Programmation Système sous UNIX

2ème Ingénieur Informatique

Chapitre1. Contrôle des processus sous Linux

Les processus

Processus UNIX?

- Un programme en cours d'exécution qui utilise les ressources de la mémoire + processeur
- Quelques informations relatives à un processus:
  - PID: Process ID
  - PPID: Parent Process ID
  - User ID (UID) et Group ID (GID) ayant lancé le processus
  - Temps CPU
  - Tables de références des fichiers ouverts, …

Les processus — Caractéristiques

- Espace d'adressage privé
- Fonctionnement en mode utilisateur ou en mode noyau
- Le mécanisme de création des processus est le fork : un processus nouvellement crée est une copie exacte du processus qui l'a créé
- Organisation arborescente des processus
- Un processus est identifié par un numéro unique : pid
- Un processus interagit avec l'extérieur à l'aide des E/S standards
- Un processus peut communiquer avec un ou plusieurs processus à l'aide de tubes de communication (pipes) et les sockets.
- Un processus peut émettre ou recevoir des signaux.

Les processus — Processus init

- C'est le grand-père de tous les processus de **pid = 1**.
- Créer des processus fils à partir de /etc/inittab
- **pstree** : affiche les processus sous forme d'un arbre => facilite la reconnaissance des processus père et fils.

 # pstree –p

```
init(1)-+-atd(468)
        |-bdflush(5)
        |-crond(454)
        |-httpd(440)-+-httpd(450)
        |            |-httpd(451)
        |            |-httpd(452)
        |            |-httpd(453)
        |            |-httpd(455)
        |            |-httpd(456)
        |            |-httpd(457)
        |            `-httpd(458)
        |-keventd(2)
        |-kjournald(7)
        |-klogd(335)
        |-ksoftirqd_CPU0(3)
        |-kswapd(4)
        |-kupdated(6)
        |-login(475)---bash(478)---pstree(518)
        |-sendmail(420)
        |-sshd(385)
        |-syslogd(330)
        `-xinetd(402)
```

Les processus — ps

- Lister les processus en cours d'exécution par la commande **ps**
- Afficher tous les processus du système:

  # ps –A ,ou # ps -ef

- Manipulations:
  - L'utilisateur nommé « user » exécute la commande: $ vi test
  - Afficher les processus de l'utilisateur user:

    # ps –U user
  - Afficher les utilisateurs qui exécutent la commande vi

    # ps –f –C vi

UID	PID	PPID	C	STIME	TTY	TIME	CMD
user	5229	5201	0	18:23	pts/5	00:00:00	vi test
ahmed	5278	4370	0	18:31	pts/0	00:00:00	vi test

Les processus — top

- **top**: Afficher des informations sur l'activité du système en temps réel

```
top - 22:20:34 up  9:54,  2 users,  load average: 0.15, 0.21, 0.21
Tasks: 139 total,   1 running, 138 sleeping,   0 stopped,   0 zombie
Cpu(s):  5.9%us,  1.1%sy,  0.0%ni, 92.8%id,  0.0%wa,  0.0%hi,  0.2%si,  0.0%st
Mem:    504628k total,   498108k used,     6520k free,     7388k buffers
Swap:   995988k total,    49676k used,   946312k free,    94100k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 5557 root      20   0  419m  35m 8404 S    8  7.3  11:11.88 Xorg
10027 kucing    20   0 28072  14m 9076 S    3  3.0   0:00.42 gnome-screensho
 8327 kucing    20   0  215m  79m  23m S    2 16.0   1:34.71 firefox
 5967 kucing    20   0 25608  15m 6664 S    1  3.2   2:12.11 compiz.real
 5979 kucing    20   0 21708 6268 4960 S    1  1.2   0:20.58 gnome-screensav
 5983 kucing    20   0 20976 9620 6672 S    1  1.9   0:06.52 gtk-window-deco
 5984 kucing    20   0 37872  19m  11m S    1  3.9   0:15.54 gnome-panel
 6145 kucing    20   0 24100  12m 7352 S    1  2.6   0:02.70 notification-da
 7036 kucing    20   0  134m  19m  10m S    1  4.0   8:30.97 transmission
10022 kucing    20   0  2416 1156  876 R    1  0.2   0:00.12 top
    1 root      20   0  3056 1776  496 S    0  0.4   0:01.46 init
    2 root      15  -5     0    0    0 S    0  0.0   0:00.00 kthreadd
    3 root      RT  -5     0    0    0 S    0  0.0   0:00.00 migration/0
    4 root      15  -5     0    0    0 S    0  0.0   0:03.94 ksoftirqd/0
    5 root      RT  -5     0    0    0 S    0  0.0   0:00.00 watchdog/0
    6 root      RT  -5     0    0    0 S    0  0.0   0:00.00 migration/1
    7 root      15  -5     0    0    0 S    0  0.0   0:01.14 ksoftirqd/1
```

- Quelques options interactives:
  - Ctrl-L: refresh
  - h: help
  - n: nombre de processus à afficher
  - q: quitter
  - r: (renice) changer la priorité d'un processus

Les processus — Priorités des processus

- **top** ou bien **ps –l** : afficher les priorités des processus
- Le noyau offre plus de temps CPU pour « hight priority process »
- Par défaut les processus d'un utilisateur sont créés avec la priorité 0
- Priorité positif => moins de priorité
- Priorité négatif => plus de priorité
- Le niveau de priorité varie de -20 à 19
- Un utilisateur peut lancer un processus avec un niveau de priorité positif
- **Seul root** peut lancer un processus avec un niveau de priorité négatif

Les processus — Modifier la priorité d'un processus: nice et renice

- Un utilisateur lance cmd1 avec le niveau de priorité +5

```
$ nice -5 cmd1
```

- Seul root peut lancer des processus avec des niveaux de priorité négatifs

```
# nice - -10 vi /etc/hosts.deny

# nice -n -10 vi /etc/hosts.deny
```

- **renice**: modifier la priorité d'un processus

```
# renice 19 501

# renice -10 –u ahmed –p 501
```

Les processus — Primitives de gestion des processus (1)

❑ Identification des processus:

int pid = getpid() int ppid = getppid() : Ces deux primitives fournissent resp. les numéros du processus appelant et celui du processus père.

❑ Identification des propriétaires:

int pid = getuid() int ppid = getgid() : fournissent resp. le numéro du propriétaire, et celui du groupe.

❑ Mise en sommeil d'un processus:

void sleep(int n) : suspend l'exécution du processus appelant pour une durée de n secondes.

void usleep (int micro_seconds)

void pause(): attente du processus appelant jusqu'au réception d'un signal, puis reprise du travail.

Les processus — Primitives de gestion des processus (2)

❑ Terminaison d'un processus:

Un processus se termine lorsqu'il n'a plus d'instructions ou lorsqu'il exécute la fonction :

void exit(int statut)

❑ Elimination d'un processus:

L'élimination d'un processus terminé de la table ne peut se faire que par son père, grâce à la fonction: int wait(int * code_de_sortie)

wait(p) = waitpid(-1, p, 0) : Attente de la terminaison d'un de ses processus fils.

Les processus — Primitives de gestion des processus (3)

- **int wait(int *Y)** : retourne le numéro du processus fils ayant émis le signal au père. Le paramètre passé sert pour récupérer le résultat de la fin du fils. Y étant d'une taille de 16 bits structurés de la façon suivante :

  si l'octet de poids faible (b7-b0) vaut zéro

  alors Le fils a eu une fin normale et l'Octet de poids fort (b15-b8) contient de résultat retourné par le fils.

  sinon Le fils s'est terminé d'une façon anormale (suite à un signal). Les bits b0 à b6 contiennent un entier (numéro du signal qui a tué le fils). Cependant si le bit 7 vaut 1, un fichier "core" est créé.

Les processus — Primitives de gestion des processus (4)

- Création d'un processus :

  int fork()

  - Cette primitive crée un nouveau processus (appelé fils) qui est une copie exacte du processus appelant (processus père)
  - La différence est faite par la valeur de retour de fork(), qui est égale à zéro chez le processus fils, et elle est égale au pid du processus fils chez le processus père. La primitive fork() renvoie -1 en cas d'erreur

- Le processus fils hérite du processus père :
  - La priorité
  - La valeur du masque
  - Le propriétaire
  - Les descripteurs des fichiers ouverts
  - Le pointeur de fichier (offset) pour chaque fichier ouvert
  - Le comportement vis à vis des signaux

Les processus — Exemples

Exemple1:

```
#include<stdio.h>
main()
{int n;
if ((n=fork())<0)
{perror("fork");exit();}
if (n)
printf(" %d : Je suis le processus
père \n",getpid());
else
printf(" %d : Je suis le processus
fils de %d \n",getpid(),getppid());
}
```

Exemple2:

```
#include<stdio.h>
main()
{ int a,b,n;
a=2;b=3;
if ((n=fork())<0)
{perror("fork");exit();}
if (n) a= a+b;
else b=a*b;
printf("%d %d\n",a,b);
}
```

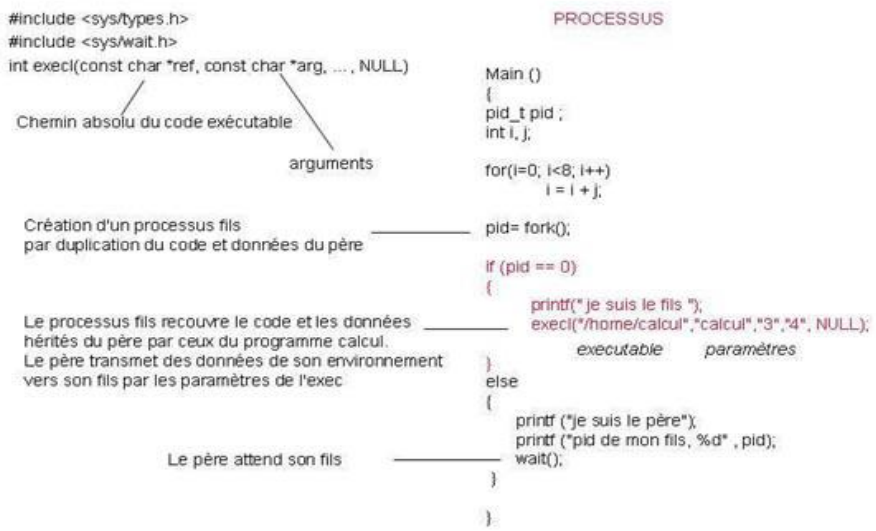

Les processus — Primitives de gestion des processus (5)

❑ Primitives **EXEC** (de recouvrement) : permettent le lancement, par le processus appelant, d'un nouveau programme.

- int **execl**(char *ref, char *arg0, char *arg1, …,char *argn, 0) : Elle lance l'exécution du programme arg0 se trouvant dans le fichier ref avec les arguments arg1 … argn
- Int **execv**(char *ref, char *argv[]) : Cette primitive est similaire à execl mais les arguments du programme sont fournis dans un tableau
- Primitives **execlp** et **execvp** : Ces foncions ont le même effet que execl et execv , mais la recherche de programme se fait dans la liste des exécutables donnée par la variable shell PATH

Les processus — Exemple

Primitives de recouvrement (execl)



Les signaux — Généralités

❑ Les signaux sont des interruptions logicielles envoyés aux processus par le noyau pour indiquer l'arrivée d'un événement anormal. Par exemple, la signalisation des erreurs :

- Accès à une zone mémoire interdite
- Erreur dans une E/S
- Division par zéro, …

❑ Les signaux peuvent être envoyés par les processus

➔ Communication inter-processus

❑ Comportement d'un processus vis à vis des signaux:

- Comportement par défaut: terminaison
- Prendre en compte certains signaux
- Ignorer certains signaux (à l'exception de **SIGKILL**)

Les signaux — Identification d'un signal

- Chaque signal est identifié par un numéro
- Le fichier */usr/include/signal.h* contient la liste des signaux accessibles.
- la commande **kill -l** donne la liste des noms des signaux
- kill [numéro-du-signal] PID: envoyer un signal à un processus. Avec le système Unix/Linux, un processus utilisateur peut également envoyer un signal à un autre processus. Les deux processus appartiennent au même propriétaire, ou bien le processus émetteur du signal est le super-utilisateur.
- Quelques exemples de signaux :
  - SIGINT / 2 signal d'interruption(^C)
  - SIGKILL / 9 destruction d'un processus
  - SIGSEGV / 11 émis en cas de violation mémoire, …

Les signaux — Liste des signaux les plus connus

- **SIGKILL/KILL** de numéro **9**: Il sert pour tuer un processus sans sauvegarder son contexte. Ne peut être ni ignoré ni intercepté.
- **SIGTERM/TERM/15**: tuer de façon correcte un processus (avec sauvegarde de son contexte) => c'est le signal par défaut.
- **SIGINT/INT/2**: interrompre l'exécution, mais le 2 peut être ignoré.
- **SIGHUP/HUP/1**: réservé pour tuer les fils. Par exemple, si un processus père en cours d'exécution est sensé s'arrêter par le 15 => le père envoie le signal 1 à ses fils pour les arrêter car l'arrêt d'un père causera par défaut l'arrêt de ses fils.
- **SIGALRM/14** (Horloge): Émis quand l'horloge d'un processus s'arrête.
- **SIGUSR1/10**: Premier processus utilisateur.
- **SIGUSR2/10**: Deuxième processus utilisateur.
- **SIGSTOP/19**: Stopper le processus dans le cas normal.
- **SIGCONT/18**: Continuer l'exécution du processus
- **SIGCHILD/17**: Mort d'un fils: Envoyé par un fils pour signaler sa fin à son père. Un processus fils ne peut pas se terminer quand son père n'est pas en train de l'attendre. Un tel processus devient un processus zombi.

Les signaux — Envoyer un signal à un processus

- Envoyer SIGTERM aux processus (PIDs 1000 et 1001)
  - $ kill 1000 1001
  - $ kill -15 1000 1001
  - $ kill -SIGTERM 1000 1001
  - $ kill -TERM 1000 1001
- Relecture des fichiers de configuration
  - kill -HUP `cat /var/run/httpd.pid` , ou bien
  - kill -HUP 1015 (on suppose que 1015 est le pid de http)
- Arrêt forcé!
  - kill -9 1000 1001 , ou bien kill –KILL 1000 1001
- Afficher les processus qui s'exécutent en arrière plan (**bg**)
  - # ./firefox &
  - [1] 5788
  - #**jobs**
  - [1]+ Running ./firefox &

Les signaux — Envoyer un signal à un processus

- Vous avez oublié de lancer firefox en arrière plan (bg)

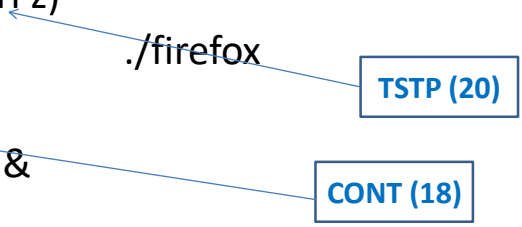
  # ./firefox

  (vous faites ctrl z)

  [1]+ Stopped ./firefox

  # bg

  [1]+ ./firefox &



- Envoyer un signal à des processus indiqués par leurs noms

  Killall –HUP httpd

Les signaux | Primitive d'émission d'un signal

- La primitive int kill(int pid, int sigint) permet à un processus d'envoyer un signal à un autre processus.
  - pid est le n° du processus visé,
  - sigint est le n° du signal employé.
- La valeur de retour est -1 en cas d'erreur et 0 autrement.
- Si sigint=0 alors aucun signal n'est émis.
- pid peut prendre comme valeurs :
  - Si pid > 0 : processus pid.
  - Si pid = 0 : groupe de l'émetteur.
  - Si pid =-1 : tous les processus (seulement root peut le faire).
  - Si pid < 0 : au groupe gid = |pid|

Les signaux | Primitives de capturation et masquage des signaux

Capturation d'un signal

- Un processus peut modifier son comportement aux signaux reçus (càd, prévoir une fonction de traitement de ce signal) par l'appel de la fonction :

  void *signal(int sigint, void (* fonction)(int))

  - sigint est le n° du signal
  - fonction est la fonction qui sera exécutée à l'arrivée du signal sigint

Masquage des signaux

- signal (sigint, SIG_IGN): permet au processus appelant d'ignorer sigint. Le processus rétablira son comportement par défaut lors de l'arrivée de l'interruption (la terminaison)

Les signaux — Exemple 1

```
#include <stdio.h>
#include <signal.h>
#include <unistd.h>

void sighandler(int signum);

int main(void)
{
 char buffer[256];

 if(signal(SIGTERM,&sighandler) == SIG_ERR )
 {
    printf("Ne peut pas manipuler le signal\n");
    exit(1);
 }
 while(1)
 {
   fgets(buffer,sizeof(buffer), stdin);
   printf("Input: %s",buffer);
 }
 return 0;

 }

 void sighandler(int signum)
 {
   printf("Masquage du signal SIGTERM\n");
 }
```


Les signaux — Exemple 1

Nous allons montrer l'exécution de dans deux cet exemple dans deux terminaux, afin de montrer le masquage de signaux et d'essayer de tuer le processus:

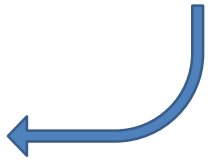
Terminal 1 (pts/0):

```
HOLA
Input: HOLA
KJHKHBNFYNBknbjhhgdt6565
Input: KJHKHBNFYNBknbjhhgdt6565
ls -l
Input: ls -l
who
Input: who
...

Masquage du signal SIGTERM
Masquage du signal SIGTERM
Killed
```

Terminal 2 (pts/1):

```
kill 22096
kill 22096
kill -SIGKILL 22096
```



Les signaux — Exemple 2

Exemple (Ignorance des interruptions clavier)

```
#include<stdio.h>
#include<signal.h>
void hand()
{ printf("Envoyer le signal SIGKILL pour me tuer\n");
}
main()
{
signal(SIGINT,hand);
for(;;);
}
```


Les signaux — Exemple 3

Exemple (horloge)

```
#include<stdio.h>
#include<signal.h>
int    hh,mn,sc;
void tick(int i)
{       sc++;
        if (sc == 60) {sc=0; mn++;
                        if (mn ==60) {mn=0; hh++;
                                        if (hh == 24) hh=0;} }
        alarm(1);
        printf("%d:%d:%d\n",hh,mn,sc);
}
main()
{       signal(SIGALRM, tick);
        alarm(1);
        for(;;); }
```