TD UNIX

IPC - gestion mémoire

INFO - 2éme année

1 Mémoire partagée

Le but de cette seconde partie est de manipuler les fonctionnalités IPC de partage de segments de mémoire entre processus.

Les commandes, fonctions et appels systèmes importants :

- `ipcs(1)` et `ipcrm(1)` permettent respectivement de visualiser et de supprimer les segments de mémoire partagée (ainsi que les sémaphores et les files de messages, non abordées dans ce TD) ;
- `ftok(3C)` génère une clé nécessaire à l'appropriation d'un objet IPC ;
- `shmget(2)` récupère un identificateur de segment de mémoire partagée ;
- `shmat(2)` attache un segment de mémoire partagée dans l'espace de mémoire du processus ;
- `shmdt(2)` détache un segment de mémoire ;
- `shmctl(2)` opérations diverses telles que la suppression définitive du segment.

Après sa création, un segment de mémoire partagée existe jusqu'à sa destruction explicite. Il faut donc ne pas oublier de le détruire lorsqu'il n'est plus utilisé.

Rappel signaux:

On peut envoyer un signal à un autre processus en appelant la primitive qui envoie le signal de numéro `signum` au processus de PID `pid`.

```
int kill( int pid, int signum )
```

Pour spécifier qu'une fonction C doit être appelée lors de la réception d'un signal non mortel, on utilise :

```
void (*signal(int signum, (void *handler)(int))))(int);
```

L'argument `handler` est une fonction C qui prend un entier en argument et sera automatiquement appelée lors de la réception du signal par le processus. Voici un exemple de fonction traitante :

```
void traite_signal_usr1( int signum )
{
     printf("signal %d recu.\n", signum );
}
```

Cette fonction est installée par l'appel :

```
signal( SIGUSR1, traite_signal_usr1 );
```

Question

1. Un processus père `p1` dispose d'un tableau de 10 caractères (contenant une chaîne quelconque que nous nommerons `A`). De même, un processus fils `p2` dispose d'un tableau de 10 caractères (contenant une chaîne quelconque que nous nommerons `B`). Le but est d'échanger les chaînes respectives de `p1` et `p2` au moyen d'un unique segment de mémoire partagée de taille 15 caractères. Ecrire les codes de ces deux processus pour que chacun récupère les données de l'autre séquentiellement. Le processus p1 écrit ses données en SHM et envoie un signal SIGUSR1 au processus p2. Le processus p2 n'ira lire les données de p1 (et les afficher) puis les remplacer par les siennes qu'à la réception du signal SIGUSR1 puis à son tour envoie un signal SIGUSR2 au processus p1. Enfin à la réception du signal SIGUSR2 le processus p1 lit les données de p2 les affiche et meurt
2. Réalisez les mêmes échanges en gérant les conflits d'accès par sémaphore.

Gestion mémoire:

Soit les deux petits programmes suivants:

/* Programme mem_1_100.c (le nom contient 100 qui est la valeur de N)*/

```
#include<stdlib.h>

#include<sys/times.h>

#define     N     100

int i,j,n1,n2,m[N][N];

main()
{
struct tms, t1,t2;

n1=times(&t1);

for(i=0;i<N;i++)
        for(j=0;j<N;j++)
                m[i][j] = 1; /* version 1, on accède aux éléments en lignes  */

n2=times(&t2);

/* regarder la composition de la structure tms et afficher les temps demandés */
}
```

/* Programme mem_2_100.c (le nom contient 100 qui est la valeur de N)*/

```
#include<stdlib.h>

#include<sys/times.h>

#define     N     100

int i,j,n1,n2,m[N][N];

main()
{
struct tms, t1,t2;

n1=times(&t1);

for(i=0;i<N;i++)
        for(j=0;j<N;j++)
            m[i][j] = 1; /* version 2, on accède aux éléments en colonnes  */
```

```
n2=times(&t2);

/* regarder la composition de la structure tms et afficher les temps demandés */
}
```

1. Testez ces programmes pour différentes valeurs de N (100≤N≤3000), que remarquez-vous. Remplissez le tableau suivant:

	clicks réels 1 processus	**clicks user 1 processus**	**clicks système 1 processus**	**clicks réels 3 processus en //**	**clicks user 3 processus en //**	**clicks système 3 processus en //**
Mem_1_100						
mem_2_100						
mem_1_500						
mem_2_500						
mem_1_1000						
mem_2_1000						
mem_1_2000						
mem_2_2000						
mem_1_2500						
mem_2_2500						

2. Expliquez ces résultats.