

Rapport

Systèmes temps réel

FABRE Maxime – LEPOT Florian

3/26/2014

Tous les .c sont commentés de manière à expliquer le code dans ses moindres détails. Ce rapport ne contient que les questions autres que de programmation

Question 1

Voir code.

Question 2

Voir code.

Question 3

On crée deux sémaphores, l'un qui prendra la valeur du nombre de places du buffer, l'autre initialisé à 0. A chaque écriture (dans le thread équivalent qui tournera à l'infini) le sémaphore libre sera décrémenté signifiant qu'une place libre du buffer vient d'être écrasée, et le sémaphore occupé sera incrémenté. A chaque lecture ce sera l'inverse. Chaque place libérée prendra la valeur -1.

Question 4

Comme dit précédemment il faut 2 sémaphores. L'un à 0, l'autre prendra la valeur du nombre de places que contient le buffer.

Question 5

Voir code.

Question 6

Dans l'exercice numéro 4, on compte 2 types producteurs, les threads pour faire sortir un avion du hangar et le thread pour amener un avion pour l'atterrissage. On compte ensuite 2 types consommateurs, les threads pour faire atterrir et décoller un avion. Ce principe permet d'écrire dans le buffer que lorsqu'il y a de la place. De plus la lecture ne pourra pas lire une place du buffer où rien n'a été écrit dessus. Cela permet donc de résoudre les problèmes de lectures et d'écriture dans un même buffer par exemple.

L'exclusion mutuelle est désignée lorsque par exemple deux threads sont chargés de changer une même variable. Il faut alors qu'un des threads soit exclu grâce à un mutex.

Pour le type "producteur/consommateur" il faudra donc 2 sémaphores par buffer. Pour le type exclusion mutuelle il faudra un mutex.

Questions 7

A chaque décollage ou atterrissage certains sémaphores seront décrémenté ou incrémenté pour ainsi respecter le type producteur/consommateur.

Question 8

L'ordre des sémaphores est effectivement important. Selon lequel on applique `sem_post` ou `sem_wait` influence bien évidemment le fonctionnement des threads concernés. Un mauvais ordre des sémaphores peut entraîner une gestion des threads totalement corrompue.

Question 9

Si le nombre d'avions pouvant décoller ou atterrir est infini, le système ne pourrait alors pas fonctionner correctement avec n'importe quelle politique d'ordonnancement. En effet si l'alternance des threads n'est pas correctement appliquée, c'est-à-dire qu'un thread serait bien trop actif par rapport à un autre. Un nombre très important d'avions décolleront sans pour autant les faire atterrir. L'écart entre les avions décollant et atterrissant deviendrait alors contraignant. Il serait judicieux d'alterner les threads avec un mutex pour une meilleure gestion.

Question 10

Voir code.