Communication par segment de mémoire partagé

Exercice 1 :
Ecrire un programme qui crée un fils. Le père écrira dans un segment de mémoire partagé et le fils lira ce qu'il a écrit. Le message échangé est une structure contenant le pid du père, le pid du fils et le message échangé.

Exercice 2:
Ecrire deux programmes indépendants. L'un écrira dans un segment de mémoire partagé. L'autre lira ce le rédacteur a écrit.
Modifier le code pour que le rédacteur écrive en SHM ce que l'utilisateur tapera au clavier. Le lecteur écrira sur une autre fenêtre à chaque fois que l'utilisateur entrera un retour chariot.

Corrigé exercice 1 :

```
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
#include <string.h>
#include <sys/types.h>
#include <unistd.h>
#include <signal.h>
#include <sys/shm.h>
#include <errno.h>

 typedef struct{
                        pid_t pidLecteur;
                       pid_t pidRedacteur;
                       int iEntier;
                       char strMessage[50];
 } typeSHM;

int main(int argc , char * argv []) {

    pid_t  fils;
    key_t  key;
    int shm;
    typeSHM * ptrSHM=NULL;

    /* Creation de la clé */
    if ((key = ftok ("SHM",0 )) < 0)
   {
          perror ("ftok");
         exit(-1);
    }
   printf("Clee recuperee\n");
```

```
 /* creation de la SHM */
 if ((shm = shmget (key, sizeof(typeSHM), IPC_CREAT |IPC_EXCL| 0600)) < 0) {
     perror ("shmget");
    exit(-1);
 }
 printf("SHM cree %d\n", shm);

 /* Attachement a la SHM */
 if ((ptrSHM = shmat (shm, 0,0)) == NULL )  {
  perror ("shmat");
  exit(-1);
 }
 printf("SHM Attachee %p\n", ptrSHM);

if ((fils = fork())<0) {
    perror("fork");
   exit(-1);
 }

  if (fils > 0) {
     printf("PERE DEBUT\n");
     ptrSHM->iEntier = 1;
     strcpy(ptrSHM->strMessage, "coucou je suis le pere");
     ptrSHM->pidLecteur = 2;

     /* Ecriture dans la SHM */
     printf ("\n\t\tPERE: j ai ecrit : %s %d\n", ptrSHM->strMessage, ptrSHM->iEntier );
     printf("PERE fin\n");
     sleep(2);
     kill(fils, SIGCONT);
     wait (NULL);
  }
 else   {
     kill(getpid(), SIGSTOP);
     printf("FILS DEBUT\n");
     printf("FILS SE REVEILLE\n");
     printf ("\n\t\tFILS: j ai lu : %s %d\n", ptrSHM->strMessage, ptrSHM->iEntier );
     printf("FILS FIN\n");
 }
 return(0);
}
```

Corrigé exercice 2 :

fichier redacteur.c

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

 typedef struct{
    pid_t pidLecteur;
    pid_t pidRedacteur;
    char strMessage[50];
 } typeSHM;

 void handler (int iSig)
 {

 switch (iSig)
{
  case SIGCONT:
     printf("handler: SIGCONT\n");
     //exit(-1);
     break;
   case SIGSTOP: /* Ne peut etre capturé */
      printf("handler: SIGSTOP\n");
      break;
    case SIGTSTP: /* pareil que SIGSTOP mais peut etre capturé */
      printf("handler: SIGTSTP\n");
      break;
   case SIGALRM:
      system("date +\"%H:%M:%S\"");
      printf("\nhandler: SIGALARM - top\n");
      break;
  default:
      printf("handler: signal inconnu : %d\n", iSig);
      break;
    }
}
```

```
int main(int argc , char * argv [])
{
   struct sigaction action;
   key_t key;
   int shm;
   typeSHM * ptrSHM;
   int iRetour;

printf ("\n\t\tREDACTEUR : je suis %d\n", getpid());

/* Attachement aux signaux */
action.sa_handler = handler;
 if (sigaction (SIGALRM, & action, NULL) != 0)
{
 printf ("erreur attachement signal\n");
exit (-1);
}

/* Creation de la clé */
if ((key = ftok ("SHM1",0 )) < 0)
{
 perror ("ftok");
exit(-1);
 }

/* creation de la SHM */
if ((shm = shmget (key,sizeof(typeSHM),IPC_CREAT | 0600 )) < 0)
{
   perror ("shmget");
exit(-1);
}

/* Attachement a la SHM */
if ((ptrSHM = shmat (shm, NULL,0)) == NULL )
{
perror ("shmat");
exit(-1);
}

/* Ecriture dans la SHM */
ptrSHM->pidRedacteur = getpid();
strcpy(ptrSHM->strMessage, "hello world");

sleep(15);

printf ("\n\t\tREDACTEUR : le redacteur a un pid = %d et il dit : %s\n", ptrSHM->pidLecteur,
```

```
        ptrSHM->strMessage)

/* Destruire le SHM. Il  ne peut être detruit que si personne n'y est attaché
 Appel bloquant*/
 if ((iRetour = shmctl (shm, IPC_RMID, NULL)) < 0 )
{
   perror ("shmctl");
  exit(-1);
}

printf("\n\t\tREDACTEUR : je me termine\n");
return(0);
}
```

fichier lecteur.c

```
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
#include <string.h>
#include <sys/types.h>
#include <unistd.h>
#include <signal.h>
#include <sys/shm.h>
¨
 typedef struct{
    pid_t pidLecteur;
    pid_t pidRedacteur;
    char strMessage[50];
 } typeSHM;

 void handler (int iSig)
 {

    switch (iSig)
   {
      case SIGCONT:
           printf("handler: SIGCONT\n");
           //exit(-1);
           break;
      case SIGSTOP: /* Ne peut etre capturé */
           printf("handler: SIGSTOP\n");
           break;
      case SIGTSTP: /* pareil que SIGSTOP mais peut etre capturé */
           printf("handler: SIGTSTP\n");
           break;
      case SIGALRM:
           system("date +\"%H:%M:%S\"");
           printf("\nhandler: SIGALARM - top\n");
           break;
      default:
          printf("handler: signal inconnu : %d\n", iSig);
         break;
     }
}
```

```
int main(int argc , char * argv [])
{
    struct sigaction action;
    key_t key;
    int shm;
    int iRetour;
    typeSHM * ptrSHM;

   printf ("\n\t\tLECTEUR : je suis %d\n", getpid());

   /* Attachement aux signaux */
   action.sa_handler = handler;
   if (sigaction (SIGALRM, & action, NULL) != 0)
   {
     printf ("erreur attachement signal\n");
     exit (-1);
   }

   /* Creation de la clé */
   if ((key = ftok ("SHM1",0 )) < 0)
   {
      perror ("ftok");
      exit(-1);
   }
   /* creation de la SHM */
  if ((shm = shmget (key, 0,0)) < 0)
  {
     perror ("shmget");
     exit(-1);
   }

    /* Attachement a la SHM */
   if ((ptrSHM = shmat (shm, NULL,0)) == NULL )
  {
     perror ("shmat");
     exit(-1);
   }

  /* Ecriture dans la SHM */
  printf ("\n\t\tLECTEUR : le redacteur a un pid = %d et il a dit : %s\n", ptrSHM->pidRedacteur,
            ptrSHM->strMessage);
```

```
/* Detache de la  SHM  */
if ((iRetour = shmdt (ptrSHM)) < 0 )
{
  perror ("shmdt");
  exit(-1);
}

  sleep(2);

  /* Reveil du redacteur  pour detruire le SHM*/
 kill((ptrSHM->pidRedacteur), SIGALRM);

  printf("\n\t\tLECTEUR : je me termine\n");

  return(0);
}
```