TD Programmation Système

Série 4 Mémoires Partagées : Correction

Exercice 1 :

Le processus P1 récupère une chaine de caractères passée en paramètre, l'envoi au processus P2 en utilisant la mémoire partagée. Le processus P2 accède à la mémoire partagée, récupère la chaine et l'affiche en majuscule caractère par caractère.

P1.C

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/shm.h>
#include<unistd.h>
#include<string.h>
#include<ctype.h>
#define key 200

int main(int argc, char *argv[])
{ int idM, i;
char *chaine;

idM=shmget(key,sizeof(char)*1024,IPC_CREAT|0666);
if(idM==-1){
                perror("shmget");
                exit(0);
        }
chaine=(char *)shmat(idM,0,0);

strcpy(chaine,argv[1]);
printf( "%s \n", chaine);

shmdt(chaine);

return 0;
}
```

P2.C

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/shm.h>
#include<unistd.h>
#include<string.h>
#include<ctype.h>
#define key 200

int main(int argc, char *argv[])
{ int idM, i;
char *chaine;

idM=shmget(key,sizeof(char)*1024,IPC_EXCL|0666);
if(idM==-1){
                perror("shmget");
                exit(0);
        }
chaine=(char *)shmat(idM,0,0);

for(i=0;i<strlen(chaine); i++)
printf( "%c", toupper(chaine[i]));

printf( "\n");
shmdt(chaine);
shmctl(idM,IPC_RMID,0);
return 0;
}
```

Exercice 2:

Le processus P1 crée la mémoire partagée de type entier l'utilise pour afficher les multiple de 2 jusqu'au premier multiple de 5. Le processus P2 accède à la mémoire partagée récupère la dernière valeur l'utilise pour afficher les 5 premières valeurs par ajout de 3.

P1.C

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/shm.h>
#include<unistd.h>
#define key 100

int main()
{ int idM,*n;

idM=shmget(key,sizeof(int),IPC_CREAT|0666);
if(idM==-1){
                    perror("shmget");
                    exit(0);
            }
n=(int*)shmat(idM,0,0);

*n=0;
do {    *n=*n+2;
          printf(" %d ",*n);
          }while (*n%5 !=0);
printf("\n");
shmdt(n);

return 0;
}
```

P2.C

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/shm.h>
#include<unistd.h>
#define key 100

int main()
{int idM,*n;

idM=shmget(key,sizeof(int),IPC_EXCL|0666);
if(idM==-1){
                    perror("shmget");
                    exit(0);
            }
n=(int*)shmat(idM,0,0);

do{       *n=*n+3;
          printf(" %d ",*n);
}while (*n%5 !=0);
printf("\n");
shmdt(n);
shmctl(idM,IPC_RMID,0);

return 0;
}
```

Exercice 3: Synchronisation

Les processus P1 et P2 partagent une mémoire pour stocker et accéder aux valeurs de leurs pid et de la valeur n sous forme d'une structure.

```
typedef struct MP{
    int pid1;
    int pid2;
   int n;
}MP;
```

Au début le processus P2 ne peux pas accéder à la mémoire partagée qui n'est pas encore créée le processus P1 crée la mémoire partagée affiche les valeurs de n jusqu'au premier multiple de 5 envoi un message au processus 2 pour le réveiller et se met en pause. Le processus P2 accède à la dernières valeur de n affiche les valeurs obtenues en ajoutant 3 jusqu'au premier multiple de 5 envoi un message au processus 1 pour le réveiller et se met en pause. Les processus P1 et P2 répètent le même traitement jusqu'a la valeur d'arrêt où n=100.

Les processus P1 et P2 redéfinissent le signal SIGUSR1 qui sera utilisé pour la communication. Le nouveau comportement et sans code car le signal est utiliser seulement pour le réveil.

```
Void handler(){}
.....

signal(SIGUSR1,handler);
```

P1 réveille P2 par l'envoi du signal : SIGUSR1

```
    kill(mp->pid2,SIGUSR1);
pause();
```

P2 réveille P1 par l'envoi du signal : SIGUSR1

```
    kill(mp->pid1,SIGUSR1);
pause();
```

P1.C

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

#define key 120

typedef struct MP{
  int pid1;
  int pid2;
  int n;
}MP;

void handler(){
}


int main(){
       int idm;
       MP *mp;
        signal(SIGUSR1,handler);

idm=shmget(key,sizeof(MP),IPC_CREAT|0666);

    if(idm==-1){   perror("shmget");
                   exit(0);}

     mp=(MP*)shmat(idm,0,0);
     mp->pid1=getpid();

     mp->n=0;
     while (mp->n <= 100)
    {    sleep(1); //voir l'Exécution au ralenti
              do {
              mp->n=mp->n+2;
           printf(" %d",mp->n);
           } while (mp->n%5!=0);

       kill(mp->pid2,SIGUSR1);
       pause();
       printf("\n");
            }

shmdt(mp);
return 0;
}
```

P2.C

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


#define key 120

typedef struct MP{
  int pid1;
  int pid2;
  int n;
}MP;

void handler(){
}

int main(){
       int idm;
       MP *mp;
        signal(SIGUSR1,handler);
       idm=shmget(key ,sizeof(MP),IPC_EXCL|0666);

    if(idm==-1){
                       perror("shmget");
                        exit(0);}

     mp=(MP*)shmat(idm,0,0);
             mp->pid2=getpid();

 while (mp->n < 100)
    {  sleep(1); // voir l'Exécution au ralenti
           do {
              mp->n=mp->n+3;
          printf(" %d",mp->n);
          } while (mp->n%5!=0);

       kill(mp->pid1,SIGUSR1);
       pause();
              printf("\n");
    }
shmdt(mp);
shmctl(idm,IPC_RMID,0);
return 0;
}
```