Université Bordeaux I
Session de septembre 2000

U.F.R. Mathématiques et Informatique
Alain Griffault

Maîtrise d'Informatique
Systèmes d'exploitation
(Durée : 3 heures)

Documents interdits

SUJET + CORRIGE

---

1 Cours : 2 points

Voici les premières lignes du fichier `dma.c` du noyau de Linux 2.0.35.

```
 1 /* $Id: dma.c,v 1.7 1994/12/28 03:35:33 root Exp root $
 2  * linux/kernel/dma.c: A DMA channel allocator. Inspired by linux/kernel/irq.c.
 3  *
 4  * Written by Hennus Bergman, 1992.
11  */
12
13 #include <linux/kernel.h>
14 #include <linux/errno.h>
15 #include <asm/dma.h>
16 #include <asm/system.h>
17
18
19 /* A note on resource allocation:
20  *
21  * All drivers needing DMA channels, should allocate and release them
22  * through the public routines 'request_dma()' and 'free_dma()'.
23  *
24  * In order to avoid problems, all processes should allocate resources in
25  * the same sequence and release them in the reverse order.
26  *
27  * So, when allocating DMAs and IRQs, first allocate the IRQ, then the DMA.
28  * When releasing them, first release the DMA, then release the IRQ.
29  * If you don't, you may cause allocation requests to fail unnecessarily.
30  * This doesn't really matter now, but it will once we get real semaphores
31  * in the kernel.
32  */
```

Question 1.1 *Expliquez les commentaires, sur l'allocation de ressources, compris entre les lignes 19 et 32.*

Réponse à la question 1.1 (2 points) : Le commentaire est destiné aux développeurs de pilotes de périphériques, qui auraient besoin à la fois d'un canal DMA et d'un IRQ. Le commentaire spécifie un contrat : l'utilisateur doit respecter certaines règles s'il ne veut pas avoir des problèmes.
Le problème potentiel est l'interblocage : un pilote réquisitionne une IRQ, un second réquisitionne un canal DMA, puis le premier désire le même canal DMA alors que le second souhaite l'IRQ réquisitionné par le premier.
Le contrat dit que le second pilote n'est pas correctement écrit (il réquisitionne un canal DMA avant l'IRQ).
L'objectif du commentaire est donc d'avoir une prévention de l'interblocage en introduisant un ordre sur les ressources, et en demandant que les allocations soient compatibles avec cet ordre.

2 Producteur/Consommateur avec sémaphores : 5 points

Un processus `Producteur` dépose dans un buffer de taille `n` des messages qui sont ensuite retirés par un processus `Consommateur`.

Question 2.1 *Donnez le code des processus, et pour chacun des sémaphores utilisés, leur rôle, et leur valeur initiale.*

Réponse à la question 2.1 (2 points) :

```
Semaphores : libre  (n,()) // compte les places libres
             occupe (0,()) // compte les places occupées

processus Producteur              processus Consommateur
debut                             debut
tant que vrai faire debut         tant que vrai faire debut
     wait(libre);                      wait(occupe);
     deposer(message);                 retirer(message);
     signal(occupe);                   signal(libre);
     fin                               fin
fin                               fin
```

Le système comprend maintenant `m` couples `(Producteur(i), Consommateur(i))` et le buffer commun à tous est toujours de taille `n`, mais le processus `Consommateur(i)` ne peut pas retirer plus de messages qu'en a déposé le processus `Producteur(i)`.

Question 2.2 *Donnez le code des processus* `(Producteur(i), Consommateur(i))`*, et pour chacun des sémaphores utilisés, leur rôle, et leur valeur initiale.*

Réponse à la question 2.2 (3 points) :

```
Semaphores : libre     (n,()) // compte les places libres
             occupe(i) (0,()) // compte les places occupées par le ième couple

processus Producteur(i)           processus Consommateur(i)
debut                             debut
tant que vrai faire debut         tant que vrai faire debut
     wait(libre);                      wait(occupe(i));
     deposer(message);                 retirer(message);
     signal(occupe(i));                signal(libre);
     fin                               fin
fin                               fin
```

3 Allocation de ressources et interblocage : 4 points

Trois processus partagent trois ressources distinctes `R1, R2, R3`, avec accès exclusif et sans réquisition. Les processus disposent des procédures `allouer(ListeDeRessources)` et `liberer(ListeDeRessources)`, et chaque processus n'utilise pas plus de deux ressources.

Question 3.1 *Donnez un code pour chacun des processus de telle sorte qu'il existe un risque d'interblocage global des trois processus.*

Réponse à la question 3.1 (2 points) :

```
Processus P1         Processus P2         Processus P3
debut                debut                debut
allouer(R1);         allouer(R2);         allouer(R3);
allouer(R2);         allouer(R3);         allouer(R1);
liberer(R1);         liberer(R2);         liberer(R3);
liberer(R2);         liberer(R3);         liberer(R1);
fin                  fin                  fin
```

La séquence suivante provoque un interblocage :

```
Processus P1            Processus P2            Processus P3
debut                   debut                   debut
allouer(R1);
                        allouer(R2);
                                                allouer(R3);
```

Question 3.2 *Donnez un code pour chacun des processus de telle sorte qu'il existe un risque de privation (famine) pour un des trois processus, sans qu'il y ait possibilité d'interblocage.*

Réponse à la question 3.2 (2 points) :

```
Processus P1            Processus P2            Processus P3
debut                   debut                   debut
tant que vrai           tant que vrai           tant que vrai
     debut                   debut                   debut
     allouer(R1,R3);         allouer(R2,R3);         allouer(R1,R2);
     liberer(R3);            liberer(R3);            liberer(R1);
     liberer(R1);            liberer(R2);            liberer(R2);
     fin                     fin                     fin
fin                     fin                     fin
```

La séquence suivante peut être répétée indéfiniment :

```
Processus P1            Processus P2            Processus P3
debut                   debut                   debut
tant que vrai           tant que vrai           tant que vrai
     debut                   debut                   debut
     allouer(R1,R3);         --bloque--              --bloque--         (1)
     liberer(R3);            --debloque--                               (2)
                             allouer(R2,R3);                            (3)
     liberer(R1);                                                       (4)
     --bloque--                                                         (5)
     --debloque--            liberer(R3);                               (6)
     allouer(R1,R3);                                                    (7)
                             liberer(R2);                               (8)
                             --bloque--                                 (9)
     liberer(R3);            --debloque--                               (10)=(2)
```

D'autre part il n'y a pas d'interblocage, car il n'y a jamais de cycle dans le graphe d'allocation des ressources. En effet, il est toujours possible de *sortir de la boucle précédente.*

4 Banquier et interblocage : 3 points

On considère un système à une classe de ressources, où les ressources sont allouées et libérées une par une. Soit `n` le nombre de processus et `m` le nombre de ressources. L'annonce maximale pour chaque processus est au plus égale à `m` et la somme de toutes les annonces est inférieure strictement à `m+n`.

Question 4.1 *En vous inspirant de l'algorithme du banquier, (ou par démonstration directe) et en considérant un état où toutes les ressources sont prises, démontrez que l'interblocage est impossible.*

Réponse à la question 4.1 (3 points) : Notons M_i l'annonce maximale du processus P_i et T_i les ressources détenues par P_i.
Par hypothèse nous savons que $\sum_{i\leq n} M_i < m+n$ et $\forall i, M_i \leq m$
Soit un état où toutes les ressources sont prises. Ainsi $\sum_{i\leq n} T_i = m$

Supposons que l'on soit dans une situation d'interblocage, c'est à dire que chaque processus attend au moins une ressource supplémentaire. Ainsi $\forall i, T_i < M_i$.

$$\begin{aligned}\forall i, T_i < M_i &\Rightarrow \sum_{i\leq n}(1+T_i) \leq \sum_{i\leq n} M_i\\ &\Rightarrow n + \sum_{i\leq n} T_i \leq \sum_{i\leq n} M_i\\ &\Rightarrow n + m \leq \sum_{i\leq n} M_i\end{aligned}$$

Ce qui contredit l'hypothèse $\sum_{i\leq n} M_i < m + n$.
En conclusion, un état où toutes les ressources sont prises ne peut pas être un état d'interblocage. L'interblocage est donc impossible dans un tel système.

5 Gestion mémoire : 6 points

Soit une chaîne de références de pages pour un processus avec `m` cadres de pages (au début tous vides). La longueur de la chaîne de références est `p` avec `n` numéros de pages différents.

Question 5.1 *Donnez une limite inférieure sur le nombre de défauts de pages valable quelque soit l'algorithme de remplacement utilisé.*

Réponse à la question 5.1 (1 point) : Il faut au minimum `n` défauts de pages. (1 par page)

Question 5.2 *Donnez une limite supérieure sur le nombre de défauts de pages valable quelque soit l'algorithme de remplacement utilisé.*

Réponse à la question 5.2 (1 point) : Il faut au maximum `p` défauts de pages. (1 par référence) Par exemple, la suite $1, 2, ..n, 1, 2, ..n, 1, 2, ..$, un algorithme FIFO et $m < n$ provoque un nombre de défauts de pages égal à la longueur de la suite.
Néanmoins, dans le cas où `m` > `n`, il est fort probable que l'algorithme n'utilise au maximum que `m` défauts de pages ; sauf s'il est *optimisé* avec un nombre minimum de cadres devant toujours restés libres, auquel cas, on peut de nouveau avoir besoin de `p` défauts de pages.

Question 5.3 *Vous avez inventé un nouvel algorithme de remplacement de pages, que vous pensez optimal. Votre algorithme présente dans quelques rares cas, l'anomalie de Belady. Votre algorithme est-il optimal ? Justifiez.*

Réponse à la question 5.3 (1 point) : Un algorithme possède l'anomalie de Belady lorsqu'il existe des séquences pour lesquelles le nombre de défauts de page n'est pas une fonction décroissante du nombre de cadres. L'anomalie existe notamment pour l'algorithme FIFO.
Par ailleurs, il existe un algorithme optimal qui consiste à évincer la page dont l'occurrence de la prochaine utilisation est la plus éloignée dans le futur. Il est malheureusement non implémentable car il nécessite la connaissance du futur, mais il ne présente pas l'anomalie de Belady.
Comparons notre nouvel algorithme avec l'algorithme optimal sur les cas rares.

Nombre de cadres	Nombre de défauts de pages		Remarques
	optimal	nouvel algorithme	
m	n	n	il est optimal
$\exists p > m$	$\leq n$	$> n$	anomalie de Belady

Notre nouvel algorithme n'est donc pas optimal.

Considérez le tableau à deux dimensions (le code est en pascal) :

```
var A : array[1..100] of array[1..100] of integer;
```

Les pages de la mémoire paginée peuvent contenir 200 entiers. Un processus occupe la page 0 (adresses de 0 à 199), la page 1 est occupée par les variables `A[1][1], .., A[1][100], A[2][1], .., A[2][100]` et ainsi de suite jusqu'à la page 50.
Le système comporte 3 cadres de pages, utilise l'algorithme LRU, et initialement, le premier cadre contient le processus (la page 0) alors que les deux autres sont vides.

Question 5.4 *Calculez le nombre de défauts de pages pour le processus suivant :*

```
for j := 1 to 100 do
     for i := 1 to 100 faire
          A[i,j] := 0;
```

Réponse à la question 5.4 (1 point) :
La première instruction provoque le chargement de la page 1 (cadre 2)
La seconde instruction est possible avec la page 1.
La troisième instruction provoque le chargement de la page 2 (cadre 3)
La quatrième instruction est possible avec la page 2.
La cinquième instruction provoque le chargement de la page 3 (cadre 2)
et ainsi de suite :
L'instruction $4n + 1$ provoque le chargement de la page $2n + 1$ (cadre 2)
L'instruction $4n + 3$ provoque le chargement de la page $2n + 2$ (cadre 3)

Il y aura donc au total $100 * 100/4 = 5000$ défauts de pages.

Question 5.5 *Calculez le nombre de défauts de pages pour le processus suivant :*

```
for i := 1 to 100 do
     for j := 1 to 100 faire
          A[i,j] := 0;
```

Réponse à la question 5.5 (1 point) :
La première instruction provoque le chargement de la page 1 (cadre 2)
La seconde instruction est possible avec la page 1.
...
La deux-centième instruction est possible avec la page 1.
La deux-cent-unième instruction provoque le chargement de la page 2 (cadre 3)
...
La quatre-centième instruction est possible avec la page 2.
La quatre-cent-unième instruction provoque le chargement de la page 3 (cadre 2)
et ainsi de suite :
L'instruction $400n + 1$ provoque le chargement de la page $2n + 1$ (cadre 2)
L'instruction $400n + 201$ provoque le chargement de la page $2n + 2$ (cadre 3)

Il y aura donc au total $100 * 100/400 = 50$ défauts de pages.

Question 5.6 *Quelles sont les conséquences pour un programmeur de vos réponses aux deux questions précédentes ?*

Réponse à la question 5.6 (1 point) : Faire attention !