Examen 2008/2009 Temps Réel

Le 9/6/2009, Tout document autorisé
Durée : 1 heure et demie

1. Citez un algorithme de tri adapté au temps réel et un qui ne l'est pas.
   Expliquez pourquoi.
2. Que garantit un traitement temps réel ?
3. Peut-on faire un processus temps réel fonctionnant dans un système d'exploitation qui n'est pas temps réel ?
   Détaillez votre réponse.
4. Un système temps réel fait fonctionner des tâches non périodiques.
   Comment garantir que l'ensemble de ces taches pourra fonctionner en temps réel ?
5. Avec les ordonnanceurs (HPF, EDF, LLF) indiquez comment faire pour que les tâches périodiques démarrent au début de leur période.
   Est-ce possible tout le temps ?
6. Quand on lance une tâche dans un ordonnanceur temps réel (HPF, EDF, LLF), celui-ci peut-il nous garantir que cette tâche s'exécutera en temps réel ? Pourquoi ?
7. Comment calculer la capacité d'une tâche ?
8. Soit une liaison réseaux bipoint *fullduplex* directe entre 2 machines.
   Des processus temps réel périodiques fonctionnent sur les deux machines et utilisent la liaison pour communiquer.
   Quels sont les problèmes ?
   Comment procéderiez-vous pour vous assurer que tous les processus soient temps réel ?
   Que doit prendre en compte le système d'exploitation pour que cela soit réalisable ?
9. On veut faire un serveur de base de donnée "temps réel".
   Qu'est-ce que cela veut dire ?
   Quels sont les problèmes ?
   Comment les résoudre ?
10. Les logiciels de traitement de texte sont *normalement* temps réel.
    Qu'est-ce que cela veut dire ?
    Quels sont les problèmes ?
    Comment les résoudre ?

Une correction

1. Citez un algorithme de tri adapté au temps réel et un qui ne l'est pas.
   Expliquez pourquoi.

   Ce qui est adapté : les algorithmes dont le pire cas est on $O(n\ logn)$: *heap sort*, *merge sort*

   Inadapté : les autres qui sont en $O(n^2)$: *bubble sort*, *quick sort*. Même en mélangeant les données du *quick sort* il est possible (très improbable) que l'on tombe dans le pire.

   Remarque : tous les algorithmes sont déterministes, on peut toujours calculer le pire cas. À l'exception des algorithmes utilisant des valeurs aléatoires pour fonctionner (Monte Carlo par exemple).

Remarque : Une implémentation de ces algorithmes interruptible à n'importe quel moment et laissant une table partiellement triée peut-être intéressante dans le cas du temps réel. L'implémentation doit travailler directement dans la table initiale et ne pas utiliser de structures de données externes.

2. Que garantit un traitement temps réel ?

   Il garanti que si l'ordonnancement temps réel est possible alors **les tâches respecteront leur échéance**.

   Un "vrai" système temps réel n'autorise une tâche à être lancée que s'il peut garantir son fonctionnement temps réel.

   Le système temps réel ne peut garantir qu'une tâche ne dépassera pas sa capacité. Si une tâche dépasse sa capacité c'est que le programmeur a mal fait son travail et dans ce cas le temps réel ne peut être assuré par le système.

3. Peut-on faire un processus temps réel fonctionnant dans un système d'exploitation qui n'est pas temps réel ?
   Détaillez votre réponse.

   Premier problème : le processus temps réel ne doit pas être interrompu par le système, il doit donc arrêter les interruption matérielles (dangereux). Les systèmes d'exploitation ne permettent pas cela à des processus utilisateurs, la tâche doit obligatoirement être dans le système.

   Deuxième problème : latence de démarrage. Le processus doit être réveillé par une interruption matérielle afin de garantir un démarrage sans latence. Les systèmes d'exploitation ne permettent générallement pas cela à des processus utilisateurs.

   Troisième problème : tout s'écroule si le système d'exploitation suspend les interruptions car la tâche temps réel ne peut démarrer.

   En conséquence, il n'est pas possible d'avoir une tâche temps réel dans un système non temps réel.

4. Un système temps réel fait fonctionner des tâches non périodiques.
   Comment garantir que l'ensemble de ces taches pourra fonctionner en temps réel ?

   Premier problème : on garanti le fonctionnement temps réel des tâches périodiques en regroupant les tâches apériodiques dans une tâche périodique chargée de les gérer.

   Deuxième problème : garantir le fonctionnement temps réel des tâches apériodiques. Ceci n'est possible que si l'on peut fixer un intervalle de temps minimum entre 2 déclenchements. Cela revient à dire que l'on peut associer une période minimum à la tâche apériodique. Dans ce cas, on peut garantir (ou non si c'est impossible) le fonctionnement temps réel des tâches apériodiques.

5. Avec les ordonnanceurs (HPF, EDF, LLF) indiquez comment faire pour que les tâches périodiques démarrent au début de leur période.
   Est-ce possible tout le temps ?

   Remarque : un système temps réel garanti le respect des échéances, pas le moment du démarrage.

   Remarque : il est impossible de faire en sorte que deux tâches démarrent simultanément au même moment. Ce qui suit n'est donc valable que si l'on a qu'une seule tâche que l'on veut faire démarrer au début de sa période.

   HPF : On donne à la tâche la priorité la plus grande. (Le test du *rate monotonic* n'est plus valable).

EDF : On donne à la tâche une échéance égale à 0. Comme cette échance est plus petite que la capacité les tests permettant de vérifier que le système est temps réel échoueront tous. C'est visiblement pas un bonne idée.

LLF : On donne à la tâche une échéance égale à sa capacité. L'ordonnanceur la fera donc démarrer immédiatement.

6. Quand on lance une tâche dans un ordonnanceur temps réel (HPF, EDF, LLF), celui-ci peut-il nous garantir que cette tâche s'exécutera en temps réel ? Pourquoi ?

   HPF et EDF : non, car ne connaissant pas la capacité, ils ne peuvent donc pas évaluer le critère *rate monotonic* pour savoir s'il y a un problème.

   LLF : oui, car lors de la création de la tâche on fournit la capacité et l'échéance à l'ordonnanceur. *Évidemment l'utilisateur doit calculer correctement la capacité.*

7. Comment calculer la capacité d'une tâche ?

   On détermine les données en entrée qui maximisent la durée des traitements.

   On évalue la durée du traitement en supposant le pire cas : rien dans le cache, délai maximum pour l'acquisition des verrous, durée maximum pour les entrées sorties.

8. Soit une liaison réseaux bipoint *fullduplex* directe entre 2 machines.
   Des processus temps réel périodiques fonctionnent sur les deux machines et utilisent la liaison pour communiquer.
   Quels sont les problèmes ?
   Comment procéderiez-vous pour vous assurer que tous les processus soient temps réel ?
   Que doit prendre en compte le système d'exploitation pour que cela soit réalisable ?

   On suppose évidemment que seuls les processus temps réels peuvent utiliser cette liaison.

   On supposera la liaison sans erreur, s'il y a des erreurs, on ne les corrigera pas, on préviendra le processus qui lit le paquet que les données sont erronées.

   Dans les tâches, on programmera l'envoit et la réceptions d'information de manière asynchrone. Si l'on pose une question, la réponse sera connu dans une future période (définissable à l'avance en calculant bien les périodes et échéances).

   Ceci dit, on s'apperçoit que l'on peut calculer pour chaque tâche une quantité maximale de données à transférer de manière périodique. Et donc un temps maximal d'utilisation de la liaison. Mais aussi une échéance si celle-ci a besoin d'être plus petite que la période.

   Avec toutes ces informations, on peut évaluer un critère comme la *zone critique* par exemple pour savoir si l'envoi des données se fera en respectant toutes les échéances.

   Ce qui précède est faux, en effet :

   - quand on a commencé l'envoit d'un paquet sur la liaison on ne peut l'interrompre pour envoyer un paquet plus prioritaire. Il faut donc trouver un autre critère (non vu en cours) pour savoir si la liaison est temps réel.
   - le moment où est lancé l'envoit de données n'est pas périodique, la tâche a pu être lancée en retard ou les calculs fait avant l'envoit des donnée n'ont pas pris le même temps. Là aussi, le critère classique n'est pas suffisant pour garantir le temps réel.

9. On veut faire un serveur de base de donnée "temps réel".
   Qu'est-ce que cela veut dire ?
   Quels sont les problèmes ?
   Comment les résoudre ?

   Temps réel implique tâche périodique et critères pour décider si le système complet est temps réel ou non.

Les critères du cours suppose que l'ordonnanceur est HPF et que les tâches (requêtes) sont périodiques.

Moralité : la base de donnée doit connaître la priorité des requêtes qu'elle reçoit. Elle doit immédiatement suspendre les traitements si une requête plus prioritaire arrive. Elle doit aussi posséder un algorithme évitant les problème d'inversion de priorité sur les sections critiques.

Du coté utilisateur, celui-ci doit être capable de calculer la capacité de la requête. La capacité étant une constante cela implique qu'elle est calculée pour une certaine taille de la base de données et que celle-ci ne doit pas grossir dans le temps.

Pour être **certain** que ne pas sous évaluer la capacité, on peut tester pratiquement la base de donnée en lançant en grandeur nature le test de la zone critique sur une base de donnée qui vient d'être lancée (sans cache disque). On lancera simultanément toutes les requêtes périodiques. La capacité sera largement sur évaluée mais on est certain que rien de pire ne pourra arrivé dans la réalité.

10. Les logiciels de traitement de texte sont *normalement* temps réel.
Qu'est-ce que cela veut dire ?
Quels sont les problèmes ?
Comment les résoudre ?

L'intéraction utilisateur déclenche un traitement pouvant duré plusieurs secondes pour un gros texte.

Problème 1 : si une deuxième interaction arrive, elle doit interrompre le premier traitement (qui la plupart du temps est obsolete). Ceci est fondamental et l'application n'est pas utilisable sans ceci.

Problème 1.1 : si deux interactions arrivent simultanément, on les fusionne pour ne lancer qu'une seule traitement.

Problème 1.2 : si le flux d'interaction est constant et qu'aucun traitement n'a le temps de se terminer. Alors peut être utile de ne plus répondre aux interactions utilisateur afin de mettre à jour l'écran afin que l'utilisateur ne croit pas que la machine soit planté. Par exemple au bout d'une demi seconde d'écran figé.

Problème 2 : il faut mettre à jour en premier ce que l'utilisateur voit pour lui faire croire que le logiciel tout fini. Par exemple, le mot puis la ligne puis la page courante et enfin le reste du texte.

Problème 3 : On veut pouvoir aller instantanément à n'importe quelle page il faut que la pagination complète soit mémorisée. (Attention aux artefacts : on fait une modification sur la première page qui influe sur tous le texte puis en demande à voir la millième. La page sera affichée tout de suite mais elle se mettra à jour toute seule quelques secondes plus tard.)

Problème 4 : La sauvegarde ne pouvant être instantanée, il faut faire une copie des données en mémoire et sauvegarder en arrière plan. Des solutions plus performantes existent (*copy on write* ou gestion de *snapshot*), elles permettent d'avoir des versions différentes d'une même donnée. Elles sont complexes à implémenter.

Problème 5 : La lecture. On doit pouvoir instantanément aller à la page que l'on veut sur le document qui vient d'être ouvert. Le format de sauvegarde doit donc contenir un index des pages permettant de lire en priorité la bonne page ainsi que les données de formatage pour cette page. Si cette page nécessite des données suplémentaires ; par exemple une feuille de style, seules les données utiles au rendu seront chargées pas le style complet. (C'est comme cela que les processus fonctionnent sous linux, seules les pages utiles dans les librairies sont chargées pendant l'exécution. au départ, le processus ne contient qu'une seule page mémoire)