

Systemes d'Exploitation

TP4 : *Ordonnancement et Mémoire partagée*

Jean-Christophe Deneuille
<jean-christophe.deneuille@xlim.fr>

7 avril 2016

Exercice 1 I need to borrow your boat

Le but de cet exercice est de créer un jeu de bataille navale dont le fonctionnement sera le suivant :

- a. 2 joueurs (tour à tour) représentés par 2 processus exécutés dans 2 terminaux
- b. Les tirs seront échangés dans un segment de mémoire partagée
- c. Chaque case de la grille de jeu est un caractère : '.' pour l'eau, 'B' pour un bateau non coulé, 'C' pour un bateau coulé, et 'R' pour un tir raté

La condition *a.* impose d'ores et déjà la création de trois sémaphores (non nommés) qui utiliserons le segment de mémoire partagée¹ : 1 pour que le joueur 1 bloque le joueur 2 pendant son temps de jeu, 1 pour faire de même dans l'autre sens, et 1 pour s'assurer qu'il n'y aura que 2 joueurs.

1. Avant toute chose, récupérez le code déjà écrit dans les TPs précédents (légèrement adapté pour ce contexte) [ici](#).
2. Créez donc 3 sémaphores nommés *S0*, *S1*, et *S2*, initialisés respectivement à 0 (le joueur 2 est bloqué), 1 (le joueur 1 a la main), et 2 (le nombre de joueur).
3. Écrivez le code de la partie initialisation du tableau de bateaux (cherchez le commentaire `/* QUESTION 1 */`).
4. Écrivez le code qui place aléatoirement des bateaux sur ce tableau (`/* QUESTION 2 */`).
5. Écrivez le code qui crée un segment de mémoire partagée et s'y attache (`/* QUESTION 3 */`).
6. Créez le groupe de 3 sémaphores (`/* QUESTION 4 */`).
7. Écrivez le code qui gère le premier tir (`/* QUESTION 5 */`).
8. Écrivez le code qui gère la boucle des tirs suivants (`/* QUESTION 6 */`).
9. Faites de même pour la réplique de l'adversaire (`/* QUESTION 7 */`).
10. Gérez l'affichage des résultats de manière "user-friendly" (`/* QUESTION 8 */`).

1. En effet d'après [man 7 sem_overview](#) : "Un sémaphore partagé entre processus doit être placé dans une région mémoire partagée (par exemple, un segment de mémoire partagée System V créé avec `semget(2)`, ou un objet mémoire partagée POSIX créé avec `shm_open(3)`)"

11. Complétez le code manquant en fin de boucle while (`/* QUESTION 9 */`).
12. Gérez la fin de partie : destruction sémaphores & libération mémoire (`/* QUESTION 10 */`).
13. Testez votre jeu :p

Exercice 2 Voir plus ?

Dans cet exercice, nous allons “jouer” avec un simulateur d’ordonnement des processus nommé **SIMOR** (pour **SIMulateur d’ORdonnement**). Commencez par récupérer le soft [ici](#). Pour le lancer : `java -jar Simor.jar`.

Vous pouvez définir dans la partie centrale en haut le nombre de processus que vous souhaitez, ainsi que le quantum pour les algorithmes de type tourniquet (RR et PRI, voir annexe pour un bref rappel ou le [cours sur l’ordonnement](#)).

En bas à gauche, vous pourrez définir les paramètres des différents processus créés : temps d’exécution et date d’arrivée. Nous n’utiliserons pas la partie en bas à droite (mode Statistiques).

Lorsqu’une simulation est effectuée, un résultat est présenté pour chaque politique d’ordonnement (cliquez sur l’onglet correspondant).

Reprenez l’exercice 1 du [TD3](#), et assurez-vous que l’enseignant ne vous a pas dit (trop) de bêtises. Attention pour la seconde question, les priorités dans Simor sont gérées de façon ~~anarchique~~ non standard : elles vont de 1 à 40, 40 correspondant à un processus ultra-prioritaire (alors que sous UNIX elles vont de -20 à 20, -20 étant la plus prioritaire). Vous inverserez donc les priorités :

- Prio(A) = 3 3
- Prio(B) = 5 1
- Prio(C) = 2 4
- Prio(D) = 1 5
- Prio(E) = 4 2

Annexe Exercice 2 Rappels sur les algorithmes d’ordonnement

FCFS (First Come First Serve), premier arrivé premier servi. Dans un système à ordonnancement non préemptif ou sans réquisition, le système d’exploitation choisit le prochain processus à exécuter, en général, le Premier Arrivé est le Premier Servi PAPS (ou FCFS First-Come First-Served) ou le plus court d’abord (**SJF** Short Job First). Il lui alloue le processeur jusqu’à ce qu’il se termine ou il se bloque (en attente d’un événement). Il n’y a pas de réquisition.

SJF (Short Job First), plus court d’abord. Si l’ordonneur fonctionne selon la stratégie SJF, il choisit parmi le lot de processus à exécuter, le plus court d’abord (plus petit temps d’exécution). Cette stratégie est bien adaptée au traitement par lots de processus dont les temps maximaux d’exécution sont connus ou fixés par les utilisateurs car elle offre un meilleur temps moyen de séjour. Le temps de séjour d’un processus (temps de rotation ou de virement) est l’intervalle de temps entre la soumission du processus et son achèvement.

SRT (Shortest Remaining Time), plus petit temps de séjour. L'ordonnancement du plus petit temps de séjour ou Shortest Remaining Time est la version préemptive de l'algorithme SJF. Un processus arrive dans la file de processus, l'ordonnanceur compare la valeur espérée pour ce processus contre la valeur du processus actuellement en exécution. Si le temps du nouveau processus est plus petit, il rentre en exécution immédiatement.

RR (Round Robin), algorithme circulaire. L'algorithme du tourniquet, circulaire ou round robin est un algorithme ancien, simple, fiable et très utilisé. Il mémorise dans une file du type FIFO (First In First Out) la liste des processus prêts, c'est à dire, en attente d'exécution. Il alloue le processeur au processus en tête de file, pendant un quantum de temps. Si le processus se bloque ou se termine avant la fin de son quantum, le processeur est immédiatement alloué à un autre processus (celui en tête de file). Si le processus ne se termine pas au bout de son quantum, son exécution est suspendue. Le processeur est alloué à un autre processus (celui en tête de file). Le processus suspendu est inséré en queue de file. Les processus qui arrivent ou qui passent de l'état bloqué à l'état prêt sont insérés en queue de file.

PRI (Priorités, sans évolution). L'ordonnanceur à priorité attribue à chaque processus une priorité. Le choix du processus à élire dépend des priorités des processus prêts. Les processus de même priorité sont regroupés dans une file du type FIFO. Il y a autant de files qu'il y a de niveaux de priorité. L'ordonnanceur choisit le processus le plus prioritaire qui se trouve en tête de file. En général, les processus de même priorité sont ordonnancés selon l'algorithme du tourniquet.

Linux (Priorités, avec évolution style Linux pour les processus qui ne sont pas temps réel). Pour empêcher les processus de priorité élevée de s'exécuter indéfiniment, l'ordonnanceur diminue régulièrement la priorité du processus en cours d'exécution. La priorité du processus en cours est comparée régulièrement à celle du processus prêt le plus prioritaire (en tête de file). Lorsqu'elle devient inférieure, la commutation a lieu. Dans ce cas, le processus suspendu est inséré en queue de file correspondant à sa nouvelle priorité. L'attribution et l'évolution des priorités dépendent des objectifs fixés et de beaucoup de paramètres.