Labo 6[1]

Objectif : les appels système liés à la gestion de la mémoire partagée

Introduction

Dans certain cas, il est utile de permettre à des processus de se transmettre des informations. Il y a plusieurs manières de procéder pour permettre d'échanger des informations entre deux, ou plusieurs processus.

L'une d'entre elle est de communiquer au moyen de fichiers. Dans ce cas, les processus voulant envoyer des informations écrivent dans un (ou plusieurs) fichier(s) à une certaine position. Les processus souhaitant utiliser ces informations cherchent leur position dans le fichier et les lisent.

La mémoire partagée par les processus peut aussi être utilisée pour des échanges de données. Suivant le type de processus, les outils utilisés ne sont pas les mêmes.

Dans le cas des processus "classiques", l'espace mémoire d'un processus n'est pas partagé. On utilise alors les segments de mémoire partagés. Dans le cas des processus légers (*threads*) l'espace mémoire est partagé. Dans les deux cas, le partage d'information se fait via des variables partagées par les processus.

Inter Process Communication – IPC

Les IPC permettent de faire communiquer des processus *sans lien de parenté* mais *situés sur la même machine*. Trois types de communication sont possibles :

- les messages, qui permettent à des processus de s'envoyer des messages, avec une gestion des files d'attente de messages,
- les segments de mémoire partagée, qui permettent à plusieurs processus de partager un segment de mémoire dans lequel ils peuvent lire ou écrire des données,
- les sémaphores, qui permettent à plusieurs processus de se synchroniser (pour éviter les sections critiques) et de communiquer des informations sur l'utilisation de ressources.

REMARQUE

Trois laboratoires seront nécessaires pour maitriser ces 3 types de communication.

La particularité de l'utilisation des IPC est qu'ils sont fonctionnellement détachés du système de fichier, ce qui est très rare dans le monde Unix. Ceci implique que les fonctions d'accès seront particulières aux IPC.

Identification des IPC

Les **IPC** sont identifiés de manière unique par un identificateur externe, la **clé**, et par un identificateur interne, le **descripteur**.

Accéder à une IPC suppose, soit de connaître son descripteur (récupéré au moyen des primitives prévues à cet effet), soit de disposer de la clé (qui permet de récupérer cet identificateur).

La clé est une valeur numérique que les processus doivent partager pour communiquer. Plusieurs solutions sont utilisées telles que figer sa valeur dans le code (peu recommandé) ou demander de la générer à partir d'une référence commune à tous les processus (un nom de fichier et un caractère).

Pour cela, on utilise l'appel système `ftok`.

[1] Inspiré du cours d programmation système de Christine Solnon et de celui de Bruno Garcia

ftok	
Librairie	#include <sys/ipc.h>
Prototype	`key_t ftok(char *pathname, char project);`
Commentaires	Permet de générer une clé en utilisant le numéro identifiant (`inode`) du fichier (ou répertoire) indiqué par `pathname` (qui doit exister et être accessible), et les huit bits de poids faible de `project` (qui doit être un caractère non nul) pour créer la clé d'un IPC. Cette clé, de type `key_t` (type défini dans `sys/types.h`) est ensuite utilisable dans les appels système `shmget`, `semget` ou `msgget`, pour récupérer le numéro identifiant de l'IPC en question.

Les commandes liées aux IPC

Ces commandes pourraient s'avérer utiles en cas d'erreur lors de l'exécution d'un programme.

ipcs

ipcs [-s] [-m] [-q] [-u *utilisateur*]		
Signification	Lister l'ensemble des IPC présentes sur un système	
Options	-s	pour afficher uniquement les ensembles de sémaphore
	-m	pour afficher uniquement les segments de mémoire partagée
	-q	pour afficher uniquement les messages
	-u *utilisateur*	pour afficher les ICP lié à un utilisateur (créateur)

La Figure 1 montre la façon dont les informations seront affichées. Comme aucune mémoire partagée, sémaphore ou message n'ont été créés, aucune information n'est affichée !

```
piroc@svr24:~$ ipcs

------ Files de messages --------
clef       msqid      propriétaire perms      octets utilisés messages

------ Segment de mémoire partagée --------
clef       shmid      propriétaire perms      octets     nattch     états

------ Tableaux de sémaphores --------
clef       semid      propriétaire perms      nsems
```

Figure 1 - Résultat de la commande `ipcs`

La Figure 2 montre les informations associées à la mémoire partagée créée dans le cadre d'un programme.

```
------ Segment de mémoire partagée --------
clef       shmid      propriétaire perms      octets     nattch     états
0x50110001 0          piroc      600        8          1
```

Figure 2 - Résultat de la commande `ipcs -m`

ipcrm

ipcrm [-M key | -m id | -Q key | -q id | -S key | -s id] ...		
Signification	Détruire un IPC dont on précise le `pid`	
Description	Permet de détruire un IPC sous réserve que plus aucun processus ne l'utilise. S'il y a encore des processus qui utilisent l'IPC, il est marqué comme étant à détruire, et le système attend que tous les processus aient fini de l'utiliser pour le détruire effectivement.	
Options	-M key	Supprimer le segment de mémoire partagée créé avec `key` après son dernier détachement
	-m id	Supprimer le segment de mémoire partagée identifié par `id` après son dernier détachement
	-Q key	Supprimer la file de messages créée avec `key`
	-q id	Supprimer la file de messages identifiée par `id`
	-S key	Supprimer le jeu de sémaphores créé avec `key`
	-s id	Supprimer le jeu de sémaphores identifié par `id`

Création et récupération d'un segment de mémoire partagée

La création d'un segment de mémoire partagée se fait par un seul processus en utilisant une clé. La création alloue l'emplacement mémoire nécessaire, et retourne un numéro identifiant de cet emplacement mémoire. Les autres processus récupéreront ensuite l'identificateur du segment créé grâce à la clé.

Dans les deux cas (création ou récupération), on utilise l'appel système `shmget`.

shmget	
Librairie	#include <sys/ipc.h> #include <sys/shm.h>
Prototype	`int shmget(key_t key, size_t size, int shmflg);`
Commentaires	Donne le descripteur du segment ayant la clé `key`. `key` est la clé (de préférence générée par `ftok`) qui est attachée (lors de la création) au segment de mémoire partagé. On peut utiliser `IPC_PRIVATE` si on ne veut pas associer une clé particulière au segment. `size` est le nombre d'octets du segment (ce nombre peut-être connu à l'aide de la fonction `sizeof`). `shmflg` est une liste d'arguments séparés par des '|' : • Si `shmflg` est égal à 0, il s'agit d'une **récupération** d'un segment existant • Si `shmflg` est égal à `IPC_CREAT|IPC_EXCL|mode`, il s'agit de la **création** d'un nouveau segment (`IPC_CREAT|IPC_EXCL` permet de garantir l'échec de l'appel si le segment existe déjà, `mode` est un nombre octal indiquant les permissions) Un nouveau segment (de taille `size`) est créé si `key` est `IPC_PRIVATE`, ou bien si les indicateurs de `shmflg` contiennent IPC_CREAT.

Le comportement de `shmget` peut être résumé comme présenté à la Figure 3 :

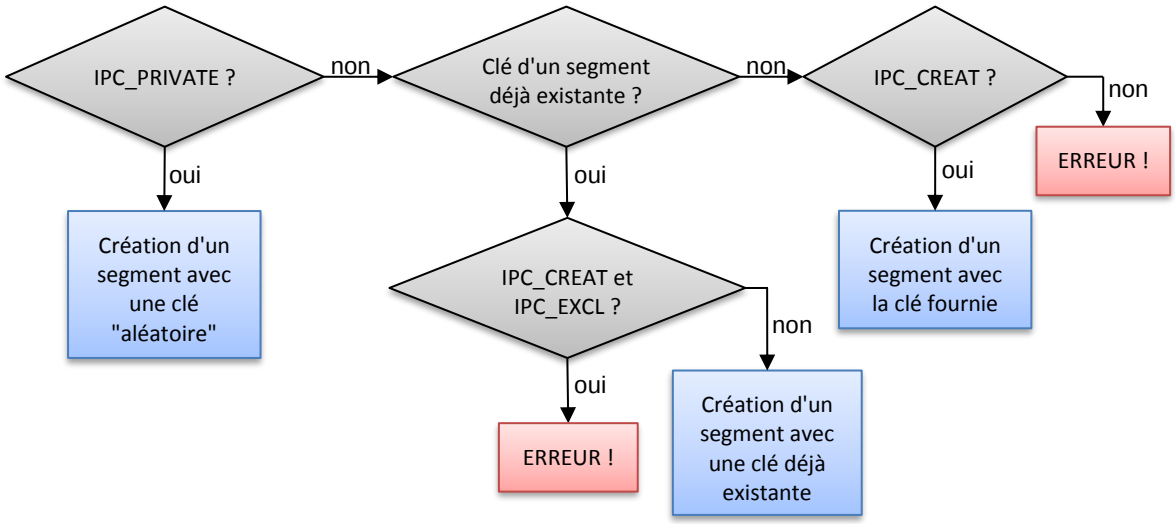

Figure 3 - Comportement de `shmget`

Attachement d'un segment de mémoire partagée à un processus

Pour que les processus puissent effectivement utiliser le segment de mémoire partagée, il faut qu'ils se l'attachent, c'est-à-dire qu'ils mettent dans leur zone mémoire (le tas) un pointeur sur le segment de mémoire partagée. On utilise pour cela l'appel système `shmat`.

shmat	
Librairie	#include <sys/ipc.h> #include <sys/shm.h>
Prototype	`void * shmat(int shmid, const void *shmaddr, int shmflg);`
Commentaires	Attache le segment de mémoire partagée identifié par `shmid` au segment de données du processus appelant. `shmaddr` permet de préciser l'adresse physique à laquelle on veut réaliser l'attachement... En pratique, on n'utilisera pas cette possibilité, et on mettra `shmaddr` à NULL pour indiquer au système qu'il doit se charger de trouver cette adresse lui-même. `shmflg` précise ce que le processus va faire dans ce segment de mémoire : • `SHM_RDONLY` si le processus ne fait que lire, • `SHM_W` si le processus ne fait qu'écrire, • `SHM_R | SHM_W` si le processus lit et écrit. Il retourne l'adresse du premier octet du segment partagé. Cette adresse doit être stockée dans une variable du même type que celui utilisé lors de la création du segment... Le processus pourra alors utiliser le segment partagé, en passant par ce pointeur, comme s'il avait créé dans sa propre mémoire un objet de ce type.

REMARQUE

Pour faire les choses proprement, il faudra convertir la valeur retournée par `shmat` (qui est un `void*`), en une variable de type pointeur sur `T`, où `T` est le type du segment de mémoire partagée.

Détachement d'un segment de mémoire partagée pour un processus

Quand un processus a fini de travailler avec un segment de mémoire partagée, il en avertit le système en utilisant l'appel système `shmdt`.

shmdt	
Librairie	#include <sys/ipc.h> #include <sys/shm.h>
Prototype	`int shmdt(const void *shmaddr);`
Commentaires	Détache le segment de mémoire partagée situé à l'adresse `shmaddr`. Le segment doit être effectivement attaché, et l'adresse `shmaddr` doit être celle renvoyée précédemment par `shmat`. Concrètement, cela revient à supprimer le pointeur vers la mémoire partagée de la zone mémoire du processus appelant (son tas).

REMARQUE

Si un processus "oublie" de détacher ses segments de mémoire partagée, le détachement sera automatiquement fait par le système à la mort du processus.

Contrôle d'un segment de mémoire partagée

Pour obtenir des informations sur un segment de mémoire partagée, ou pour le détruire, on utilise l'appel système `shmctl`.

shmctl	
Librairie	#include <sys/ipc.h> #include <sys/shm.h>
Prototype	`int shmctl(int shmid, int cmd, struct shmid_ds *buf);`
Commentaires	Permet diverses opérations, dont la destruction d'une mémoire partagée. `cmd` peut prendre une des 3 valeurs suivantes : `IPC_STAT`, `IPC_SET` ou `IPC_RMID`. • Si `cmd = IPC STAT`, alors on récupéré dans `*buf` les informations concernant le segment identifié par `shmid`, où `*buf doit` être une structure du type `shmid_ds` décrite ci-dessous. • Si `cmd = IPC_SET`, alors les changements que l'utilisateur a apportés dans les champs `uid`, `gid`, ou `mode` de la structure `ipc_perms` seront appliquées. • Si `cmd = IPC_RMID`, alors le segment partagé sera marqué comme étant "prêt pour la destruction". Il sera détruit effectivement après le dernier détachement (quand le membre `shm_nattch` de la structure `shmid_ds` associée vaudra zéro). L'appelant doit être le créateur du segment, son propriétaire, ou le super-utilisateur.

La structure shmid_ds regroupe les informations suivantes :

```
struct shmid_ds {
   struct ipc_perm shm_perm;        /* Permissions d'accés */
   int shm_segsz;                   /* Taille segment en octets */
   time_t shm_time;                 /* Heure dernier attachement */
   time_t shm_dtime;                /* Heure dernier détachement */
   time_t shm_ctime;                /* Heure dernier changement */
   unsigned short shm_cpid;         /* PID du créateur */
   unsigned short shm_lpid;         /* PID du dernier opérateur */
   short shm_nattch;                /* Nombre d'attachements */
};
```

Le champ shm_perm a la forme suivante :

```
struct ipc_perm {
   key_t key;
   ushort uid;                      /* UID et GID effectifs du propriétaire */
   ushort gid;
   ushort cuid;                     /* UID et GID effectif du créateur */
   ushort cgid;
   ushort mode;                     /* Mode d'accès sur 9 bits de poids faible */
   ushort seq;                      /* Numéro de séquence */
};
```

Exemple d'utilisation d'un segment de mémoire partagée

La première chose à faire est de définir, dans le fichier *entete.h*, le type du segment de mémoire partagée (structure ci-dessous) ainsi que les habituelles bibliothèques et constantes symboliques éventuelles… Ce fichier d'entête sera à importer dans les différents programmes proposés ci-dessous.

```
typedef struct donnees Donnees;
struct donnees {
   int nbEntiers;
   int tabEntiers[256];
};
```

Code pour créer le segment

```
int main(void){
   key_t cle;
   int id;
   Donnees * commun;
   /* Génération de la clé d'accés */
   cle = ftok(getenv("HOME"), 'A');
   if (cle == -1) {
      perror("ftok");
      exit(EXIT_FAILURE);
   }
   /* Création du segment */
   id = shmget(cle, sizeof(Donnees), IPC_CREAT | IPC_EXCL | 0666);
   if (id == -1) {
      perror("Création shm");
      exit(EXIT_FAILURE);
   }
   /* Attachement en écriture du segment à 'commun' */
   commun = (Donnees *) shmat(id, NULL, SHM_W);
   if (commun == NULL) {
      perror("Attachement shm");
      exit(EXIT_FAILURE);
   }
   /* Initialisation des données du segment (facultatif) */
   commun->nbEntiers = 0;
   /* Détachement du segment */
   if (shmdt(commun) == -1) {
      perror("Détachement shm");
      exit(EXIT_FAILURE);
   }
   exit(EXIT_SUCCESS);
}
```

Les informations sur le segment de mémoire attaché au processus est visible via la commande suivante :

```
ipcs -m
```

Le résultat est présenté à la Figure 4.

```
------ Segment de mémoire partagée --------
clef       shmid      propriétaire perms      octets     nattch     états
0x41110001 32769      piroc      666        1028       0
```

Figure 4 - Création du segment de mémoire partagée

Code pour accéder en lecture/écriture au segment

```
void vider(void) {
   char c = getchar();
   while(c != '\n' && c != EOF) {
      c = getchar();
   }
}

int nombreLu(void) {
   int nombre;
   bool valide;
   do {
      scanf("%d", &nombre);
      vider();
      valide = nombre >= 0 && nombre <= 99;
      if(!valide) printf("Nombre entre 0 et 99 !\n");
   } while (!valide);
   return nombre;
}

int main(void){
   key_t cle;
   int id;
   Donnees * commun;
   int nbLu;

   /* Génération de la clé d'accés */
   cle = ftok(getenv("HOME"), 'A');
   if (cle == -1) {
      perror("ftok");
      exit(EXIT_FAILURE);
   }
   /* Récupération de l'adresse du segment */
   id = shmget(cle, sizeof(Donnees), 0);
   if (id == -1) {
      perror("Récupération shm");
      exit(EXIT_FAILURE);
   }
   /* Attachement en lecture et écriture du segment à 'commun' */
   commun = (Donnees *) shmat(id, NULL, SHM_R | SHM_W);
   if (commun == NULL) {
      perror("Attachement shm");
      exit(EXIT_FAILURE);
   }
   /* saisie d'entiers et ajout dans tableau à partir de l'indice nbEntiers */
   printf("Entrez des nombres entiers (0 pour terminer) :\n+ ");
   nbLu = nombreLu();
   while (nbLu != 0) {
      commun->tabEntiers[commun->nbEntiers] = nbLu;
      commun->nbEntiers++;
      printf("+ ");
      nbLu = nombreLu();
   }
   /* Détachement du segment */
   if (shmdt(commun) == -1) {
      perror("Détachement shm");
      exit(EXIT_FAILURE);
   }
   exit(EXIT_SUCCESS);
}
```

Voici un exemple d'utilisation de ce programme, en Figure 5.

```
piroc@svr24:~$ ./ex
Entrez des nombres entiers (0 pour terminer) :
+ 1
+ 2
+ 3
+ 4
+ 5
+ 6
+ 0
piroc@svr24:~$ ipcs -m

------ Segment de mémoire partagée --------
clef       shmid      propriétaire perms      octets     nattch     états
0x41110001 65536      piroc      666        1028       0
```

Figure 5 – Accès en lecture/écriture au segment de mémoire partagée

Code pour accéder en lecture au segment

```
int main(void){
   key_t cle;
   int id, i, total;
   Donnees * commun;
   /* Génération de la clé d'accés */
   cle = ftok(getenv("HOME"), 'A');
   if (cle == -1) {
      perror("ftok");
      exit(EXIT_FAILURE);
   }
   /* Récupération de l'adresse du segment */
   id = shmget(cle, sizeof(Donnees), 0);
   if (id == -1) {
      perror("Récupération shm");
      exit(EXIT_FAILURE);
   }
   /* Attachement en lecture du segment à 'commun' */
   commun = (Donnees *) shmat(id, NULL, SHM_R);
   if (commun == NULL) {
      perror("Attachement shm");
      exit(EXIT_FAILURE);
   }
   /* Utilisation du segment en lecture */
   printf("Il y a %d valeurs dans le tableau d'entiers :\n", commun->nbEntiers);
   total = 0;
   for (i = 0; i < commun->nbEntiers; i++){
      printf("%d\n", commun->tabEntiers[i]);
      total += commun->tabEntiers[i];
   }
   printf("Total = %d\n", total);
   /* Détachement du segment */
   if (shmdt(commun) == -1) {
      perror("Détachement shm");
      exit(EXIT_FAILURE);
   }
   exit(EXIT_SUCCESS);
}
```

La **Error! Reference source not found.** résulte de l'exécution simultanée de deux processus accédant au même segment en lecture/écriture et d'un troisième processus qui lit les informations mémorisées.

```
piroc@svr24:~$ ./ex6-creer
piroc@svr24:~$ ./ex6-ecrire
Entrez des nombres entiers (0 pour terminer) :
+ 1
+ 2
+ 3
+ ^Z
[1]+  Stoppé                 ./ex6-ecrire
piroc@svr24:~$ ./ex6-ecrire
Entrez des nombres entiers (0 pour terminer) :
+ 9
+ 8
+ 7
+ ^Z
[2]+  Stoppé                 ./ex6-ecrire
piroc@svr24:~$ ./ex6-lire
Il y a 6 valeurs dans le tableau d'entiers :
1
2
3
9
8
7
Total = 30
piroc@svr24:~$ fg
./ex6-ecrire
5
+ 6
+ ^Z
[2]+  Stoppé                 ./ex6-ecrire
piroc@svr24:~$ ./ex6-lire
Il y a 8 valeurs dans le tableau d'entiers :
1
2
3
9
8
7
5
6
Total = 41
```

Figure 6 - Accès en lecture au segment de mémoire partagée

Code pour détruire le segment

```
int main(void){
   key_t cle;
   int id;
   /* Génération de la clé d'accés */
   cle = ftok(getenv("HOME"), 'A');
   if (cle == -1) {
      perror("ftok");
      exit(EXIT_FAILURE);
   }


   /* Récupération de l'adresse du segment */
   id = shmget(cle, sizeof(Donnees), 0);
   if (id == -1) {
      perror("Récupération shm");
      exit(EXIT_FAILURE);
   }
   /* Suppression segment */
   if (shmctl(id, IPC_RMID, NULL) == -1) {
      perror("Suppression shm");
      exit(EXIT_FAILURE);
   };
   exit(EXIT_SUCCESS);
}
```

REMARQUE

Le segment ne sera effectivement détruit que lorsque plus aucun processus n'aura ce segment attaché à sa mémoire…

La Figure 7 permet de visualiser ce qu'il se passe lorsqu'on détruit le segment de mémoire partagée avant que les processus n'en soient détachés…

```
piroc@svr24:~$ ps
  PID TTY          TIME CMD
 9908 pts/24   00:00:00 bash
11057 pts/24   00:00:00 ex6-ecrire
11061 pts/24   00:00:00 ex6-ecrire
11504 pts/24   00:00:00 ps
piroc@svr24:~$ ipcs -m

------ Segment de mémoire partagée --------
clef       shmid      propriétaire perms      octets     nattch     états
0x41110001 98304      piroc      666        1028       2

piroc@svr24:~$ ./ex6-detruire
piroc@svr24:~$ ipcs -m

------ Segment de mémoire partagée --------
clef       shmid      propriétaire perms      octets     nattch     états
0x00000000 98304      piroc      666        1028       2          dest

piroc@svr24:~$ fg
./ex6-ecrire
0
piroc@svr24:~$ fg
./ex6-ecrire
0
piroc@svr24:~$ ipcs -m

------ Segment de mémoire partagée --------
clef       shmid      propriétaire perms      octets     nattch     états
```

Figure 7 - Destruction du segment de mémoire partagée

Autre exemple connu : le producteur et le consommateur

Le producteur lit une suite de nombres et effectue le cumul dans une variable en mémoire partagée. Le consommateur affiche périodiquement le contenu de la mémoire partagée. À vous de comprendre le programme…

Pour faire fonctionner cet exemple, il est utile d'ouvrir deux consoles, l'une exécutant le producteur et l'autre exécutant le consommateur. Pour arrêter le consommateur, il faut utiliser la combinaison [CTRL+C].

Le producteur

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

#define KEY_FOR_FTOK  'P' // changer la lettre pour ne pas tous créer la même zone mémoire

typedef struct donnees Donnees;
struct donnees {
   int nb;
   int total;
};

void createSHM();
void dettachSHM();
void destroySHM();

// mémoire partagée
int shmId;
Donnees * pDonnees;

void vider(void) {
   char c;
   c = getchar();
   while(c != '\n' && c != EOF) {
      c = getchar();
   }
}

int chiffreLu(void) {
   int chiffre;
   bool valide;
   do {
      printf("Entrer un chiffre (0 pour terminer) : ");
      scanf("%d", &chiffre);
      vider();
      valide = chiffre >= 0 && chiffre <= 9;
      if(!valide) printf("Chiffre -> 0..9 !\n");
   } while (!valide);

   return chiffre;
}
```

```
int main(void) {
   int id;
   int chiffre;

   printf("Debut du producteur !\n");

   createSHM();

   chiffre = chiffreLu();
   while (chiffre != 0) {
      pDonnees->nb++;
      pDonnees->total += chiffre;

      chiffre = chiffreLu();
   }

   printf("Fin du producteur !\n");

   dettachSHM();

   destroySHM();
   exit(EXIT_SUCCESS);
}

void createSHM() {
   key_t key;
   int i;

   key = ftok(getenv("HOME"), KEY_FOR_FTOK);
   if (key == -1) {
      perror("erreur creation de la cle !");
      exit(EXIT_FAILURE);
   }

   // allocation memoire partagee
   printf("key : %d\n", key);
   shmId = shmget(key, sizeof(Donnees), IPC_CREAT | 0600);
   if (shmId == -1) {
      switch (errno) {
         case ENOENT :
            printf("Pas de segment !\n");
            exit(EXIT_SUCCESS);
         case EEXIST :
            fprintf(stderr, "Le segment existe deja !\n");
            break;
         default :
            perror("shmget");
            exit(EXIT_FAILURE);
      }
         exit(EXIT_FAILURE);
    }

    // attacher la mémoire partagée au processus
    pDonnees = (Donnees *) shmat(shmId, NULL, SHM_R | SHM_W);
    if (pDonnees == NULL || (int)pDonnees == -1) {
        perror("Erreur pour attacher la memoire partagee !");
        shmctl(shmId, IPC_RMID, NULL);
        exit(EXIT_FAILURE);
    }

    // initialisation mémoire partagée
    pDonnees->nb = 0;
    pDonnees->total = 0;
}
```

```
void dettachSHM() {
    if (shmdt((char *)pDonnees) == -1)     {
        perror("Erreur lors de la liberation de la memoire partagee !\n");
        exit(EXIT_FAILURE);
    }
}

void destroySHM() {
    if (shmctl(shmId, IPC_RMID, NULL) == -1) {
        perror("Erreur lors de la destruction de la memoire partagee !\n");
        exit(EXIT_FAILURE);
    }
}
```

Le consommateur

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

#define KEY_FOR_FTOK  'P'
#define DELAI         2

typedef struct donnees Donnees;
struct donnees {
   int nb;
   int total;
};

void createSHM();
void dettachSHM();
void destroySHM();

// mémoire partagée
int shmId;
Donnees * pDonnees;
int encore;

void arret(int signal) {
   encore = 0;
}

int main(void) {
   int id;
   signal(SIGINT, arret); /* si contrôle-C */

   printf("Debut du consomateur !\n");

   createSHM();

   encore = 1;
   while (encore) {
      sleep(DELAI);
      printf("Sous-total %d = %d\n", pDonnees->nb, pDonnees->total);
   }
   printf("Fin du consomateur !\n");

   dettachSHM();
   exit(EXIT_SUCCESS);
}
```

```
void createSHM()
{
   key_t key;
   int i;

   key = ftok(getenv("HOME"), KEY_FOR_FTOK);
   if (key == -1) {
      perror("erreur creation de la cle!");
      exit(EXIT_FAILURE);
   }

   // allocation mémoire partagée
   printf("key : %d\n", key);
   shmId = shmget(key, sizeof(Donnees), IPC_CREAT | 0600);
   if (shmId == -1) {
      switch (errno) {
         case ENOENT :
            printf("Pas de segment !\n");
            exit(EXIT_SUCCESS);
         case EEXIST :
            fprintf(stderr, "Le segment existe déjà\n");
            break;
         default  :
            perror("shmget");
            exit(EXIT_FAILURE);
      }
      exit(EXIT_FAILURE);
   }

   // attacher la mémoire partagée au processus
   pDonnees =(Donnees *) shmat(shmId, NULL, SHM_R | SHM_W);
   if (pDonnees == NULL || (int)pDonnees == -1) {
      perror("Erreur pour attacher la memoire partagee !");
      shmctl(shmId, IPC_RMID, NULL);
      exit(EXIT_FAILURE);
   }

   // initialisation mémoire partagée
   pDonnees->nb = 0;
   pDonnees->total = 0;
}

void dettachSHM() {
   if (shmdt((char *)pDonnees) == -1) {
      perror("Erreur lors de la liberation de la memoire partagee !\n");
      exit(EXIT_FAILURE);
   }
}

void destroySHM() {
   if (shmctl(shmId, IPC_RMID, NULL) == -1) {
      perror("Erreur lors de la liberation de la memoire partagee !\n");
      exit(EXIT_FAILURE);
   }
}
```

L'exécution concurrente (on lance deux fois PUTTY en lançant le producteur d'un côté et le consommateur de l'autre côté) des deux processus produit le résultat observable à la Figure 8.

```
piroc@svr24:~$ ./prod
Debut du producteur !
key : 1343291393
Entrer un entier (0 pour terminer) : 1
Entrer un entier (0 pour terminer) : 2
Entrer un entier (0 pour terminer) : 3
Entrer un entier (0 pour terminer) : 4
Entrer un entier (0 pour terminer) : 5
Entrer un entier (0 pour terminer) : 6
Entrer un entier (0 pour terminer) : 0
Fin du producteur !
```

```
piroc@svr24:~$ ./cons
Debut du consomateur !
key : 1343291393
Sous-total 0 = 0
Sous-total 1 = 1
Sous-total 2 = 3
Sous-total 3 = 6
Sous-total 4 = 10
Sous-total 4 = 10
Sous-total 5 = 15
Sous-total 5 = 15
Sous-total 6 = 21
Sous-total 6 = 21
Sous-total 6 = 21
Sous-total 6 = 21
^CSous-total 6 = 21
Fin du consomateur !
piroc@svr24:~$
```

Figure 8 - Producteur et consommateur

Exercice

Un père démarre 2 processus. Le premier processus va créer une zone mémoire partagée de 50 caractères et y copier un message « message 1 du fils 1\n ». Ensuite il attend que le buffer soit lu par le second processus qui va afficher le message. Le premier processus répètera l'opération 10 fois, il ne doit pas aller trop vite, i.e. il doit attendre que le second processus ait affiché le message pour écrire le message suivant. Pour synchroniser il faut utiliser les signaux.