Travaux pratiques informatique
Module Système d'exploitation
Année 2006-2007
TP N°3 : Mémoire Partagée

Mathieu.Barcikowski@univ-savoie.fr
Pierre.Hyvernat@univ-savoie.fr
Cyril.Vachet@univ-savoie.fr

1. Objectifs du TP

Ce TP a pour objectif de vous faire appréhender les concepts de **mémoire partagée** à travers d'un petit jeu de réflexion : le **morpion n par m**. Votre travail consistera à implémenter un ensemble de programme en **langage C** tout en utilisant des bibliothèques Unix spécifiques de gestion de la mémoire partagée. Ce travail donnera lieu à un compte-rendu et à des programmes qui seront notés.

2. Instructions de rendu

Pour le compte-rendu, la clarté, la pertinence et les justifications serviront de base à la notation. Il sera par ailleurs nécessaire de mettre en avant les concepts du cours que vous aurez appliqué. Vous insisterez bien sur les structures de donnée utilisées et vous pourrez si vous le jugez nécessaire utiliser des schémas explicatifs.

Au moment de rendre votre travail, créer une archive au format *.zip (nom binôme 1 – nom binôme 2.zip) contenant :

- **Le compte-rendu au format Adobe PDF ou Mocrosoft Word (*.doc) (tout compte-rendu utilisant un autre format ne sera pas lu !!!)**
- Le code source commenté programme.
- Le script Bash.
- Les autres programmes éventuels permettant de répondre aux questions impératives ou à la question subsidiaire.
- Les instructions d'installation et les ressources nécessaires à l'installation dans le cas de l'utilisation d'une librairie non-standard.

Conseils :

– Ne négligez pas un rapport bien construit: il facilitera fortement votre évaluation. Par ailleurs, un bon rapport indique en principe un bon programme. Cela fonctionne aussi dans l'autre sens ;).
– Mettez toujours en avant les points forts (et faibles) de votre réalisation.
– Il est interdit de s'échanger du code ou des bouts de code entre binôme.
– Indiquez les étapes à effectuer dans le cas de l'utilisation d'une libraire non-standard à Linux.

3. Introduction aux pré-requis du TP

(A) Mécanismes d'IPC (Inter Processus Communications)

Les mécanismes d'IPC permettent de faire communiquer et/ou de synchroniser n'importe quel couple de processus locaux (de la même machine). Il existe 3 mécanismes d'IPC : files de messages, segments de mémoire partagée, sémaphores. Dans un système Unix, ces mécanismes sont purement mémoire. Le système prend en charge la gestion de ces objets. C'est lui qui tient à jour les tables qui les contiennent. (http://www-igm.univ-mlv.fr/~dr/CS/node197.html)

Dans ce TP, nous nous intéressons plus particulièrement à la mémoire partagée. Dans le but de créer un segment de mémoire partagée, le programme va émettre une requête au système pour créer ou accéder à un segment. Ce dernier est en effet chargé d'assurer la gestion de ces segments. Il fait cela à l'aide d'une table, la table des segments (ici simplifiée), suivante :

Clé (n°)	**Description de segment partagé**			**Taille**	**Adresse logique de début de segment**
	PID	**...**	**Compteur (refs)**		
Key_x	Pid_Créateur	...	1	N	Adresse_y

Exemple simplifié de la table des segments

Un segment de mémoire partagée est une partition de la mémoire logique identifié au minimum par une clé unique. Une entrée intègre par ailleurs une structure décrivant les spécificités du segment : le pid de son créateur et surtout un compteur comptant le nombre de processus référençant le segment. Ainsi, à chaque requête, si le segment n'existe pas, il est alors créé. Si le segment est présent dans la table, alors le système d'exploitation fourni au processus l'identifiant du segment; le processus « s'attache » alors ce segment à son propre contexte d'exécution ce qui a pour effet d'incrémenter le compteur de référence. Enfin, lorsqu'un processus se « détache » du segment, il perd la référence à ce dernier et informe le système. Lorsque le système nettoie la table des segments partagés, il libère la mémoire et supprime les entrées pour les segments dont le compteur a pour valeur zéro.

Pour informations complémentaires, les attributs du tableau sont disponible à l'adresse : http://www.linux-kheops.com/doc/man/manfr/man-html-0.9/man2/shmctl.2.html

Sous les systèmes à base d'Unix, des bibliothèques normalisées de gestion de la mémoire partagée sont mise à disposition du développeur. Cette bibliothèque s'articule autour des primitives suivantes :

<sys/shm.h>

ftok() : Cette fonction convertit le nom et le chemin d'accès d'un fichier existant, ainsi qu'un identificateur de projet en une clé IPC Système V de type **key_t**. (http://www.linux-kheops.com/doc/man/manfr/man-html-0.9/man3/ftok.3.html)

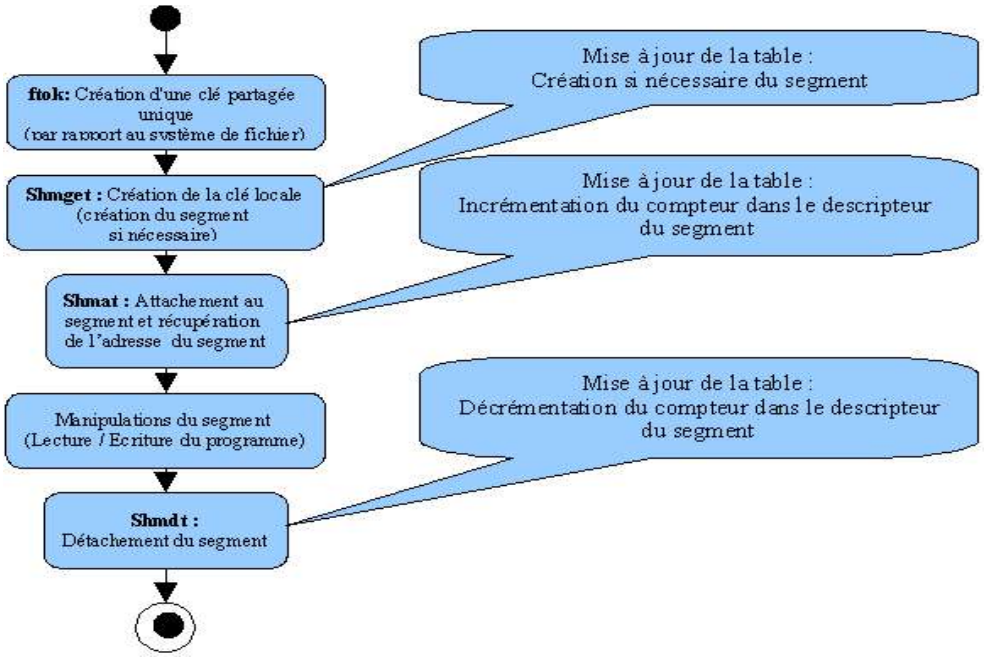
shmget(): shmget() renvoie l'identificateur du segment de mémoire partagée associé à la valeur de l'argument *clé*. Un nouveau segment mémoire, de taille *size* arrondie au multiple supérieur de **PAGE_SIZE**, est créé si *clé* a la valeur **IPC_PRIVATE** ou si aucun segment de mémoire partagée n'est associé à *clé*, et **IPC_CREAT** est présent dans *shmflg*. (http://www.linux-kheops.com/doc/man/manfr/man-html-0.9/man2/shmget.2.html)

shmat(): La fonction **shmat** attache le segment de mémoire partagée identifié par **shmid** au segment de données du processus appelant. (http://www.linux-kheops.com/doc/man/manfr/man-html-0.9/man2/shmdt.2.html)

shmdt() : La fonction **shmdt** détache le segment de mémoire partagée situe à l'adresse indiquée par *shmaddr*. Le segment doit être effectivement attaché, et l'adresse *shmaddr* doit être celle renvoyée précédement par **shmat.** (http://www.linux-kheops.com/doc/man/manfr/man-html-0.9/man2/shmdt.2.html)

shmctl() permet à l'utilisateur d'obtenir des informations concernant un segment de mémoire partagée, ainsi que de fixer le propriétaire le groupe et les permissions d'accès à ce segment. Cette fonction permet également de détruire un segment. (http://www.linux-kheops.com/doc/man/manfr/man-html-0.9/man2/shmctl.2.html)

De manière classique, la démarche pour accéder à la mémoire partagée avec les librairies normalisées se fait de la manière suivante :



(B) Exemple commenté

On vous propose de mettre le programme suivant :

```
/* shmdemo.c -- utilisation de la mémoire partagée en C; exemple simple :
- si le programme est utilisé sans aucun argument, il lit le contenu  du segment de mémoire partagée.
- le le programme est utilisé avec un argument de type « chaîne de caractères », alors le programme
stocke cet argument dans le segment partagé */

/* inclusion des librairies standards... */
#include <stdio.h>           /* "standard Input Output" */
#include <stdlib.h>          /* "standard libraries" */
#include <string.h>          /* fonctions standards sur les chaînes */

/* Les trois libraries suivantes sont nécessaires pour les opérations « système ». En particulier,
"ipc" veut dire "Inter Process Communications", et "shm" veut dire "shared memory". */
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

/* On va seulement utiliser 1024 octets (1K) pour le segment partagé : on définit une constante de
taille. */
#define TAILLE_SHM 1024

/* la fonction principale prend des arguments : argc contient le nombre d'arguments, et argv est un
pointeur vers les arguments */

int main(int argc, char *argv[])
{
    key_t key;
    int shmid;
    char *data;

/* erreur s'il y a plus que deux arguments */
    if (argc > 2) {
        fprintf(stderr, "utilisation : shmdemo [chaîne]\n");
        exit(1);
    }
/* création de la clé */
    if ((key = ftok("shmdemo.c", 'R')) == -1) {
        perror("ftok");
        exit(1);
    }
    /* allocation du segment de mémoire partagée */
    if ((shmid = shmget(key, TAILLE_SHM, 0644 | IPC_CREATE)) == -1) {
        perror("shmget");
        exit(1);
    }
/* récupération d'un pointeur vers ce segment de mémoire partagée */
    data = shmat(shmid, (void *)0, 0);
    if (data == (char *)(-1)) {
        perror("shmat");
        exit(1);
    }
/* suivant le nombre d'arguments (0 ou 1), on lit le contenu du segment ou on écrit l'argument
dessus. */
    if (argc == 2) {
        printf("écriture de \"%s\" en mémoire partagée\n", argv[1]);
        strncpy(data, argv[1], TAILLE_SHM);  // copie de l'argument
    } else {
        printf("contenu de la mémoire partagée : \"%s\"\n", data);
    }
/* avant de terminer, on « détache » le segment de mémoire partagée. */
    if (shmdt(data) == -1) {
        perror("shmdt");
        exit(1);
    }
    return 0;
}
```

Questions :

- Compilez le programme ci-dessus.
- Décrivez en vos mots à vous ce que fait ce programme tout en pensant à faire le lien entre le code et les concepts (ainsi que les étapes) vues en partie A.
- Lancez le programme. Testez les commandes « ipcs » et « ipcrm -m clé » et déduisez à quoi servent ces commandes ?

(C) Traitement des paramètres passés en ligne de commande

Pour traiter les options passées en paramètres, vous pouvez utiliser la fonction C getopt.

Lien pour l'exemple simple d'utilisation de la librairie getopt.

http://www.gnu.org/software/libc/manual/html_node/Example-of-Getopt.html

4. Le jeu du morpion

Les règles sont simples : il s'agit d'un plan de jeu n par m où deux joueurs s'affrontent pour aligner un certain nombre de pions (crois vs ronds). Le joueur i gagne lorsqu'il a aligné ses pions; il y a égalité lorsque aucun des deux joueurs n'ont pu se départager. Pour plus d'informations, voir par exemple sur http://perso.orange.fr/therese.eveilleau/pages/jeux_mat/textes/morpion.htm pour les règles).

Le nombre de pions devant être alignés pour déterminer le gagnant est défini dynamiquement lors de l'exécution de la fonction d'évaluation.

Le programmes écrit C permettra les opérations suivantes :

Arguments	Description	Priorité
-i	Initialisation du plateau de jeu et création du segment de mémoire partagée	Obligatoire
-a	Affichage du plateau de jeu	Obligatoire
-x i j	Joue une croix dans le plateau de jeu, à la position i j	Obligatoire
-o i j	Joue un rond dans le plateau de jeu, à la position i j	Obligatoire
-e i j	Efface un coup	Obligatoire
-s i j	Supprime le plateau de jeu de la mémoire	Obligatoire
-r	Réinitialisation du jeu sans supprimer le segment de mémoire partagée	Obligatoire

-v n	Vérifie l'état du jeu : détermination du joueur gagnant le cas échéant, avec l'argument n permettant de définir le nombre de pions devant être alignés.	Subsidiaire
-h	Affichage d'un message d'aide rappelant les commandes à l'utilisateur	Subsidiaire

Exemple de scénario d'exécution standard :

- *Initialisation du segment de mémoire partagée.*
- *Le joueur 1 joue un croix*
- *Affichage de la grille de jeu*
- *Le joueur 2 joue un rond*
- *Affichage de la grille de jeu*
- *Evaluation de la grille pour déterminer le gagnant*
- <…>
- *Nettoyage de la grille pour faire un nouvelle partie*
- <…>
- *Arrêt du jeu et suppression du segment de mémoire partagée.*

Questions :

- ✔ Réalisez un schéma décrivant l'architecture de votre programme et mettant en évidence la zone de mémoire partagée (et les données que cette zone contient).
- ✔ Réalisez le programme en utilisant les librairies propres à la gestion de la mémoire partagée. Le programme traitera les différentes options présentées ci-dessus. Pensez à bien respecter les priorités et la décomposition du problème.
- ✔ Créez un script permettant de gérer une partie suivant le schéma présenté ci-dessus. Les deux joueurs seront gérés par un être humain au moyen d'un menu présent dans le script. Le script permettra aux joueurs d'interrompre une partie soit par le menu du jeu en tapant par exemple sur la touche « q », soit à l'aide d'un signal (par exemple par la prise en compte du signal envoyé par la combinaison « control+C ») : dans les deux cas, pensez-bien à supprimer le segment de mémoire partagée.

Conseils et limites :

- Nous vous rappelons que le morpion n'est qu'un prétexte pour appréhender la mémoire partagée. L'important n'est donc pas de réaliser le programme gérant parfaitement les règles. Il s'agit avant tout de réaliser un programme fonctionnel mettant en oeuvre des structures de données judicieusement choisies ainsi que des grands principes du jeu mettant à l'épreuve ces structures. Vous verrez dans un second temps les détails ...
- Nous ne mettrons pas en oeuvre dans ce TP les problèmes de concurrence d'accès à la zone de mémoire partagée. Ainsi, ce sera aux utilisateurs de veiller à ne pas compromettre la bonne marche du jeu.

5. Question(s) ouverte(s)

- En quelques lignes, préconisez les avantages et les inconvénients de cette approche (par rapport par exemple à l'utilisation d'une communication entre processus via le réseau) si elle est utilisée avec plusieurs processus (Exemple clients-serveur).
- On souhaiterait garantir que chaque message écrit soit lu avant qu'il puisse être écrasé par une nouvelle écriture dans le segment de mémoire partagée. Est-ce que l'exemple fourni respecte cette contrainte ? Pourquoi ? Si non, proposez une solution (sans la coder) permettant de respecter cette contrainte de sécurité et de concurrence d'accès à la ressource mémoire.
- Quel lien pouvez-vous faire entre l'utilisation d'un segment de mémoire partagée et le concept de segmentation ? Où placez vous le concept de pagination ? Sans trop entrer dans les détails, réalisez un schéma simplifié montrant la cohabitation entre la table des segments et la table des pages. (NB : Pensez bien à relire rapidement la documentation des fonctions de manipulation, vous y trouverez peut être des liens avec des mécanismes plus générique de gestion de la mémoire).
- et profitez en pour rappeler les notions fragmentation interne et externe ainsi l'importance du choix, par les concepteurs du système, d'une taille de page appropriée.

6. Annexes

- Utilisation des librairies externes (non nécessaire pour les fonctionnalités de base du programme) :
  - http://palantir.swarthmore.edu/maxwell/classes/tutorials/maketutor/
- Tutoriaux Makefile (si nécessaire) :
  - http://gl.developpez.com/tutoriel/outil/makefile/
  - http://bellet.info/creatis/unix/makefile.html
  - http://users.actcom.co.il/~choo/lupg/tutorials/writing-makefiles/writing-makefiles.html