

TD1 : 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

Y'a le bon et le mauvais ordonnanceur : "En fait le bon ordonnanceur, il monte sur scène, il a la RAM, et il cr...."
Que fait le 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 $P_1 = a_1; a_2; \dots a_N$ et un processus $P_2 = b_1; b_2; \dots b_M$. Chaque a_I ou b_J est une action atomique. Décrire une exécution possible de $[P_1 || P_2]$: en mode non-préemptif ; 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 – 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 dénomination ? Pourquoi a-t-on créé ces objets ?

Question 3 – Gestion des threads

Qui s'occupe de la gestion des threads ?

Question 4 – Contraintes d’ordonnement

Les contraintes d’ordonnement 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’utilisateur 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 fonctionnement des threads Posix ? Quelle précautions sont à prendre pour éviter au programme précédant de produire un résultat indéfini ?