TD1-TME1 : Programmation Concurrente, généralités

Exercice 1 – Processus

Question 1
Expliquer les terminologies suivantes : système d'exploitation (ou *OS*), ordonnanceur (ou *scheduler*), concurrence des processus, préemption.

Question 2
Quelles sont les caractéristiques d'un bon ordonnanceur.

Question 3
Soient deux ensembles S et S' de processus définis ainsi :

$$S = [(x := x + 1; x := x + 1)||x := 2 * x],$$

$$S' = [(x := x + 1; x := x + 1)||(wait(x = 1); x := 2 * x)],$$

où un processus ne peut exécuter l'instruction $wait\ b$ que si l'expression booléenne b est vraie. On suppose que S et S' sont à mémoire commune (à l'instar des *threads*, voir Exercice 2). On suppose de plus que x est initialisée à 0 avant chaque exécution et que toutes les instructions de S et S' sont atomiques (c'est à dire non préemptible, mais on peut préempter entre deux actions atomiques).

Que peut valoir x après l'exécution de S en mode préemptif ? et après l'exécution de S' ? Que peut il se passer si les instructions ne sont pas atomiques ? Peut on avoir un scénario où x garde sa valeur 0 à la fin de l'exécution ?

Question 4 – Exemple d'ordonnancement
Soit un Processus $P_1 = a_1; a_2; ...a_N$ et un processus $P_2 = b_1; b_2; ...b_M$. Chaque a_I ou b_J est une action atomique. Décrire une exécution possible de $[P_1||P_2]$: en mode non-préemtif ; puis en mode préemptif. Donner toutes les exécutions possibles pour `N=2` et `M=2`, et le nombre d'exécutions possibles dans le cas général.

Question 5 – Ces x86 qui nous gouvernent

Soit le programme système suivant, tournant sur une architecture multi-coeur. `R V` lit (atomiquement) la variable `V`, `W V N` écrit (atomiquemet) l'entier `N` dans la variable `V` et || est l'execution parallele de deux sous-programmes :

```
(W X 1; R Y) || (W Y 1; R X)
```

1. En supposant que la valeur `0` est stockée dans les deux variables en début d'exécution, décrivez les résultats possibles des deux lectures (et les différentes manières d'y arriver), en utilisant la séquentialisation.
2. Sur les processeurs x86, environ une fois sur un million, l'execution de ce code produit le résultat `0` et `0` pour les deux lectures. Pourquoi ?
3. Comment analyser et vérifier des programmes tournant sur cette architecture ?

Question 6 – Etats de processus
Quels sont les états possibles d'un processus ? Donner les transitions possibles entre états puis le graphe de transition.

Exercice 2 – Threads

Question 1
Citez les différences principales entre *threads* et *processus*.

Question 2
Les *threads* sont aussi appelés *processus légers*, pourquoi une telle denomination ? Pourquoi a-t-on créé ces objets ?

Question 3 – Gestion des threads
Qui s'occupe de la gestion des threads ?

Question 4 – Contraintes d'ordonnancement
Les contraintes d'ordonnancement des threads sont elles les mêmes que pour les processus ?

Question 5 – Difficultés des Threads
Quelles sont les difficultés principales liées à l'utilisation des threads ?

Exercice 3 – Dîner des philosophes
L'histoire se passe dans un monastère reculé...

En bref, nous avons n moines assis autour d'une table ronde ayant chacun une assiette privée mais partageant des fourchettes. Il y a une et exactement une fourchette entre chaque pair de moines. Pour pouvoir manger proprement un moine doit posséder à la fois la fourchette à sa gauche et la fourchette à sa droite. La vie d'un moine est une boucle infinie : penser, manger, penser, manger, penser, et ainsi de suite. Si un moine ne mange pas pendant un certain temps il meurt de faim, entraînant bien de tristes conséquences pour tout le monastère. Peut on définir des points supplémentaires dans le règlement intérieur pour un fonctionnement paisible et studieux, sans accident regrettable ?

Question 1
Donner une solution dans le cas où n est pair.

Question 2
Dans le cas général, la proposition Chandy-Misra (1984) est basée sur le principe fondamental de la politesse et peut se résumer ainsi :

- A l'inauguration du monastère on salit toutes les fourchettes, installe les moines, et confit chaque fourchette au plus ancien des deux moines voisins.
- Un moine manquant une fourchette demande poliment à son voisin l'obtention de celle-ci. De même s'il lui manque les deux fourchettes (il demande alors à chacun de ses deux voisins).
- Un moine recevant une demande de fourchette examine la fourchette en question. S'elle est sale alors très poliment le moine nettoie et donne celle-ci au moine qui la demande. Sinon, il la garde toute de même.
- Un moine ayant deux fourchettes propres doit manger pendant un temps fini.
- Les fourchettes deviennent sales après chaque usage.

Est-ce suffisant ?

Exercice 4 – Implantation de processus et threads

Question 1
En utilisant l'appel système `pid_t fork(void);`, créer deux processus affichant la valeur de retour de l'appel à `fork`.

Question 2
Ecrire maintenant un programme possédant une variable entière `i`, créant un processus fils et tel que le père affiche un message indiquant qu'il est le père et demande à l'utlisateur de saisir une valeur pour `i` au clavier et tel que le fils commence par dormir 4 secondes, puis affiche le contenu de `i`.

Question 3
En utilisant la fonction `pid_t getpid(void);` (`pid_t` est compatible avec `int`), écrire le même programme avec des `threads` POSIX sauf que le père attend cette fois ci la fin du fils avant de terminer. Que constatez vous quant au contenu de la variable `i` dans le fils ? Qu'en deduisez vous sur le fonctionnment des threads Posix ? Quelle précautions sont à prendre pour éviter au programme précédant de produire un résultat indéfini ?

Exercice 5 – Implantation de processus et threads

Question 1
En utilisant l'appel système `pid_t fork(void);`, créer deux processus affichant la valeur de retour de l'appel à `fork`.

Question 2
Ecrire maintenant un programme possédant une variable entière `i`, créant un processus fils et tel que le père affiche un message indiquant qu'il est le père et demande à l'utlisateur de saisir une valeur pour `i` au clavier et tel que le fils commence par dormir 4 secondes, puis affiche le contenu de `i`.

Question 3
En utilisant la fonction `pid_t getpid(void);` (`pid_t` est compatible avec `int`), écrire le même programme avec des `threads` POSIX sauf que le père attend cette fois ci la fin du fils avant de terminer. Que constatez vous quant au contenu de la variable `i` dans le fils ? Qu'en deduisez vous sur le fonctionnment des threads Posix ? Quelle précautions sont à prendre pour éviter au programme précédant de produire un résultat indéfini ?

Exercice 6 – Compteur partagé

Question 1 – Architecture de base
Ecrire un programme utilisant l'API des `pthread` effectuant la tâche suivante :

Le programme principal initialise deux variables entières `temp` et `SHARED_compteur` à 0 puis lance un nombre `NB_THREAD` de pthread éxecutant la routine suivante :

- lire la valeur de la variable partagée dans `temp`
- rendre la main à l'ordonnanceur (`usleep, sched_yield`)
- incrémenter la valeur de `temp`
- rendre la main à l'ordonnanceur
- incrémenter la variable `SHARED_compteur`

Une fois les `pthreads` lancés, le programme principal attend tant que la valeur de `SHARED_compteur` soit `NB_THREAD` puis il affiche `"TERMINE"`

Dans un premier temps, ne pas synchroniser le programme.

Question 2 – Terminaison et jonction
Le programme principal utilise une boucle qui vérifie la valeur du compteur. Cela s'appelle une attente active qui occupe inutilement le processeur. Pour détecter la terminaison des threads, on peut utiliser le mécanismes de jonction. Ecrire une variante du programme avec jonction.

Remarque:
Les ressources allouées à un thread ne sont libérées que dans quelques cas :

— Le thread principale prend fin.
— On fait un `pthread_join`
— Le thread fini est dans un état `detached`, auquel cas son code de retour est inaccessible.

Le programme affiche-t-il tout le temps "TERMINE" ? Si non, donnez une exécution (avec `NB_THREAD=2`) qui n'affiche rien

Exercice 7 – Producteurs et Consommateurs

Le patron Producteur/Consommateur est un des patrons les plus courants dès lors qu'on utilise des threads. On le retrouve dans de très nombreuses applications :

— Jeux : les routines d'IA modifient l'aire de jeu et la routine d'affichage affiche le nouvel univers
— Sommation : Des routines auxiliaires calculent de éléments de la somme totale, et une routine regroupe les différents éléments
— ...

Question 1 – Architecture de base

La patron s'articule autour de deux ensembles :

— une équipe de producteurs qui produit des données
— une équipe de consommateurs qui les consomme.

Les deux équipes communiquent entre elles par une zone de transfert. Dans cet exercice on utilisera un `FIFO` de taille fixe comme Buffer de communication entre les équipes. Une implantation simple de cette structure de donnée se fait en utilisant un tableau de taille fixe et deux pointeurs dans ce tableau : l'un pour désigner le prochain endroit où on pourra consommer une donnée et l'autre pour désigner le prochain endroit où mettre une donnée.

Dans notre exercice, on enregistre un nombre `BUF_SIZE` maximum de données de type `int`. On écrira une fonction `Buffer* CreerBuffer()` pour créer un buffer.

On écrira en outre les fonctions auxiliaires :

— `void EcrireBuffer(Buffer *buf, int val)` pour ajouter un élément (afficher "P" sur la sortie d'erreur si le buffer est plein).
— `int LireBuffer(Buffer *buf)` pour lire le plus ancien élément (si disponible, afficher "V" sur la sortie d'erreur si le buffer est vide)

Ecrire les fonctions mentionées ci-dessus

Question 2

Ecrire deux fonctions qui seront exécutées respectivement par les threads producteurs et les threads consommateurs :

— Une fonction de prototype : `void * THREAD_Producteur(void * arg)`, implantantant la tache dévolue au producteur, i.e. produisant `NB_TURN` entiers à intervalle régulier (CF commande `usleep` pour la temporisation)
— Une fonctions de prototype : `void * THREAD_Consommateur(void * arg)`, implantant la tâche dévolue au consommateur, i.e. consommant des entiers à intervalle régulier (CF commande `usleep` pour la temporisation) et affichant les entiers consommés.

Ecrire cette premiere version sans synchronisation

En redirigeant la sortie d'erreur vers `/dev/null`, donnez un exemple d'affichage du programme.

Quels sont les problèmes posés par ce programme ?

Rappel : API POSIX Thread (Extrait)

Quelles sont les fonctionalités de base de l'API POSIX PThread
cf. Rappels Posix Threads

Creation de thread

```
int pthread_create(pthread_t  *  thread,
                   pthread_attr_t * attr,
                   void * (*start_routine)(void *),
                   void * arg);
```

Exemple :

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

void* THREAD_Routine(void *arg)
{    /* routine de thread*/
  return NULL;
}

int main(void)
{
  pthread_t thread_id;
  int ok;

  ok = pthread_create(&thread_id, NULL, THREAD_Routine, /* arg */ NULL);
  if(ok!=0)
    {
      fprintf(stderr,"Impossible de creer de le thread\n");
      exit(EXIT_FAILURE);
    }
}
```

Destruction de thread

```
void pthread_exit(void *retval);
// note: retval est disponible dans pthread_join
```

Identification

```
  pthread_t pthread_self(void);
  int pthread_equal(pthread_t thread1, pthread_t thread2);
```

Jonctions

```
int pthread_join(pthread_t th, void **thread_return);
```