Examen 2003/2004 Temps Réel

Le 10/06/2004, Tout document autorisé
Durée : une heure

Recopier tel quel des extraits de documents n'apporte jamais de points. Vous devez répondre aux questions qui vous sont posées. Vous devez argumenter (étayer) vos réponses. Une réponse correcte non étayée ne rapporte pas de points, et une réponse incorrecte bien étayée peut en rapporter.

Description du problème

Un ordinateur contient une carte audio. Cette carte audio n'est utilisée que pour générer des sons (elle ne fait pas d'entrée ou de numérisation de son). Cette carte contient 1000 octets de mémoire tampon (*buffer*). Cette mémoire tampon est utilisée comme une FIFO. La FIFO est remplie par l'ordinateur et vidée par la carte. Tant qu'elle n'est pas vide la carte génère le son à partir des données contenues dans la FIFO. Lorsque la FIFO est pleine, la carte met $\frac{1}{8}$ de seconde pour la vider. La carte peut envoyer une interruption pour prévenir quand le *buffer* est vide.

Un gestionnaire (*driver*) simple pour la carte son

Ce *driver* est dans le système d'exploitation, il permet de voir la carte son comme un fichier dans lequel on peut écrire. Lorsqu'il reçoit des données il les écrit dans la FIFO de la carte son. S'il y a trop de données à envoyer, il attend que la FIFO soit vide.

Questions :

1. Quand un processus écrit dans le fichier “audio”, est-il bloqué ?
2. Un processus doit “jouer” un son contenu dans un fichier. Proposez un algorithme pour ce processus.
3. Quels sont les problèmes sonores qui peuvent apparaître ?
4. Proposez des méthodes pour éliminer ces problèmes.
5. Vos méthodes ont certainement des désavantages. Lesquels ?
6. Pourquoi ces problèmes n'apparaissent pas sur les chaînes HiFi ?
7. Est-ce qu'il y a plus de problèmes si la FIFO fait 10 octets ?
8. Quel-est donc la taille de FIFO critique ?
9. Et si la FIFO fait 10000 octets, quel problème va apparaître ?
10. Votre processus est maintenant un jeu interactif qui génère des sons en fonction de ce qui se passe sur l'écran. Proposez un algorithme ou bien faites des remarques générales.
11. Le *driver* ne permet *généralement* pas d'ouvrir le fichier “audio” plus d'une fois. Hors il est intéressant que plusieurs applications puissent générer des sons simultanément. Ce problème peut être résolu en lançant un serveur (processus) audio dans l'espace utilisateur. Le serveur audio est le seul à ouvrir le fichier “audio” les applications s'adressent au serveur pour générer les sons. Le serveur mélange (mixe) les sons avant de les envoyer dans le fichier audio. Quelles tailles de FIFO faut-il utiliser en interne dans ce serveur ?
12. Le serveur audio doit-il être un processus réveillé périodiquement ?
13. Comment faire en sorte pour que le passage par ce processus intermédiaire améliore la qualité (diminue le nombre de défaut) du son plutôt qu'il ne la baisse ?

Une correction

1. Le processus est suspendue quand le buffer de la carte son est plein et que le *driver* n'a pas fini d'envoyer les données. (Sauf si on demande explicitement d'utiliser le fichier de manière asynchrone).

2. Version simple : Tant que l'on peut on lit un bloc d'information du fichier contenant le son et on le stocke dans le fichier représentant le périphérique son.

3. Problème du au *driver* : si celui-ci suspend le processus alors qu'il ne reste qu'un octet à écrire alors quand le processus redémarre il n'a pas le temps d'envoyer la suite du son en moins de 1/8000 de seconde. Il y a une coupure.

   Problème du au processus : La lecture du bloc d'information peut prendre un certain temps (accès disque par exemple). Il peut y avoir une coupure.

4. Pour le *driver* : si possible on rerempli le buffer avant qu'il ne doit vide (si la carte son préviens quand le buffer est à moitié vide. Sinon, le processus d'envoi de son peu devenir périodique et par exemple envoyer 4000 octets tous les 1/16 de seconde. Ainsi le buffer ne sera jamais plein ni vide (sauf gros pépin).

   Dans le cas du processus : on peut lire le fichier complet en mémoire et verrouiller la mémoire pour qu'elle ne soit pas *swappée*. On aura pas de problème d'accès disque/réseau.

5. Modifier le *driver* est complexe. Verrouiller la mémoire consomme trop de ressources. Donc il faut travailler périodiquement en prévoyant un tampon intermédiaire (16Ko?) verrouillé en mémoire.

6. Par ce que il n'y a qu'un seul processus. Mais en fait cela n'est pas totalement vrai, en effet les lecteurs de CD doivent gérer les corrections d'erreurs sur le médium. D'autre part, on peut imaginer que le lecteur essaye de relire plusieurs fois des passages abîmés sur le disque. Donc, il peut y avoir des cas ou cela plante un peu...

7. Avec une FIFO de 10 octets, le driver va recevoir de nombreuses interruptions qui consommeront des ressources. De plus on ne pourra pas avoir un processeur utilisateur périodique car la période sera trop courte pour un système normal. Par contre cela serait possible sur un système temps réel (en consommant beaucoup de ressources).

8. La taille critique est quand le temps pour vider la FIFO est inférieur au cycle de base de l'ordonnanceur. Sur un système UNIX, c'est $\frac{1}{100}s$ donc la FIFO ne doit pas faire moins de 80 octets.

9. Avec une FIFO de 10000 octets il y a moins de coupure mais un problème de latence apparaît : quand on arrête d'envoyer les données à la carte son, le son continue de sortie pendant $\frac{10000}{8000}s$ soit plus d'une seconde ce qui est gênant.

10. Généralement dans un jeu il y a des choses qui bougent à l'écran donc on est obligé d'afficher au moins 25 fois par secondes. Pour cela le jeu doit être un processus réveillé périodiquement. À chaque réveille : il génère $\frac{1}{25}s$ de son, soit $\frac{8000}{25} = 320$ octets, il évalue la simulation (qui décide entre autres des sons faire) et puis il génère l'image

11. 3 solutions possibles, dépendant de l'algorithme. On suppose dans les trois cas que les clients se connectent avec des *sockets*.

    - Le mixer a une FIFO par client, périodiquement il vide les *sockets* et remplit les FIFO puis il somme le début des FIFO et envoie sur le fichier son. Une taille de FIFO de 1000 octets est correcte. Prendre plus peut introduire des problèmes de latence à l'arrêt du son. Avantage : on découple la taille de FIFO du mixer et de la carte son. Cela permet de faire croire aux application que la carte son à un buffer raisonnable.

   - Le mixer a une seule FIFO. périodiquement il vide les *sockets* et remplit la FIFO en faisant directement la somme dans celle-ci. Pour chaque client, il faut noter à quel endroit il est dans la FIFO. Cet algo est comme l'autre sauf qu'il utilise moins de mémoire.
   - Le mixer n'a pas de FIFO : périodiquement il prend dans chacun des *sockets* la quantité d'information que la carte son pourra accepter, il fera le mixage en enverra directement. Cette version fait plus d'appel système pour lire les *sockets*

   J'avoue ne pas savoir quelle solution est employée dans la réalité, il suffit d'aller voir les sources.

12. Si le mixer n'est pas réveillé périodiquement (solution simple) cela veut dire qu'il utilise l'appel système `select` pour attendre des données venant de plusieurs fichiers en même temps. Cet appel système permet aussi d'être prévenu quand un périphérique est prêt à recevoir des données. Il est possible que cette solution soit meilleur que l'autre mais cela dépend surtout des heuristiques du système d'exploitation. Il faut donc tester les deux.

13. Un processus utilisateur normale (pas `root`) n'a pas le droit de faire du temps réel ou d'augmenter la priorité de ses tâches. Donc les sons générés risquent d'être hachés. Par contre le serveur de son (le mixer) lui peut être exécuter pas `root` et voir sa priorité augmentée. Donc on passant par cet intermédiaire la qualité du son peut être améliorée.

Remarques :

- La FIFO dont on parle est dans la carte son, pas dans la RAM normale. Le système n'y a généralement as un accès directe, il peut seulement empiler.
- La FIFO n'a aucun problème pour être remplie et vidée simultanément. Il suffit quelle soit représentée comme un buffer circulaire.
- Le temps de remplissage de la FIFO est négligeable par rapport à la temps de sortie du son.
- Le son sort strictement à la même vitesse que la FIFO se vide.
- Le fichier périphérique ne peut être ouvert plus d'un fois c'est clairement dit à la question 11. Donc pas besoin de verrouiller le fichier son.
- A propos de verrouillage. La code : `while(1)  lock ; action ; unlock ;` est "stupide" en effet le temps pendant lequel le verrou est levé est quasiment nul. Autant verrouiller la boucle complète. *On ne doit verrouiller que des portions de programme dont le temps d'exécution est négligeable*
- Par défaut, quand on écrit dans un fichier la fonction se termine que lorsque l'écriture est terminée (ou bien que c'est dans un cache).
- Les *drivers* s'exécutent en mode noyau. Il ne sont généralement pas suspendus par des processus utilisateurs.
- A partir de l'espace utilisateur on ne peut généralement pas connaître la quantité d'octet que l'on peut écrire dans un fichier sans être bloqué.
- Puisque la carte ne fait pas d'acquisition de son, cela n'a pas de sens de faire un lecture du fichier qui représente la carte son.