le cnam

Gestion des Threads et Segments de mémoire partagés

Samia Bouzefrane

samia.bouzefrane@cnam.fr

http://cedric.cnam.fr/~bouzefra

Plan

- La gestion des threads
- La communication par les segments de mémoire partagés

Descripteur des processus Unix

Bloc de contrôle (utilisé par Linux) comporte :

-État du processus
-Priorité du processus
-Signaux en attente
-Signaux masqués
-Pointeur sur le processus suivant dans la liste des processus prêts
-Numéro du processus
-Numéro du groupe contenant le processus
-Pointeur sur le processus père
-Numéro de la session contenant le processus
-Identificateur de l'utilisateur réel
-Identificateur de l'utilisateur effectif
-Politique d'ordonnancement utilisé
-Temps processeur consommé en mode noyau et en mode utilisateur
-Date de création du processus
-etc.

Création d'un processus Unix

- **Un processus Unix** = { un espace d'adressage et une unité d'exécution unique }

- **Contexte d'un processus Unix** :
  - pid,
  - répertoire de travail,
  - descripteurs,
  - propriétaires,
  - implantation en mémoire,
  - registres,
  - priorité d'ordonnancement,
  - masque des signaux,
  - pile d'exécution

⇒ **Cela rend coûteuse la création d'un processus**

Création de threads Unix

- **Thread Unix**: - processus léger
  - permet de multiplier l'implémentation d'activités caractérisées par des contextes plus légers que ceux des processus Unix

- **Contexte d'exécution d'une thread Unix**:
  - registres,
  - priorité d'ordonnancement,
  -masque des signaux,
  -pile d'exécution.

- **L'espace virtuel d'une thread :**
  - code exécuté par la thread,
  - données définies dans la thread et
  - une pile d'exécution propre à la thread.

Primitives de manipulation de threads Unix

- ***Création et activation d'une thread*** **:**

```
int pthread_create(
pthread_t *idptr, /* ptr sur la zone où sera retournée l'identité de la thread */
pthread_attr_t attr,/* attributs de la thread: pthread_attr_default  ou 0 */
void *(*fonc) (void *), /* pointeur sur la fonction exécutée par la thread */
void *arg            /* pointeur sur le(s) argument(s) passé(s) à la thread */
) ;
```

- ***Terminaison d'une thread*** **:**

```
void pthread_exit (
void *status /* ptr sur le résultat retourné par la thread */
) ;
```

- ***Libération des ressources d'une thread*** **:**

```
int pthread_detach(
pthread_t *idptr    /* pointeur sur l'identité de la thread */
) ;
```

- ***Attente de la terminaison d'une thread*** **:**

```
int pthread_join(
pthread_t id,       /* identité de la thread attendue */
void *status        /* pointeur sur le code retour de la thread attendue */
) ;
```

Exemple1 de threads Unix

```
/* Trois_Th.c */
#include <pthread.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
int compteur[3];
/* fonction executee par chaque thread */
void *fonc_thread(void *k) {
   printf("Thread numero %d : mon tid est %d\n",(int) k, pthread_self());
   for(;;) compteur[(int) k]++;
}
main() {
int i, num; pthread_t tid[3];
 /* creation des threads */
for(num=0;num<3;num++)
{pthread_create(tid+num, 0, fonc_thread,(void *) num);
 printf("Main: thread numero %d creee: id = %d\n",num, tid[num]);
  }
 usleep(10000); /* attente de 10 ms */
printf("Affichage des compteurs\n");
for(i=0;i<20; i++) {
  printf("%d \t%d \t%d\n", compteur[0], compteur[1], compteur[2]);
  usleep(1000);
/* attente de 1 ms entre 2 affichages */
 }
exit(0); }
```

Exemple2 de threads Unix

```
#include <pthread.h>
#include <stdio.h>
#include <errno.h>
typedef struct {
                int x, y;
                } data;
/* fonction executee par  chaque thread
*/
void *mul(data *ptrdata)
{
pthread_exit((void *)((ptrdata->x) *
(ptrdata->y)));
}
void main(int argc, char *argv[]){
int i;
int a, b, c, d;
pthread_t pth_id[2];
data donnees1;
data donnees2;
int res1, res2;
if(argc < 5) { perror("\007Nbre
d'arguments incorrect"); exit(1);}
a=atoi(argv[1]); b=atoi(argv[2]);
c=atoi(argv[3]); d=atoi(argv[4]);
donnees1.x=a; donnees1.y=b;
donnees2.x=c; donnees2.y=d;
/* creation des threads */
   pthread_create(pth_id,   0, (void
*(*)())mul, &donnees1);
   pthread_create(pth_id+1, 0, (void
*(*)())mul, &donnees2);
/* Attente de la fin de la thread 1 */
   pthread_join(pth_id[0], (void **)
&res1);
/* Attente de la fin de la thread 2 */
   pthread_join(pth_id[1], (void **)
&res2);
/* Affichage du resultat a*b + c*d */
   printf("Resultat =%d\n", res1+res2);
/* Suppression des ressources des
threads */
   pthread_detach(&pth_id[0]);
   pthread_detach(&pth_id[1]);
exit(0);
}
```

Thread (3)

compilation avec :

```
gcc thread_exo1.c -o exo1 -lpthread
```

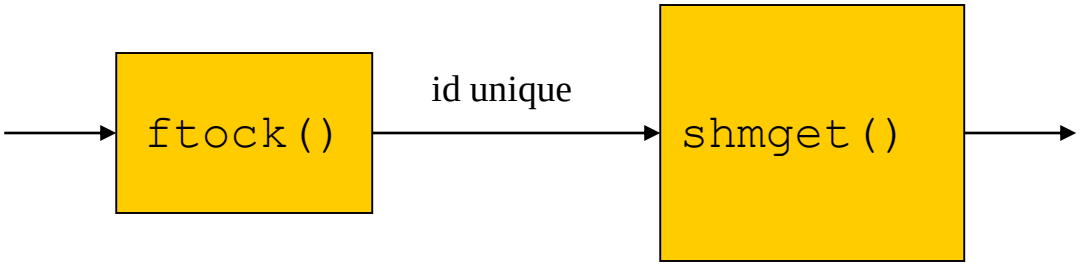
Les segments de mémoire partagés : introduction

- IPC : Inter Proccess Communication – System V d'Unix
- Repris dans POSIX
- Communication entre processus sur la même machine
- Librairies C:

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
```

Création de segments de mémoire partagés

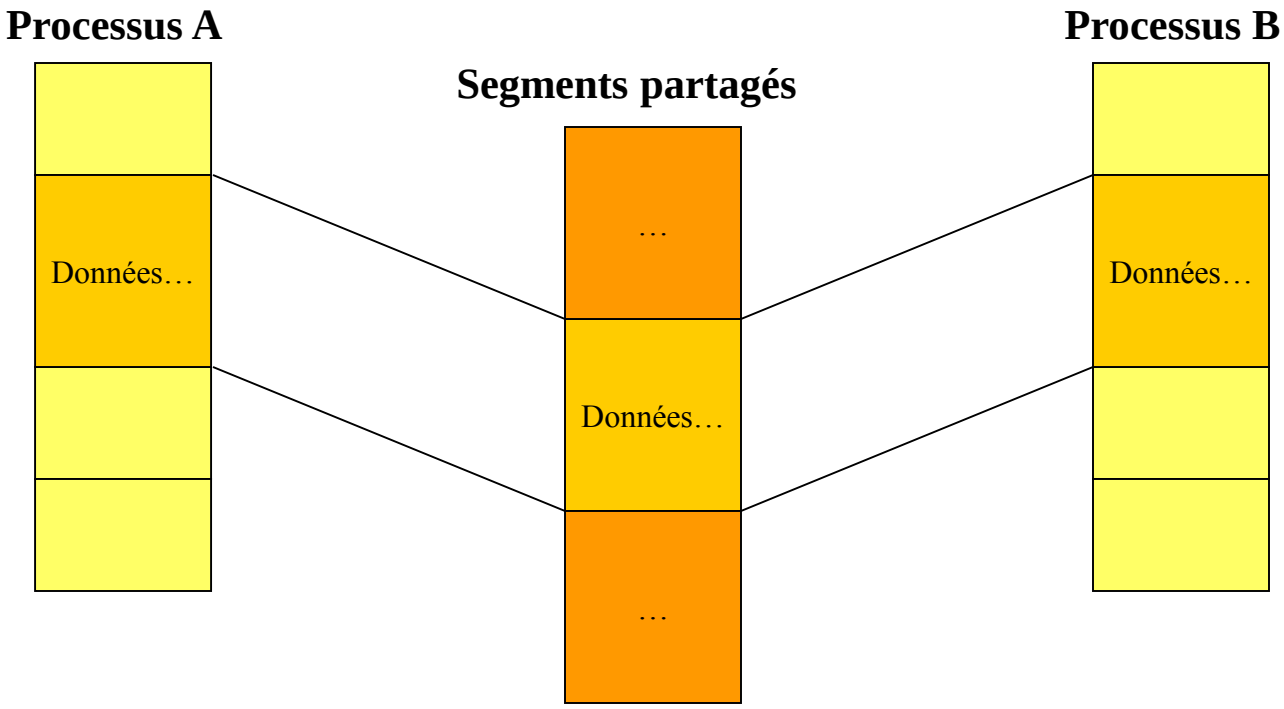
- Pour créer un segment de mémoire, on attache à ce segment un certain nombre d'informations.



Mode de création	Description
IPC_CREAT	Création si l'objet n'existe pas déjà. Sinon aucune erreur
IPC_CREAT | IPC_EXECL	Création si l'objet n'existe pas déjà. Sinon erreur
IPC_PRIVATE	Création si l'objet demandé et association de celui-ci à l'identificateur fourni
Aucun mode précisé	Erreur si l'objet n'existe pas

Partage de données

- Les segments de mémoire partagés permettent à deux processus distincts de partager physiquement des données.



Obtention d'un segment de mémoire

La primitive *shmget*()

```
int shmget( key_t cle, int taille, int options);
```

- *cle* est la clé obtenu par la fonction *ftok*()
- *taille* est le nombre d'octets du segment
- *options* indique les droits d'accès au segment, PC_CREAT | 0666 par exemple

Attachement d'un segment de mémoire

```
void  *shmat(int shmid, const void *adr, int options);
```

- *shmid* est l'identificateur du segment
- *adr* est l'adresse virtuelle à laquelle doit être implanté le segment dans l'espace adressable du processus; une valeur 0 permet de laisser le choix de cette adresse au système.

- *options* peut valoir SHM_RDONLY pour s'attacher un segment en lecture seule.

Détachement d'un segment de mémoire

La primitive *shmdt*() détache un segment de l'espace adressable d'un processus.

```
int shmdt( const void *adr );
```

adr est l'adresse retournée par *shmat*.

Un segment non détaché sera détaché automatiquement lorsque tous les processus qui se le sont attachés se terminent.

Suppression d'un segment de mémoire

int shmctl(int shmid, int option, struct shm_ds *p)

permet d'agir sur un segment partagé.

Si l'*option* est égale à :

- IPC_RMID : suppression du segment identifié par *shmid* (la suppression sera effective lorsque aucun processus n'est plus attaché au segment.

Exemple d'utilisation/1

```
// shm_prod.c
struct donnees { int nb; int total; };

void main(void) {
   key_t cle;   // dans stdlib.h
   int id;
   struct donnees *commun;
   int reponse;

   cle = ftok(getenv("HOME"), 'A');
   id = shmget(cle, sizeof(struct donnees), IPC_CREAT | IPC_EXCL | 0666);
   commun = (struct donnees *) shmat(id, NULL, SHM_R | SHM_W); // shm.h
   commun->nb = 0;
   commun->total = 0;
   while (1) {
      printf("+ ");
      scanf("%d", &reponse);
      commun->nb++;
      commun->total += reponse;
      printf("sous-total %d = %d\n", commun->nb, commun->total);
   }
   printf("---\n");
   err = shmdt((char *) commun); /* détachement du segment */
   err = shmctl(id, IPC_RMID, NULL); /* suppression segment */
}
```

Exemple d'utilisation/2

```
// shm_cons.c
#define DELAI 2
struct donnees { int nb; int total; };

void main(void)
{
   key_t cle;   // dans stdlib.h
   int id;
   struct donnees *commun;

   cle = ftok(getenv("HOME"), 'A');
   id = shmget(cle, sizeof(struct donnees), 0);
   commun = (struct donnees *) shmat(id, NULL, SHM_R); // shm.h
   while (1) {
      sleep(DELAI);
      printf("sous-total %d = %d\n", commun->nb, commun->total);
   }
   printf("---\n");
   err = shmdt((char *) commun);
}
```

Exemple d'utilisation/3

```
$ ./shm_prod
+ 10
sous-total 1 = 10
+ 20
sous-total 2 = 30
+
```

Le producteur lancé, on saisit 10 pour augmenter la valeur en mémoire

```
$ ./shm_cons
sous-total 0 = 0
sous-total 0 = 0
sous-total 1 = 10
sous-total 1 = 10
sous-total 2 = 30
sous-total 2 = 30
sous-total 2 = 30
---

$ ./shm_cons
sous-total 2 = 30
sous-total 2 = 30
sous-total 2 = 30
sous-total 2 = 30
---

$
```

Le consommateur lancé la première fois affiche les valeurs. Puis on quitte.
Le deuxième lancement du consommateur indique bien la dernière valeur en mémoire.