ISTG - RICM2
Examen d'architecture de systèmes
Durée 2h

Tous documents autorisés – Le barême est indicatif

18 avril 2003

1 Implementation de synchronisation (12 pts)

Un noyau de gestion de thread sur monoprocesseur tel que celui étudié en TD/TP repose sur les opérateurs suivants :

Exclusion mutuelle Elle est assurée par un simple mécanisme de contrôle des interruptions : `MASK() et UMASK()`

Gestion de file il est possible d'insérer/retirer un descripteur de thread `TCB` indifférement en tête ou en queue d'une `Queue` .

Temps partagé Les threads sont gérés en temps partagé **sans priorité**. Le thread courant est repéré par le pointeur global `TCB* current`. Les threads prêts sont dans la file `readyQ`.

Suspension-réveil Le thread courant peut abandonner le processeur en se suspendant dans une file par `Suspend( Queue *)`. Un thread peut insérer un autre thread dans la file des prêts par `Wakeup(TCB*)`.

On se propose d'étudier l'implantation d'une synchronisation du type moniteur c'est à dire des 4 opérateurs suivants :

Exclusion mutuelle Un verrou `Mutex` permet de garantir l'exécution en exclusion mutuelle d'une portion de code. `Lock(Mutex*)` bloque le thread courant si le verrou est pris. `Unlock(Mutex*)` libère le verrou.

Blocage Un thread détectant une condition de blocage à partir des variables exclusives se met en attente sur une `Cond` en libérant le verrou par `Wait( Cond *, Mutex*)`.

Réveil Un thread détectant une condition de réveil à partir des variables exclusives peut réveiller UN thread bloqué sur la condition correspondante par `Signal(Cond*)`.

L'implémentation du moniteur dans un système temps partagé sans priorité pose le problème du conflit d'attribution du droit d'accès exclusif :

Conflit signaleur/reveillé Lors d'un `signal()`, le signaleur garde-t-il la section critique ou la cède-t-il au réveillé ?

Conflit arrivant/résident Il y a aussi conflit entre les threads essayant de franchir un `Lock()` et les threads qui doivent y reprendre leur exécution[1].

Décrire en pseudo C l'implantation des trois statégies suivantes :

1.1 Sans prorité (4 pts)

Dans ce cas, le signaleur garde la section critique. Pas de priorité entre arrivants/résidents.

[1] Selon la règle de résolution du conflit signaleur-reveillé, ce sont l'un ou l'autre type de thread.

1.2 Priorité signaleur(4 pts)

Dans ce cas, le signaleur garde la section critique. Priorité des résidents par rapport aux arrivants.

1.3 Priorité réveillé (4 pts)

Dans ce cas, le signaleur cède la section critique au réveillé. Priorité des résidents par rapport aux arrivants.

2 Serveur concurrent : ordonnancement des threads (8 pts)

Un serveur concurrent est à l'écoute du réseau et crée un thread par requête. On se pose le problème de la relation entre l'ordonnancement des threads et débit/temps de réponse du serveur.

Pour simplifier, on suppose un régime de fonctionnement où les requêtes arrivent et sont traitées par groupe de N. Il y a donc N threads en exécution pour traiter un groupe de requêtes. Les requêtes sont indépendantes et de durée quelconque.

2.1 Stratégie FIFO (2 pts)

Les threads sont exécutés dans l'ordre FIFO[2]. Soit t0, la date d'arrivée d'un groupe de requêtes. A quelle date la requête de rang `i` de durée T_i sera-t-elle terminée ? Au bout de combien de temps le groupe de requêtes est il terminé ?

2.2 Stratégie RR ou tourniquet (2 pts)

Les threads sont exécutés en temps partagé. Soit t0, l'arrivée d'un groupe de requêtes. A quelle date la requête la plus courte de durée T_{min} sera-t-elle terminée sous l"hypothèse d'un quantum $q < T_j \forall j$ et d'un surcoût de δ à chaque commutation de temps partagé. A quelle date la requête la plus longue de durée T_{max} sera-t-elle terminée? Comment faudrait il fixer le quantum q? Pourquoi ?

2.3 Débit/ Temps de réponse (2 pts)

Quels sont les débits[3]pour FIFO et RR. Que pouvez vous dire du temps de réponse?

2.4 Stratégie du tourniquet multiniveau (2 pts)

A la réception d'une requête d'un client, on ne peut prédire son temps d'exécution mais une analyse statistique des requêtes indique que sur des paquets de N requêtes, 75% des requêtes durent un temps $0 < t <= T0$, 20% prennent un temps $T0 < t <= T1$, 4% prennent un temps $T1 < t <= T2$ et les requêtes restantes prennent un temps fini quelconque $> T2$. On peut exploiter ces propriétés temporelles pour définir une gestion en temps partagé optimisant débit et temps de réponse.

temps partagé Quelle modification doit être faite au temps partagé? quelles sont les différents quantums à manipuler ? et comment ?

Amélioration Pourquoi temps de réponse et débit sont-ils améliorés ?

[2] Le premier du groupe puis le second etc..
[3] en groupes de N requêtes par seconde