10 - GESTION DES ENTREES/SORTIES

RAPPELS

Le problème majeur lié à la gestion des périphériques d'entrées/sorties est que chaque périphérique possède ses caractéristiques propres.

Pour permettre une gestion standardisée de toutes les unités périphériques, le périphérique est considéré comme un bus dans lequel circulent des données. Il est fréquemment fait (par exemple sous Unix) la distinction entre les périphériques de type "**flux de caractères**" (les données sont écrites/reçues caractère par caractère) et les périphériques de type "**flux de blocs**" (les données sont écrites/reçues bloc par bloc).

Les entrées/sorties sont gérées à l'aide d'interruptions dédiées **afin de ne pas avoir d'attente active**. Le "driver" ou "pilote" d'un périphérique est "une portion de code propre au périphérique qui sera exécutée lorsque l'interruption correspondant au périphérique surviendra".

Un driver est toujours lié à un périphérique donné. Eventuellement, en fonction de la manière dont il est programmé, à un groupe de périphériques.

1. PERIPHERIQUES EN MODE CARACTERES

On souhaite gérer la ligne série de la machine CDS. On dispose d'une interruption SER_LINE levée à chaque fois qu'un événement survient sur la ligne série. La carte mère gère la ligne série comme suit[1] :

- l'écriture d'un octet à une adresse mémoire donnée d'adresse WRT_SER_LINE provoque l'écriture de l'octet sur la ligne série.
- dès réception d'un caractère sur la ligne série, l'interruption SER_LINE est levée et le caractère se trouve dans un octet en mémoire à l'adresse RD_SER_LINE. Si on lit n'importe quand, on récupèrera n'importe quoi.
- Si on veut connaitre l'état de la ligne série (connectée, non connectée), on peut lire un octet à l'adresse ST_SER_LINE. La valeur rendue est, par convention, FF si le port série est connecté à un cable, 00 sinon.

1.1.

Programmer la primitive système SEND_CHAR (c : caractère).

```
DEBUT
       MASQUE (SER_LINE)
       SI (ST_SER_LINE) = FF ALORS
             (WRT_SER_LINE)  <- c
       SINON
             return -1 // ou traitement de l'erreur
       FSI
       DEMASQUE (SER_LINE)
FIN
```

[1] On s'inspire ici de la technique utilisée par les processeurs Motorola.

par convention, (x) signifie "valeur contenue à l'adresse x"

1.2.

Programmer la routine de traitement de l'interruption SER_LINE rendant dans un registre utilisateur "A" le code du caractère reçu.

```
DEBUT
       MASQUE (SER_LINE)       // masquer les It à partir de SER_LINE
       A <- (RD_SER_LINE)
       DEMASQUE (SER_LINE)
FIN
```

1.3.

Peut-on se contenter de gérer la ligne série de cette manière dans un système multi-utilisateur multi-tâche ? Comment résoudre les problèmes éventuels ?

Bien sûr que non, la ligne série doit être partagée. Pour éviter ce problème, il faut assurer l'exclusion de l'accès au port série pour résoudre le problème de la cohérence dans les lectures/écritures.

2. ENTREES/SORTIES "TAMPONNEES" (OU "BUFFERISEES")

Considérons les primitives ECRIRE_BLOC et LIRE_BLOC qui effectuent respectivement les actions suivantes :

- écriture bloquante d'un bloc de 512 octets sur la ligne série;
- lecture bloquante d'un bloc de 512 octets depuis la ligne série (le processus est alors suspendu s'il n'y a pas de bloc complet);

On souhaite transmettre des blocs de 512 octets sur la ligne série. On s'appuyera sur les mécanismes décrits dans la partie précédente. Dans ce qui suit, on suppose que la ligne série est utilisée en exclusion mutuelle sans s'intéresser aux mécanismes d'implémentation de cette condition.

On considère désormais que la tâche qui a réservé la ligne série est réveillée lorsqu'un bloc de 512 octets a été reçu. La tâche, qui était en attente de la fin de l'entrée/sortie redevient alors éligible.

2.1.

Ecrire la primitive de traitement de l'interruption SER_LINE qui gère la réception du bloc d'entrées/sorties.

Il faut que l'on dispose d'un compteur et d'un espace de 512 octets pour stocker les informations reçues.

La question ressemble alors un peu à ce que l'on a fait avec les terminaux dans le TD sur les interruptions.

```
DEBUT
       MASQUE (SER_LINE)
       cpt <- cpt +  1
       buffer (cpt) <- (RD_SER_LINE)
```

```
        SI cpt = 512 ALORS
                Recopier buffer dans l'espace de données de la tâche
                rendre la tâche "prête" (elle était "suspendue")
                Restaurer le contexte de la tâche qui a le processeur et
provoquer une nouvelle élection
        FSI
        DEMASQUE (SER_LINE)
FIN
```

Remarque : on a décidé qu'une fin d'entrée/sortie doit réveiller la tâche qui était suspendue si elle est plus prioritaire que la tâche qui avait le processeur au moment de la fin d'entrée/sortie si elle était en attente.

2.2.

Ecrire les primitives ECRIRE_BLOC (@bloc) et LIRE_BLOC (@bloc).

```
ECRIRE_BLOC :
DEBUT
        MASQUE (SER_LINE)
        POUR n variant entre 1 et 512 FAIRE
                (WRT_SER_LINE) <- (@bloc + n)
        FPOUR
        DEMASQUE (SER_LINE)
FIN
LIRE_BLOC :
```

On se contente de suspendre l'exécution de la tâche en attendant une entrée/sortie.

```
DEBUT
        MASQUE (SER_LINE)
        Etat de la tâche courante <- suspendu
        Restaurer le contexte de la tâche courante
        Provoquer une élection.
        DEMASQUE (SER_LINE)
FIN
```

2.3.

Avez-vous une remarque à faire à propos de la primitive ECRIRE_BLOC?

Elle n'est pas très performante et suppose que la ligne série va au moins aussi vite que le CPU, ce qui est assez improbable.

On souhaite rendre les écritures non bloquantes pour le processus qui les effectue.

2.4.

Quelle technique proposez-vous? Quelles structures de données gérées par le système introduisez-vous?

introduisez-vous?

Il faut pouvoir chaîner les blocs de données à envoyer dynamiquement en mémoire. Les choses se passent alors en trois temps :

1 Lors de l'écriture d'un bloc, la primitive d'écriture non-bloquante recopie les blocs depuis l'espace de la tâche (il y en a un par tâche ou un pour l'ensemble des tâches, cela dépend) géré par la tâche vers l'espace de la tâche géré par le système.

2 Lorsque le driver de la ligne a terminé d'envoyer le bloc qu'il traitait, il prend le premier bloc à lire dans la file des blocs à traiter.

3 Le driver gère directement le bloc en cours d'écriture dans son espace système.

REMARQUE : insister sur cette structuration en trois niveaux de la mémoire. Il faut faire une recopie au moment du ECRIRE_BLOC_NB pour empêcher l'usager de réutiliser le bloc à d'autres fins.

L'écriture d'un octet sur le périphérique se fait désormais de la manière suivante :

- écriture de l'octet à envoyer à l'adresse WRT_SER_LINE
- lorsque l'écriture est effective (cela peut prendre du temps), l'interruption SER_LINE est levée.

Afin de savoir si SER_LINE correspond à une fin d'envoi ou à une réception, on interroge le mot d'état qui contient :

- FF si le port série n'est relié à rien

- 00 si le port série est connecté et que SER_LINE a été levée à la suite de la réception d'un caractère

- 01 si le port série est connecté et que SER_LINE a été levée pour signaler que l'on a terminé l'écriture du bloc.

2.5.

Programmer la routine de traitement de l'interruption SER_LINE.

Il faut désormais tenir compte de deux comportements possibles (en fonction du mot d'état de la ligne série). Il faut également que l'on ait deux buffers (l'un en entrée, l'autre en sortie) et deux compteurs. On sufixe par "_r" les structures de données associées à la lecture, par "_w" les structures de données associées à l'écriture.

```
DEBUT
   MASQUE (SER_LINE)
   SI (ST_SER_LINE) = 0 ALORS
        // cas de la réception d'un caractère
        cpt_r <- cpt_r +  1
        buffer_r (cpt_r ) <- (RD_SER_LINE)
        SI cpt_r = 512 ALORS
             Recopier buffer dans l'espace de données de la
             tâche
             rendre la tâche "prête" (elle était
             "suspendue")
             Restaurer le contexte de la tâche qui a le
             processeur et Provoquer une nouvelle élection
        FSI
   SINON
        // cas de l'émission
        SI cpt_w = 512 ALORS
             // on passe au bloc suivant s'il y a lieu
```

```
            SI il existe un bloc en attente ALORS
                  le recopier dans buffer_w
                  cpt_w <- 0
            FSI
      FSI
      SI cpt_w /= 512  ALORS
            // émission normale
            cpt_w <- cpt_w + 1
            (WR_SER_LINE) <- (buffer_w (cpt_w ))
      FSI
   FSI
   DEMASQUE (SER_LINE)
FIN
```

2.6.

Quel est le cas où la bufferisation des entrées/sorties devient inefficace ?

Dès que la vitesse des périphériques devient comparable à celles du processeur. La solution est de déporter une partie du traitement des blocs sur un co-processeur dédié aux entrées/sorties.

3. GESTION DES ENTREES-SORTIES DISQUE

On considère une unité de disque à tête mobile dont les caractéristiques sont:

1. Nombre de pistes: 200
2. Nombre de secteurs par piste: 16
3. Informations utilisables par secteur (ou bloc) : 512 octets
4. Temps de démarrage du moteur de la tête : négligeable
5. Temps d'arrêt de la tête: négligeable
6. Temps de positionnement pour deux pistes adjacentes: 1 ms
7. Temps de rotation du disque: 9 ms

Les secteurs d'une même piste se suivent dans l'ordre (i.e., de 0 à 15) dans le sens inverse de la rotation du disque.

3.1.

Quelle est la capacité de stockage de ce disque ?

100 000 * 64 * 512 = 3,2 Mo

3.2.

Quels sont les facteurs intervenant dans le temps de réalisation d'une opération de lecture ou d'écriture d'un secteur ?

Temps d'accès à la piste correspondante

Attente du passage du secteur sous la tête de lecture/écriture (temps de latence).

Temps de transfert.

Temps total = somme des trois

4. ORDONNANCEMENT DES REQUETES

Pour des raisons d'optimisation des E/S, on définit une file d'attente de requêtes pour le périphérique. On se propose dans les questions suivantes de réduire le temps d'accès à une piste. On considère dans les applications numériques la séquence d'accès suivante (numéro de piste, numéro de secteur) :

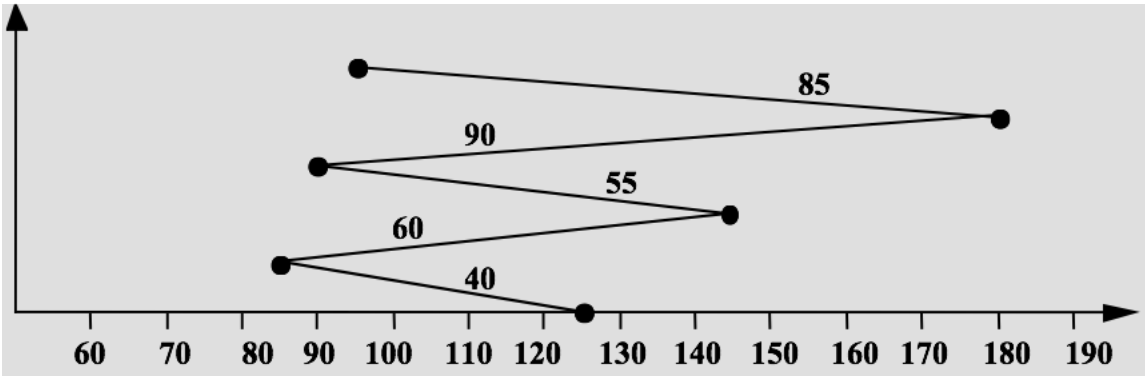
(85, 2) (145, 7) (90, 10) (180, 2) (95, 15)

On suppose aussi qu'à l'instant initial, la tête de lecture/écriture est au-dessus de la piste 125 et du secteur 0.

4.1.

On traite les requêtes dans l'ordre d'arrivée dans la file (FIFO).

- Représenter les mouvements de la tête de lecture (abscisse: numéros de piste, ordonnée: requêtes). Calculer le nombre de pistes parcourues en fin de séquence.
- Calculer le temps total de parcours des pistes pour ces 5 requêtes.
- Existe-t-il une possibilité de famine?



Nombre de pistes parcourues : 330.

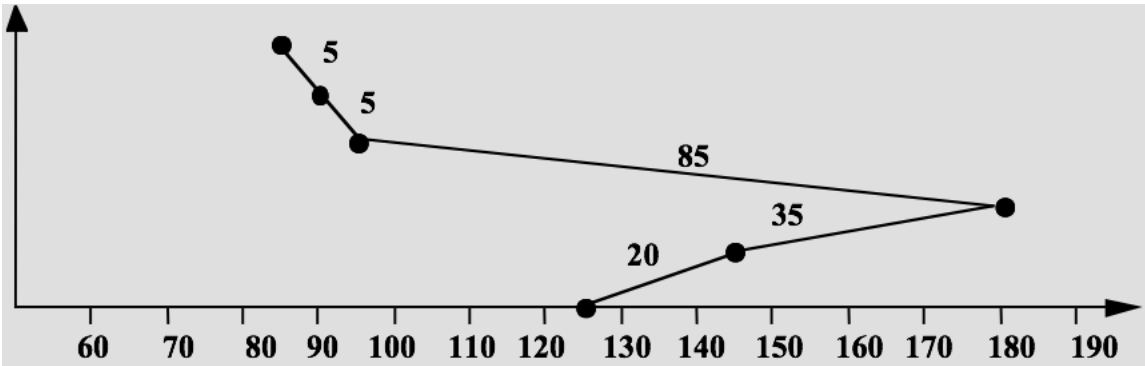
Temps de parcours : 2 * 330 = 660 ms

Pas de famine possible, évidemment.

4.2.

On propose maintenant d'ordonnancer suivant le plus court temps de recherche (PCTR). Pour cela, on va réduire le nombre de pistes parcourues en traitant les requêtes dont le numéro de piste est le plus proche du numéro de piste courant.

- Reprendre les questions posées pour l'ordonnancement FIFO.
- Y a-t-il équité entre les requêtes? si non, dire pourquoi (on considérera une distribution uniforme de requêtes).



Nombre de pistes parcourues : 150 pistes.

Temps de parcours : 2 * 150 + 5 * 2 = 310 ms

Il peut y avoir famine (cf. ci-dessous).

Il n'y a pas équité car sur une distribution uniforme, les pistes du milieu sont favorisées.

4.3.

On suppose à présent que la tête de lecture parcourt le disque dans une logique de balayage (algorithme LOOK, stratégie d'ascenseur). On cherche la requète dont le numéro de piste est le plus distant du numéro de piste courant. On va alors se déplacer vers la piste de la requête, et en profiter pour traîter toutes les requêtes qui se trouvent sur le chemin : lorsque la tête passe devant une piste i, s'il existe une requête pour cette piste, on la traite. On recommence ensuite dans l'autre sens.

- Existe-t-il un problème de famine ?
- Que pensez-vous du parcours de la tête ? Y a-t-il équité entre les requêtes ? Proposez une amélioration de LOOK (C-LOOK).

4.4.

Les derniers algorithmes proposés sont-ils forcément les meilleurs ? Evaluez dans le pire des cas le temps de latence total dans LOOK et comparez avec le temps de parcours des pistes dans le cas de FIFO.

On s'inspire ici de la technique utilisée par les processeurs Motorola.