Université de Technologie de Compiègne

SR02

Rapport de TD

3 - Mémoire partagée

Printemps 2015

Anaïs NEVEUX - Romain PELLERIN
Groupe 2
6 avril 2015

Table des matières

Première partie

Séance 1

1 Segment de mémoire partagée

1.1 Question 1

Lorsque le processus père imprime pour la première fois le `tab2[]`, le résultat est différent du fils car au moment du `fork()`, le fils a dupliqué l'espace mémoire du père. Ainsi, lorsque le fils a modifié `tab2[]`, il a en fait modifié une copie du tableau du père, située dans l'espace mémoire propre au fils.

Ensuite, le fils a copié ce tableau `tab2[]` dans l'espace mémoire partagé entre le processus fils et père. Enfin, le père a copié cet espace mémoire partagé dans son propre espace mémoire (à la place de `tab2[]`). C'est pour cela que son `tab2[]` était à la fin identique à celui du fils.

Voici la fonction qui permet de copier un tableau vers un segment de mémoire partagé, et vice-versa :

C

```
...
void copie(int* src, int* dst, int nb) {
    int i;
    for (i = 0; i < nb; i++) {
        dst[i] = src[i];
    }
}
...
int main() {
  int tab2[10];
  int shmid;
  int *adr;
  ...
  shmid = shmget(IPC_PRIVATE,100,IPC_CREAT|0666); // Création d'un espace de mé
     moire partagé de taille 100 bytes
  adr = (int*) shmat(shmid,NULL,0); // Shell memory attachment
  ...
  copie(adr,tab2,10); // adr -> tab2
  ...
}
```

1.2 Question 2

Le principe ici est exactement le même sauf qu'au lieu d'un `fork()`, nous utilisons deux programmes séparés (donc deux `int main()`). Le programme shrpte.c crée le segment de mémoire partagée, écrit dedans et détruit le segment après avoir attendu 30s (laps de temps pendant lequel l'autre programme lit le segment de mémoire partagée et l'affiche).

Il nous a seulement fallu changer quelques arguments fournis à certaines fonctions systèmes.

Fichier shrmem_e.c

```
#define CLE 123456
...
shmid = shmget(CLE,100,IPC_CREAT|0666); // Création d'un espace de mémoire
    partagé de taille 100 bytes
adr = (int*) shmat(shmid,NULL,0); // Shell memory attachment
...
```

Fichier shrmem_r.c

```
#define CLE 123456
...
shmid = shmget(CLE,100,0666); // Accès à espace de mémoire partagé de taille 100
     bytes
adr = (int*) shmat(shmid,NULL,0); // Shell memory attachment
...
```

1.3 Question 3

L'execution du programme shrpte.c déclenche une *segmentation fault*. Cela est totalement logique puisque ce programme obtient, grâce à la mémoire partagée (et au tableau de 5 pointeurs qui s'y trouve), les adresses des vecteurs d'entiers propres au programme strpte.c. En voulant aller lire ces variables, **l'OS refuse l'accès à cette partie de la mémoire au programme.**

Pour que cela fonctionne, il faudrait mettre les vecteurs d'entier dans le segment de mémoire partagée propre aux deux programmes, au lieu d'en faire des variables propres à un seul programme.

Deuxième partie

Séance 2

2 Utilisation de la directive mmap(2)

2.1 Question 1

À partir du code fourni, nous avons créé deux programmes : inific.c qui initialise le fichier « titi.dat », et un programme lirfic.c qui imprime sur la sortie standard le contenu de « titi.dat ».

inific.c

```
#include <fcntl.h>

main() {
    int tab[10] = {11,22,33,44,55,66,77,88,99,1000};
    int fd;
    fd=open("titi.dat",O_RDWR|O_CREAT|O_TRUNC,0666);
    write (fd,tab,10*sizeof(int));
    close(fd);
    return 0;
}
```

lirfic.c

```
#include <stdio.h>
#include <fcntl.h>

main() {
    int i,fd, taille = 10;
    int tab2[taille];

    fd=open("titi.dat",O_RDWR,0666);
    read(fd,tab2,taille*sizeof(int));
    close(fd);

    for (i=0;i<taille;i++)
        printf("%d\n",tab2[i]);

    return 0;
}
```

2.2 Question 2

Nous avons réutilisé les deux fichiers créés précedemment. Nous avons créés les deux nouveaux demandés :

modfic.c

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
```

```c
#include <sys/mman.h>

main() {
    int *adr;
    int fd, stdin;

    fd=open("titi.dat",O_RDWR,0666);
    adr = (int *) mmap(NULL,10*sizeof(int),PROT_READ|PROT_WRITE, MAP_SHARED, fd,
         0);
    if (adr == MAP_FAILED) {
        perror("mmap");
        exit(0);
    }
    while(1) {
        printf("Entrer un nombre (modifiera le tableau): ");
        scanf("%d",&stdin);
        while (getchar()!='\n'); // vide buffer
        if (stdin > 99 || stdin < 0) continue;
        if (stdin == 99)
            exit(0);
        adr[stdin]++;
        printf("Modifié!\n");
    }
    close(fd);
    return 0;
}
```

showfic.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/mman.h>

main() {
    int *adr;
    int fd, stdin, i;

    fd=open("titi.dat",O_RDWR,0666);
    adr = (int *) mmap(NULL,10*sizeof(int),PROT_READ, MAP_SHARED, fd, 0);
    if (adr == MAP_FAILED) {
        perror("mmap");
        exit(0);
    }
    while(1) {
        printf("Entrer un nombre (affichera le tableau): ");
        scanf("%d",&stdin);
        while (getchar()!='\n'); // vide buffer
        if (stdin > 99 || stdin < 0) continue;
        if (stdin == 99)
            exit(0);
        for (i=0;i<10;i++)
```

```
            printf("%d:\t%d\n",i,adr[i]);
    }
    close(fd);
    return 0;
}
```

L'exécution de ces 4 programmes dans deux fenêtres (terminaux) séparés se déroule bien, lirfic.c affiche le fichier final modifié grâce à modfic.c.