info502 : Système d'exploitation
TP 3 : le morpion en mémoire partagée

Pierre Hyvernat, Sylvie Ramasso, Brice Videau
Pierre.Hyvernat@univ-savoie.fr
Sylvie.Ramasso@univ-savoie.fr
Brice.Videau@univ-savoie.fr

Pour ce TP, votre travail devra être envoyé par email à votre intervenant. Les consignes suivantes sont non-négociables :
- chaque fichier doit impérativement commencer par une entête comportant votre filière et les noms de tous les membres du groupes ;
- pour les parties "programmation", les réponses aux questions posées devront être dans le même fichier que votre programme. Utilisez des commentaires ;
- si vous avez une partie "compte-rendu" qui ne rentre dans aucun fichier source, vous pouvez joindre un fichier au format texte (pas de fichiers Word) ;
- faites des efforts pour le formatage de vos fichiers : indentez, allez à la ligne etc.

Finalement, si vous contactez un intervenant par email, mettez toujours un sujet commençant par "`[info502]`". Sans ça, vous courrez le risque que votre message ne soit pas lu...

But du TP

Tout le monde connaît le jeu du morpion ? Sinon, allez voir la page suivante pour vous rafraîchir la mémoire :

`http://fr.wikipedia.org/wiki/Morpion_(jeu)`

Nous allons essayer de programmer une interface pour jouer au morpion sur un ordinateur. L'architecture globale de notre programme sera la suivante :
- l'état du "plateau de jeu" sera conservé dans une partie partagée de la mémoire.
- un programme C s'occupera de la partie "bas niveau" : tout ce qui a trait à la gestion du bout de mémoire utilisé (initialisation, lecture et écriture de l'état, ménage etc.)
- un script shell se chargera de la partie "haut niveau" : gestion des joueurs (alternance des coups, etc.) et appellera le programme C.

La mémoire partagée

En C, il est possible de réserver un emplacement mémoire qui pourra être partagé entre plusieurs processus. Un `malloc` permet de réserver de la mémoire, mais seul le processus possède les droits de lecture ou écriture dans cet espace.

La fonction correspondante au `malloc` est la fonction `shmget` ("`sh`ared `m`emory `get`"). Cette fonction prend (entre autre) un argument `size` qui est la taille de la mémoire à allouer.

Pour pouvoir utiliser les fonctions sur la mémoire partagée, il faut utiliser les bibliothèques `sys/ipc` et `sys/shm`.

Lorsque deux programme veulent partager un morceau de mémoire, il faut que chacun des programmes soit sur qu'il utilise vraiment le même emplacement que son copain : la fonction `shmget` renvoie en fait un identificateur qui référence uniquement une telle zone de la mémoire. Cet identificateur est donné par l'allocateur de mémoire, et n'est donc pas calculable en avance.

C'est pourquoi on utilise en fait une *clé* pour parler des morceaux de mémoire partagée : si cette clé (définie à l'avance) est partagée par plusieurs programmes, ils pourront faire référence au même morceau de mémoire. Cette clé est utilisée comme premier argument pour la fonction `shmget` ; on l'obtient typiquement à partir d'un nom de fichier existant :

```
key = ftok("un_fichier", 'a') ;
```

La clé créée dépendra fontionnellement du nom du fichier et de la lettre donnée en second argument : deux programmes utilisant le même nom de fichier et le même caractère utiliseront donc la même clé.

Cette fonction se trouve dans la bibliothèque `sys/types`.

Le dernier argument de `shmget` est un entier qui spécifie les droits d'accès à la mémoire et le comportement de `shmget` en cas de problème. Reportez-vous à la page de manuel pour les détails.

Avec l'identifiant de zone mémoire, on peut ensuite "attacher" la zone (récupérer un pointeur vers la zone en question) ou la "détacher" (on prévient que l'on ne se sert plus de la zone). Les fonctions correspondantes sont `shmat` et `shmdt`. Une fois la mémoire attachée au bout d'un pointeur, on peut utiliser son contenu. (Mais attention, il faut que les programmes se débrouillent pour être d'accord sur le type des données mises en mémoire...)

Enfin, on peut également supprimer un morceau de mémoire partagée en utilisant la fonction `shmctl`. (Cette fonction permet de faire plein d'autres trucs !)

Un exemple commenté

Aller récupérer le programme `shmdemo.c` sur le site

`http://www.lama.univ-savoie.fr/~hyvernat`

(rubrique "Enseignement")

Compilez-le, exécutez-le et comprenez-le.

Utilisez les commandes unix `ipcs` et `ipcrm -m clé`. À quoi servent ces commandes ?

Écrivez un deuxième programme C qui supprime le segment de mémoire partagée. Vérifiez que votre programme fonctionne en utilisant la commande `ipcs`

Gestion des arguments en C

Lorsqu'un programme C est appelé avec des arguments, deux paramètres sont passés à la fonction `main` :

- le nombre de arguments (en général appelé `argc`), de type `int`,
- un tableau de chaînes de caractères, càd un `char**` (en général appelé `argv`).

La valeur de `argv[i]` est donc la chaîne contenant le `i`-ème argument. (Et `argv[0]` est le nom du programme.)

Le prototype de la fonction `main` devient donc

```
int main (int, char**)
```

Pour gérer les options* passées au programme, on peut soit tout faire à la main, soit utiliser la commande `getopt`, disponible dans la bibliothèque `unistd`.

Cette fonction permet de gérer les options assemblées, comme dans "`ls -Alsr /sbin/`", qui est équivalente à "`ls -A -l -s -r /sbin/`. Je vous invite à lire la page de manuel pour plus de détails : `man -S3 getopt`.

Remarque : l'utilisation de `getopt` n'est pas obligatoire, vous pouvez gérer les options "à la main".

Le morpion...

Le programme C : vous devez écrire un programme C qui, suivant ces arguments, aura le comportement suivant

- `-i` : création du segment de mémoire partagée et initialisation du plateau,
- `-a` : affichage du tableau stocké en mémoire partagée,

* un argument est une option s'il commence par "`-`". Par exemple, dans "`ls -A -l /sbin/`", "`-A`" et `-l`" sont des options, et "`/sbin/`" est un argument.

- `-x` i j : ajout d'une croix dans la case (i, j) du plateau,
- `-o` i j : ajout d'un rond dans la case (i, j) du plateau,
- `-e` i j : effacement de la case (i, j) du plateau,
- `-r` : réinitialisation du plateau, sans supprimer le segment de mémoire partagée,
- `-h` : affichage d'un message d'aide
- `-v` : vérification de l'état du jeu, en déterminant si un joueur a gagné... (facultatif)

Votre programme ne doit pas posséder de boucle d'interaction : on le relance pour chaque opération...

Remarques :

- n'essayez par de mettre la taille de votre plateau comme argument de votre programme (tous les plateaux auront la même taille)
- mais votre programme doit être paramétré par des constantes (on peut facilement changer la taille de plateau très facilement, mais il faut recompiler le programme)
- pour simplifier la gestion de la mémoire partagée, il est plus simple de stocker un tableau unidimensionnel. Pour accéder à la case `tab[i][j]`, il faudra donc faire `tab[i*LARGEUR+j]`

La boucle d'interaction en shell :

Écrivez un script shell qui permettra d'avoir une interaction aisée avec votre programme pour jouer :

- choix du nom des joueurs
- initialisation et affichage
- le premier joueur joue, affichage
- le deuxième joueur joue, affichage
- ...
- ménage

Faites en particulier attention à gérer correctement l'ordre des joueurs, et à ne jamais laisser de mémoire partagée allouée après votre script.