exploitation

Évolution des systèmes d'exploitation

1. Systèmes primitifs : pas de SE, planification manuelle des travaux et rendement très faible.

2. Traitement par lot : enchaînement automatisé des travaux, mais temps de réponse long.

3. Multiprogrammation : partage de la mémoire centrale par plusieurs travaux ou multiprogrammation, ce qui permet la simultanéité du traitement et des E/S.

4. Réseaux, systèmes répartis, stations de travail : structures décentralisées, partage des ressources, coordination d'activités à distance.

5. Systèmes d'exploitation embarqués autonomes

6. Virtualisation, Cloud Computing, Internet of Things

Généralités sur les systèmes d'exploitation

Exemples de systèmes d'exploitation

1. Système d'exploitation pour ordinateur personnel
   - gestion des fichiers
   - réalisation des E/S
   - exécution et mise au point des programmes
2. Système en temps partagé
   - disponibilité
   - fiabilité et sécurité du matériel et des informations
   - exploitation efficace des ressources informatiques
3. Système de conduite de processus en temps réel
   - contrôle de l'environnement (actionneurs, capteurs, ...)
   - régulation, sécurité, garantie du temps de réponse (gestion de l'interface analogique/numérique, gestion des tâches prioritaires, ...)

Généralités sur les systèmes d'exploitation

Exemples de systèmes d'exploitation

4. Système transactionnel
   - gestion d'une base de données de taille importante
   - transactions fréquentes et nombreuses
   - prévision de plusieurs points d'accès
   - disponibilité, fiabilité, cohérence des informations et tolérance aux pannes
5. Systèmes répartis, embarqués et ubiquitaires
   - évolution des réseaux et des applications mobiles, IoT, Cloud
   - sécurité, disponibilité, hétérogénéité

Notion de processus

Définition

Un processus est un programme en exécution

- Programme : entité statique

- Processus : entité dynamique

Notion de processus

Etat d'un processus

Au cours de son évolution, un processus est dans l'un des états suivants :

- *Nouveau* : le processus est en cours de création
- *En exécution ou actif* : les instructions sont en cours d'exécution
- *En attente ou bloqué* : le processus attend qu'un événement se produise
- *Prêt* : le processus attend d'être affecté à un processeur
- *Terminé* : le processus a fini son exécution

Notion de processus

Composants d'un processus

- *segment code* : suite des instructions

- *segment données* : les données accessibles par le processus

- *segment pile* : pour l'appel de fonctions et le passage des paramètres

- *segment tas* : pour la création de données dynamiques

Notion de processus

Descripteur ou Bloc de Contrôle de Processus (PCB)

Structure contenant toutes les informations relatives à un processus, stockée dans le système dans une table appelée table des processus.

Notion de processus

Informations d'un PCB

- l'identificateur du processus (PID)
- l'état du processus
- le compteur d'instructions : adresse de l'instruction suivante devant être exécutée par le processus
- les registres de l'UC
- la priorité du processus
- informations pour l'ordonnancement
- informations sur la gestion mémoire
- information de comptabilisation : quantité de temps processeur et temps réel utilisés, les limites de temps, …
- informations sur l'état des E/S : liste des périphériques d'E/S alloués au processus, liste des fichiers ouverts, …

Notion de processus

Opérations sur les processus

1. *La création*

Appel système de création de processus
Processus père : celui qui crée, processus fils : celui crée

Les ressources d'un processus fils peuvent être obtenues :
- directement à partir du SE
- être limitées à un sous-ensemble des ressources
du processus père (partagées avec d'autres fils ou non)

Notion de processus

Opérations sur les processus

Deux possibilités par rapport à l'exécution :

- le père poursuit son exécution en concurrence avec son fils
- le père attend jusqu'à ce que certains ou tous ses fils aient terminé leur exécution

Deux possibilités par rapport à l'espace adresse :

- le processus fils est un double du processus père
- le code d'un programme différent est chargé dans le processus fils

Notion de processus

Opérations sur les processus

Cas du SE Unix : appel système **`fork()`**

Création dynamique d'un nouveau processus
Exécution concurrente du nouveau processus créé avec son processus père

```
#include <unistd.h>
pid_t fork(void)
```

processus fils = copie du processus père (copie de l'espace d'adressage)

Notion de processus

Opérations sur les processus

Résultat de l'appel **`fork()`**

- 0 dans le processus fils
- PID du fils dans le père

Retour de **`fork()`** en cas d'échec : -1

Notion de processus

Opérations sur les processus

```
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>

int main (void) {
pid_t pid;
if ((pid=fork()) == -1) {
                perror(“Creation de processus”);
                exit(2);
                }
else    if (pid == 0) {
                printf(“Je suis le fils \n”);
                exit(0)
                }
        else {
                printf(« Je suis le pere. Le PID de mon fils est %d
\n », pid) ;
                exit(0) ;
}
```

Notion de processus

Opérations sur les processus

2. *La terminaison*

- exécution de la dernière instruction
- demande au système de supprimer le processus en utilisant un appel système approprié
- libération des ressources du processus

```
#include <stdlib.h>
void exit(int valeur)
```

- fermeture des fichiers
- vidage des tampons
- appel de _exit()

Notion de processus

Opérations sur les processus

```
#include <stdlib.h>
void _exit(int valeur)
```

- Libération des ressources système
- Le processus passe à l'état zombie
- Envoi du signal SIGCHLD au père
- Réveil du père si bloqué sur **`wait()`**

Notion de processus

Opérations sur les processus

3. *Synchronisation père-fils*

- Un fils qui se termine passe à l'état zombie
- Informations conservées pour un processus zombie
:
  - Code de retour
  - Temps d'exécution dans les 2 modes
  - Son identité et celle de son père

Notion de processus

Synchronisation père-fils

Prise en compte de la terminaison d'un fils par son père

- appels système **`wait()`** et **`waitpid()`**

```
#include <sys/types.h>
#include <sys/wait.h>
pid_t wait(int *pointeur_status)
```

Attend la terminaison d'un fils quelconque.

Notion de processus

Synchronisation père-fils

Effet et retour	Cas
Retour de -1	Pas de processus fils
Valeur *pointeur_status contient des informations sur la terminaison	Au moins un processus fils zombie
Processus appelant bloqué jusqu'à ce que l'un de ses fils devienne zombie ou interruption de l'appel avec retour de -1	L'appelant a des processus fils mais aucun n'est zombie

Notion de processus

Synchronisation père-fils

```
#include <sys/types.h>
#include <sys/wait.h>
pid_t waitpid(pid_t pid, int *pointeur_status,
              int options)
```

Teste la terminaison d'un processus fils en particulier.

Notion de processus

Synchronisation père-fils

Valeur du paramètre pid	Effet
< -1	Tout processus fils dans le groupe |pid|
= -1	Tout processus fils
=0	Tout processus fils du même groupe que l'appelant
> 0	Processus fils d'identité pid

Notion de processus

Synchronisation père-fils

Paramètre options : C'est la combinaison bit à bit des valeurs suivantes :
- WNOHANG : mode non bloquant
- WUNTRACED : recevoir les informations des fils bloqués, en plus de ceux terminés

Retour :
-1 en cas d'erreur en mode bloquant ou non bloquant
0 en cas d'échec en mode non bloquant
Numéro du processus zombie sinon

Notion de processus

Autres fonctions

```
#include <sys/types.h>
#include <unistd.h>

pid_t getpid(void) ;
pid_t getppid(void) ;
```

Notion de processus

Communication avec l'environnement

```
#include <stdlib.h>

char *getenv(const char* nom_variable);

int putenv(const char *chaine)
```

où chaine est de la forme « Nom = valeur »

Notion de processus

Fonction **main()**

```
int main(int argc, char *argv[], char **arge)
```

argc : nombre total de paramètres

argv : tableau contenant argc pointeurs vers les paramètres de la commande + un pointeur NULL
argv[0] est le nom de la commande

arge : liste de pointeurs permettant d'accéder à l'environnement dans lequel le processus s'exécute.
Chaque pointeur permet d'accéder à une chaîne de la forme nom_variable = chaine_valeur