Examen 2006/2007 Temps Réel

Le 12/6/2007, Tout document autorisé
Durée : 1 heure et demie

S'il vous plaît, pour faciliter la correction, essayez de répondre aux questions dans l'ordre. Il faut étayer vos réponses sinon elles ne rapportent pas de points.

1. Dans quels cas un verrou est nécessaire ?
2. Quels problèmes posent les verrous quand on fait du temps réel ?
3. Comment calculer des capacités quand on utilise des verrous ?
4. Soit une liste chaînée utilisée par plusieurs processus.
   (a) Proposez un algorithme permettant d'implanter la destruction d'un élément de la liste **sans utiliser de verrous**.
   (b) Quels sont les inconvénients de votre algorithme ?
   (c) Comment les éliminer ?
5. Les ordonnanceurs vus en cours peuvent-ils ordonnancer autant de processus que l'on veut ?
6. Comment calculer une capacité théoriquement ?
7. Comment calculer une capacité expérimentalement ?
8. Dans quels cas est-il impossible de calculer la capacité ?
9. La notion de *quantum* de temps est-elle nécessaire à un ordonnanceur temps réel ?
10. Soit un système faisant tourner 2 tâches A et B de période 20 et de capacité 10 (unités de temps).
    (a) Sont-elles ordonnançables en utilisant le critère de la RM ?
    (b) Sont-elles ordonnançables en utilisant le critère de la section critique ?
    (c) Si les deux tâches rendent chacune un service, quel est le temps moyen mis pour rendre le service ?
    (d) Est-ce que les paramètres de l'ordonnanceur peuvent agir sur ce temps moyen ?
11. Quels sont les avantages et inconvénients de l'utilisation d'un MMU (*Memory Management Unit*) dans un système temps réel ?
12. Une tâche temps réel peut-elle être écrite dans un langage interprété (Java, Python, Prolog, ...) ?
13. Comment un système temps réel peut-il faire pour traiter des interruptions venant de périphériques tout en garantissant le respect des échéances pour toutes les tâches ?
14. Quelle échéance donneriez-vous au traitement des événements dans une application graphique interactive ?

Une correction :

1. Dans quels cas un verrou est nécessaire ?

   Quand plusieurs processus veulent accéder en lecture et écriture à une même donnée en mémoire.

2. Quels problèmes posent les verrous quand on fait du temps réel ?

   L'incertitude sur le temps d'acquisition du verrou et l'inversion de priorité.

3. Comment calculer des capacités quand on utilise des verrous ?

   On ajoute à la capacité normale le temps d'acquisition du verrou dans le pire cas en tenant compte de l'ensemble des tâches. Il ne faut pas oublier que l'on peut avoir des chaînes complexes (A bloque B qui bloque C qui bloque...). Pour simplifier ce calcul, on peut utiliser une méthode comme le plafond de priorité qui simplifie le calcul de capacité.

4. Soit une liste chaînée utilisée par plusieurs processus.

   (a) Proposez un algorithme permettant d'implanter la destruction d'un élément de la liste **sans utiliser de verrous**.

   (b) Quels sont les inconvénients de votre algorithme ?

   (c) Comment les éliminer ?

   Ceci est la solution la plus simple que j'ai trouvé, Elle n'a pas besoin d'être programmée en assembleur avec blocage du bus de donnée comme c'est le cas si l'on utilise une instruction atomique d'échange de 2 pointeurs en mémoire. Elle n'a pas besoin d'une tâche rendant le service destruction qui serait longue à appeler.

   On ajoute un champs `détruit` *dans les cellules de la liste chaîné. Quand on veut détruire l'élément, on passe ce champ à* `vrai`*. Attention, ce champ doit être modifiable atomiquement, c'est-à-dire que c'est un mot du processeur (un* int*) et non un bit dans un entier.*

   L'inconvénient est que la place mémoire n'est pas libérée.

   Une manière de le corriger (toujours sans verrou) est de laisser la capacité de vraiment détruire les éléments de la liste à un processus unique qui sera réveillé quand la liste contiendra trop d'éléments en instance de destruction.

5. Les ordonnanceurs vus en cours peuvent-ils ordonnancer autant de processus que l'on veut ?

   On suppose évidemment que l'ensemble des tâches respectent le test de la section critique, sinon la réponse est trop facile.

   Si vous avez un million de tâches, votre ordonnanceur va utiliser un part non négligeable de temps CPU, l'utilisation de ce temps pourra rendre l'ensemble des tâches non ordonnançable. Il faut donc tenir compte de ce problème.

   Sauf dans le cas de l'ordonnanceur cyclique qui n'utilise pas de temps CPU pour s'exécuter.

6. Comment calculer une capacité théoriquement ?

   On trouve le pire cas pour l'algorithme, on affecte les durées maximum aux différentes instructions, on affecte un temps limite aux entrées/sorties, on suppose ne pouvoir acquérir aucun verrou sans attendre.

7. Comment calculer une capacité expérimentalement ?

   On désactive les caches du processeur, on s'arrange pour que toutes les autres tâches aient verrouillées les sections critiques, on donne les pires données à traiter (celles déclenchant le pire cas), on mesure le temps d'exécution.

   On prend bien sur le pire temps d'exécution en lui ajoutant par exemple 10 fois l'écart-type des mesures que l'on à faites.

8. Dans quels cas est-il impossible de calculer la capacité ?

   Quand il y a des entrées sorties ou bien un calcul non déterministe.

9. La notion de *quantum* de temps est-elle nécessaire à un ordonnanceur temps réel ?

   Ne pas confondre quantum *de temps et unité de temps. Le* quantum *de temps est le temps maximum que l'on laisse à un processus avant de donner la main à un autre. Il permet de faire comme si plusieurs processus fonctionnaient simultanément.*

   L'ordonnanceur cyclique n'utilise pas de quantum *de temps, donc la réponse à cette question est évidemment 'non'.*

   HPF, EDF et LLF peuvent l'utiliser en cas d'égalité, mais cela fait perdre du temps CPU à cause des changements de contexte.

10. Soit un système faisant tourner 2 tâches A et B de période 20 et de capacité 10 (unités de temps).

    (a) Sont-elles ordonnançables en utilisant le critère de la RM ?

    Le taux d'utilisation de la machine est de 1, il est plus grand que 0.87, on ne peut rien dire ! **Je rappelle que si $A \Rightarrow B$ cela ne veut pas dire que $\overline{A} \Rightarrow \overline{B}$**

    (b) Sont-elles ordonnançables en utilisant le critère de la section critique ?

    Si l'on applique la formule ou bien si l'on dessine l'ordonnancement on voit que l'on est dans un cas limite car le CPU est utilisé à 100%

    (c) Si les deux tâches rendent chacune un service, quel est le temps moyen mis pour rendre le service ?

    Supposons que le temps d'exécution pris par la tâche seule soit égale à la capacité.

    Si les deux tâches sont lancées l'une après l'autre avec un décalage de 10 unité de temps de la périodicité, alors le temps moyen pour rendre le service est de 10 unité de temps.

    Si les deux tâches commencent en même temps (synchronisée). Alors si A est plus prioritaire que B, le temps moyen pour A sera de 10 et pour B de 20, le temps total moyen sera donc de 15.

    Si les deux tâches s'exécutent en parallèle, alors chacune des 2 tâches mettra 20 unités de temps pour s'exécuter.

    (d) Est-ce que les paramètres de l'ordonnanceur peuvent agir sur ce temps moyen ?

    Il ne faut pas utiliser de quantum *de temps pour ne pas tomber dans le pire cas pour le temps moyen.*

    Moralité : Le temps partagé est néfaste pour le temps de réponse moyen.

11. Quels sont les avantages et inconvénients de l'utilisation d'un MMU (*Memory Management Unit*) dans un système temps réel ?

    En temps réel les applications sont en mémoire centrale, leur mémoire virtuelle n'est pas swappé *sur le disque. Le MMU ne lance donc "jamais" d'interruption.*

L'avantage de l'utilisation d'un MMU est que la mémoire est protégée entre les tâches, c'est donc **plus fiable***. Si une tâche fait n'importe quoi, elle ne pourra perturber les autres en les écrasant. Par contre le MMU déclenchera une interruption pour prévenir l'ordonnanceur que la tâche à fait n'importe quoi.*

L'inconvénient est que les **changements de contexte sont plus longs** *car l'ordonnanceur doit reprogrammer le MMU à chaque fois.*

12. Une tâche temps réel peut-elle être écrite dans un langage interprété (Java, Python, Prolog, ...) ?

    La seule condition pour qu'une tâche soit temps réel et que l'on soit capable de calculer sa capacité.

    Dans le cas des langages interprétés le seul réel problème est la gestion mémoire et le garbage collector *qui peut se déclencher à n'importe quel moment.*

    Ce problème peut être résolu par l'application qui peut gérer elle même sa mémoire ou bien en utilisant un langage interprété adapté qui fera le ménage continuellement.

    Les problèmes dus à des bugs ne sont pas spécifiquement lié aux langages interprétés, ils peuvent aussi ce produire avec des langages compilés.

    Autre avantage : il n'y a pas besoin d'utiliser de MMU pour protéger des tâches interprétées les unes des autres.

13. Comment un système temps réel peut-il faire pour traiter des interruptions venant de périphériques tout en garantissant le respect des échéances pour toutes les tâches ?

    Le gestionnaire d'interruption se contente de libérer un verrou pour débloquer la tâche temps réel qui fera vraiment le travail de gestion de l'interruption.

    Cette tâche sera gérée comme un tâche sporadique : on remplira son quantum *de temps au bon moment. L'interruption ne sera pas gérée si elle se produit trop souvent.*

14. Quelle échéance donneriez-vous au traitement des événements dans une application graphique interactive ?

    On veut évidemment donner l'échéance la plus longue possible pour que l'application graphique ne perturbe pas les autres applications temps réel.

    Quand l'utilisateur fait une action, on veut que l'application réponde instantanément visuellement. L'échéance est donc celle classiquement utilisée au début du cinéma : 1/24 de secondes.

    Je rappelle que la fréquence de la télé est à 25Hz (ou 30Hz aux US) et que cela vient de la fréquence du courant électrique.