Interface C/Unix :

Gestion des processus

Gestion des processus

Un processus est un programme en cours d'exécution + son contexte d'exécution

Caractéristiques des processus Unix :

- **Espace d'adressage privé**

  Fonctionne ment 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 *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**

- **Un processus peut émettre ou recevoir des signaux**

Primitives de Gestion des processus

Identification des processus

int pid = getpid() int ppid = getppid()}

Ces deux primitives fournissent respectivement les numéros du processus appelant et celui du processus Père

Identification de proprietaires

int pid = getuid() int ppid = getgid()

Fournissent resp le numéro du propriétaire, et celui du groupe.

Mise en sommeil d'un processus

int sleep(int n)

Suspend l'exécution du processus appelant pour une durée de n secondes

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)

Primitives de Gestion des processus

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 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**

Exemples

Exemple 1

```
#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());
}
```

Exemple 2

```
#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);
}
```

Primitives de Gestion des processus (suite)

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

- **Primitive *execv***

Cette primitive est similaire à execl mais les arguments du programme sont fournis passés dans un tableau

Int execv(char *ref,char *argv[])

- **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

Exemple d'un mini-shell

```
main() (int argc, char *argv[])
{
        if (fork()) wait();
        else {execvp(argv[1], &argv[1]);
        perror("execvp"); }

}
```

La gestion des signaux

Généralités

- **les signaux sont des interruptions logicielles**

- **Par défaut, le processus qui reçoit un signal se termine**

- **Les signaux sont envoyés par le noyau pour indiquer l'arrivée d'un événement :**
  Ex : signalisation des erreurs :
  - violation mémoire
  - erreur dans une E/S

- **Les signaux peuvent être envoyés par les processus**
  ⇨ **Communication inter-processus**

Identification des signaux

- Chaque signal est identifié par un numéro
- Le fichier *include <signal.h>*
- la commande *kill -l* donne la liste de tous les signaux
- Exemples :

SIGINT	**2**	**signal d'interruption(^C)**
SIGKILL	**9**	**destruction d'un processus**
SIGSEGV	**11***	**émis en cas de violation mémoire**
SIGUSR1	**30**	
SIGUSR2	**31**	

Comportement d'un proessus vis à vis des signaux

- comportement par défaut: terminaison
- capturer certains signaux
- ignorer certains signaux

Primitives de gestion des signaux

Emission d'un signal

int kill(int pid, int sigint)

- Cette primtive 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 négative en cas d'erreur**

- **Si** sigint=0 **alors aucun signal n'est émis**
  ⇨ **permet de savoir si le n° pid correspond à un processus**

Capturation d'un signal

Un processus peut modifier son comportement aux signaux reçus 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** signit

Masquage des signaux

signal (sigint, SIG_IGN)

permet au processus appelant d'ignorer sigint

signal (sigint, SIG_IGN)

le processus rétablira son comportement par défaut lors de l'arrivée de l'interruption (la terminaison)

Exemples

Exemple 1 *(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);
signal(SIGQUIT,hand);

for(;;);
}
```

Exemple 2 *(horloge)*

```
#include<stdio.h>
″
#include<signal.h>
″
int hh,mn,sc;
″
void tick(i)
″
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(;;);
}
```

Les signaux : aspect système

⇨ **Quand un signal est envoyé, il devient pendant. Il n'est pas délivré instantanément.**

⇨ **Au bout d'un certain temps(par exemple, quand l'Ordonnanceur donne la main à un processu), le système délivre tous les signaux pendants pour ce processus**

⇨ **Dans l'espace noyau, il y a un vecteur de bits associé à chaque processus. Chaque bit correspond à un signal. Si le bit est à 1, le signal correspondant est pendant.**

⇨ **Ce mode de fonctionnement a deux conséquences très importantes :**

- ⇨ **Il est impossible de prévoir le délai entre l'envoi d'un signal et l'exécution de son handler**
- ⇨ **Si deux signaux identiques sont envoyés au même processus à un court intervalle**

de temps, un seul signal est délivré au processus