UNIVERSITÉ BORDEAUX I
Session de Janvier 1996

U.F.R. Mathématiques et Informatique
J.Bétréma

Maîtrise d'Informatique
Systèmes d'exploitation 1
(Durée: 2 heures)

Documents interdits.

1 Processus

On considère un système de gestion de processus avec n niveaux de priorité, de 1 (le moins prioritaire) à n (priorité maximale). Chaque processus P est doté d'une priorité de base $P.b$ et d'une priorité courante $P.c$, égale à sa priorité de base lors de la création du processus. A chaque priorité i correspond une file d'attente f_i, gérée par l'algorithme FIFO, et qui contient les processus prêts tels que $P.c = i$. Le temps est divisé en quanta, et l'ordonnanceur respecte les deux principes suivants :

- soit m le plus grand entier tel que la file f_m ne soit pas vide; le processus élu est le premier de cette file;
- tout autre processus prêt P tel que $P.c = i < m$ a sa priorité courante incrémentée, et passe de la file f_i à la file f_{i+1}.

1. Quel est le rôle, à votre avis, de la seconde de ces règles?
2. Lorsque le processus élu épuise son quantum de temps, dans quelle file d'attente, à votre avis, l'ordonnanceur doit-il le placer?
3. Lorsqu'un processus bloqué est réveillé par un événement extérieur, quelle doit être, à votre avis, l'action de l'ordonnanceur?

   N.B. Pour cette question comme pour la précédente, plusieurs réponses sont possibles; donnez celle qui vous semble la mieux adaptée, formulez-la de façon simple et précise, et justifiez votre choix.
4. On suppose que $n = 3$, et que quatre processus sont en compétition pour le processeur; P_1 et P_2 ont pour priorité de base 3, P_3 a pour priorité de base 2, et P_4 a pour priorité de base 1; ces processus font uniquement des calculs, et épuisent donc à tour de rôle leur quantum de temps. En utilisant votre réponse à la question 2, simulez l'action de l'ordonnanceur, jusqu'à détection d'un comportement périodique. Une fois le comportement du système devenu périodique, quelles sont les fréquences respectives d'élection des processus P_1 à P_4? Commentez le résultat, et proposez une modification simple de la règle d'incrémentation des priorités, pour augmenter l'avantage accordé aux processus les plus prioritaires.

N.B. Pour lever les ambiguïtés, on supposera que, lorsque le processus P épuise son quantum de temps, il est placé dans une file d'attente *avant* incrémentation éventuelle des priorités, et que celle-ci *ne s'applique pas à* P.

2 Pagination

On considère un système de pagination à trois niveaux, où chaque adresse virtuelle (sur 32 bits) est découpée en quatre champs de longueurs respectives 6, 6, 8 et 12 (en lisant l'adresse de gauche à droite, des chiffres de poids fort vers ceux de poids faible).

1. La table primaire comporte n_1 entrées, et chaque entrée valide contient l'adresse d'une table secondaire; une table secondaire comporte n_2 entrées, et chaque entrée valide contient l'adresse d'une table tertiaire; enfin une table tertiaire comporte n_3 entrées, et chaque entrée valide contient l'adresse d'une page.

   Donner les valeurs de n_1, n_2, n_3 et les tailles, en octets, de chaque type de table.

2. Si chaque entrée d'une table contient l'adresse complète d'une table de niveau inférieur, ou d'une page, les s_i $(i = 1, 2, 3)$ chiffres binaires de poids faibles sont nuls, sous des hypothèses raisonnables.

   Expliquer pourquoi, et donner les valeurs de s_1, s_2, s_3.

   Ces chiffres sont appelés *chiffres de service*, et ne font pas partie, en fait, de l'adresse; ils sont disponibles pour d'autres usages. Donner des exemples typiques de leur utilisation.

3. Soit un processus P, qui utilise les adresses virtuelles comprises entre 0 et 1500K, et celles comprises entre 2Gigas-100K et 2Gigas.

   Combien de tables secondaires (respectivement tertiaires) sont-elles nécessaires pour décrire l'espace d'adressage de P ? Quel est le nombre de mots, respectivement d'octets, occupés par l'ensemble de ces tables ?

4. Le système de pagination découpe la mémoire virtuelle en n_1 blocs de taille t_1, eux-mêmes découpés en n_2 blocs de taille t_2, eux-mêmes découpés en n_3 pages de taille t_3.

   Donner les valeurs de t_1, t_2, t_3 (l'unité est l'octet).

   On suppose que l'un des chiffres de service (voir question 2), noté B, est défini comme suit, dans une entrée de table primaire (respectivement secondaire) : si $B = 1$, cette entrée contient l'adresse physique d'un bloc de taille t_1 (respectivement t_2); si $B = 0$, cette entrée contient, comme d'habitude, l'adresse d'une table de niveau inférieur. Reprendre la question 3 dans ce contexte; quel est l'inconvénient de cette solution ?

5. On désigne par $f(m,i)$ le mot qui résulte de l'annulation des s_i $(i = 1,2,3)$ chiffres binaires de poids faibles du mot m, et par $g(m)$ la valeur, dans m, du chiffre de service B (voir question 4).

   Ecrire en C la fonction f.

   Dans ce qui suit, on utilise la notation hexadécimale, et on suppose que l'entrée 0 de la table primaire contient un mot x tel que : $f(x,1) = 00001b00$, $g(x) = 0$, et que le mot y d'adresse $00001b40$ vérifie : $f(y,2) = 00123400$, $g(y) = 1$. Quelle est la traduction, dans ces conditions, de l'adresse virtuelle $0105c210$? Détaillez vos calculs.

   Si $g(y) = 0$, quelle adresse faut-il consulter pour achever la traduction ?

3 Interblocages

Soit un système avec m processus, n types de ressources, et e_i ressources de type i, pour $1 \leq i \leq n$. Ecrire en C la fonction f de détection d'un interblocage ; les paramètres de cette fonction seront m, n, le vecteur e, la matrice t des ressources allouées, et la matrice r des requêtes. Donner une borne inférieure et une borne supérieure pour la complexité de f, en fonction de m et n.

N.B. On pourra programmer, si l'on préfère, la variante dite *du banquier*.

Corrigé

1 Processus

1. La seconde règle permet d'éviter la *famine* ; sans cette règle, un processus de faible priorité risquerait de ne jamais être élu.

2. Lorsque le processus élu P épuise son quantum de temps, l'ordonnanceur le place dans la file correspondant à sa priorité de base $P.b$; si P gardait sa priorité courante, l'ordonnancement se réduirait rapidement à un ordonnancement FIFO.

3. Lorsqu'un processus bloqué P est réveillé par un événement extérieur, l'ordonnanceur calcule la priorité courante de P en fonction de l'événement qui a réveillé P ; en général cette priorité est élevée, soit pour favoriser l'interactivité (processus réveillé par une frappe au clavier), soit pour améliorer le débit des entrées/sorties (processus réveillé par une interruption de fin de transfert).

4. Evolution des files d'attente (le processus élu est, chaque fois, le premier de f_3) :

quantum	0	1	2	3	4	5	6	7	8
f_3	1,2	2,1,3	1,3,2,4	3,2,4,1	2,4,1	4,1,2,3	1,2,3	2,3,1	3,1,2,4
f_2	3	4			3			4	
f_1	4						4		

quantum	9	10	11	12	13	14
f_3	1,2,4	2,4,1,3	4,1,3,2	1,3,2	3,2,1	2,1,4
f_2	3				4	3
f_1				4		

La configuration 14 est analogue à 9, à l'échange près des processus 1 et 2 (qui sont équivalents) ; donc la configuration 19 sera identique à la configuration 9. La période vaut 10, et durant une période, P_1 et P_2 sont exécutés chacun 3 fois, P_3 et P_4 2 fois.

On constate que les processus de priorité 3 sont peu avantagés, et que celui de priorité 2 n'a pas d'avantage sur celui de priorité 1. Pour augmenter l'avantage accordé aux processus les plus prioritaires, la solution la plus simple est de ne pas incrémenter les priorités lors de chaque intervention de l'ordonnanceur, mais seulement toutes les k interventions, où l'entier k est un paramètre de l'ordonnanceur.

2 Pagination

1. L'index primaire comporte 6 bits, donc la table primaire comporte $n_1 = 2^6 = 64$ entrées ; de même $n_2 = 2^6 = 64$ et $n_3 = 2^8 = 256$. Comme chaque entrée occupe 4 octets, la table primaire occupe 256 octets, comme une table secondaire, et une table tertiaire occupe 1024 octets, soit 1Ko.

2. Les tables sont en principe *alignées* : l'adresse d'une table est un multiple de sa taille. Sous cette hypothèse, une entrée de table primaire, qui contient l'adresse d'une table secondaire, de taille 2^8, se termine par huit chiffres nuls : $s_1 = 8$. De même, une entrée de table secondaire, qui contient l'adresse d'une table tertiaire, de taille 2^{10}, se termine par dix chiffres nuls : $s_2 = 10$. Enfin, une entrée de table tertiaire, qui contient l'adresse d'une page, de taille 2^{12}, se termine par douze chiffres nuls : $s_3 = 12$.

   Il est en fait vraisemblable que les conditions d'alignement sont moins sévères : on peut par exemple supposer que les adresses de tables sont des multiples de la taille minimale 2^8; dans ce cas $s_1 = s_2 = s_3 = 8$.

   Les derniers chiffres sont appelés *chiffres de service*, et ne font pas partie, en fait, de l'adresse; dans une table primaire ou secondaire, l'un d'entre eux indiquent si l'entrée est valide ou non (c'est à dire si les adresses virtuelles correspondantes font partie de l'espace d'adressage du processus); un autre est le chiffre B de la question 4. Dans une table tertiaire, les chiffres de service indiquent en outre si la page est chargée en mémoire, si elle a été modifiée ou consultée, et les droits d'accès du processus sur cette page.

3. L'espace compris entre 0 et 1500K comporte $1500/4 = 375$ pages; il faut donc deux tables tertiaires pour le décrire (seules les $375 - 256 = 119$ premières entrées de la seconde table sont valides), et une table secondaire, dont seules les deux premières entrées sont valides.

   L'espace compris entre 2Gigas-100K et 2Gigas comporte 25 pages, et sa description nécessite une table tertiaire et une table secondaire, dont seule la dernière entrée est valide.

   Soit 3 tables tertiaires et 2 tables secondaires, qui occupent $3 \times 256 + 2 \times 64 = 896$ mots (960 si on compte aussi la table primaire), ou encore 3584 (resp. 3840) octets.

4. $t_1 = 2^{32}/2^6 = 2^{26} = 64$ Mo, $t_2 = 2^{26}/2^6 = 2^{20} = 1$ Mo, $t_3 = 2^{20}/2^8 = 2^{12} = 4$ Ko (on peut évidemment faire les calculs dans l'autre sens, en commençant par la taille t_3 d'une page ...et vérifier qu'on trouve les mêmes résultats).

   Une partie (le premier méga-octet) de l'espace compris entre 0 et 1500K peut alors être mappée globalement par l'entrée correspondante de la table secondaire; on gagne alors une table tertiaire, mais on perd le bénéfice de la pagination pour cette partie de la mémoire.

5. On annule les chiffres de poids faible avec un masque :

```
unsigned int f (unsigned int m, int i) {
  switch (i) {
    case 1: return m & 0xffffff00;
    case 2: return m & 0xfffffc00;
```

```
    case 3: return m & 0xfffff000;
  }
}
```

(on suppose qu'un entier de type `int` occupe 32 chiffres binaires).

L'adresse virtuelle $0105c210$ est traduite comme suit :

- l'index primaire vaut 0, donc on consulte l'entrée 0 de la table primaire, et on obtient l'adresse $00001b00$ de la table secondaire ;
- l'index secondaire vaut 10 (en hexadécimal), donc on consulte l'entrée 10 de la table secondaire, qui est à l'adresse $00001b40$ (car chaque entrée occupe 4 octets), et on obtient l'adresse 00123400, qui est l'adresse physique d'un bloc de 1Mo (puisque le chiffre de service B vaut 1) ;
- les 20 derniers bits de l'adresse, soit $5c210$ en hexadécimal, sont un déplacement dans le bloc de 1Mo, et l'adresse physique finale vaut :

$$00123400 + 5c210 = 0017f610 \ .$$

Si le chiffre de service B vaut 0, il faut considérer l'index tertiaire $5c$, le multiplier par 4, soit 170, et ajouter cette valeur à l'adresse 00123400 de la table tertiaire, pour obtenir l'adresse de la page physique ; l'adresse à consulter vaut donc :

$$00123400 + 170 = 00123570 \ .$$

3 Interblocages

On suppose que dans les matrices, le numéro de ligne désigne le processus, et le numéro de colonne le type de ressource.

```
int executable (int m, int n, int *dispo, int **requete, int *exec);

int interblocage (int m, int n, int *existe, int **tenu, int **requete){
  int
    *dispo = malloc (n * sizeof (int) ),
    *exec  = malloc (m * sizeof (int) ),
    i, j, k;

  /* Calcul des ressources disponibles */
  for (j = 0; j < n; j++) {
    dispo[j] = existe[j];
    for (i = 0; i < m; i++)
      dispo[j] -= tenu[i][j];
  }
```

```c
  /* exec sert a marquer les processus executes */
  for (i = 0; i < m; i++) exec[i] = 0;

  for (i = 0; i < m; i++) {
    j = executable (m, n, dispo, requete, exec);
    if (j < 0) return 1;  /* interblocage */

    /* rendre les ressources allouees a ce processus */
    for (k = 0; k < n; k++)
      dispo[k] += tenu[j][k];
    exec[j] = 1;
  }

  /* tous les processus ont ete executes, pas d'interblocage */
  return 0;
}

/*
  Retourne le numero du premier processus executable (et non execute),
  -1 s'il n'en existe pas.
*/

int executable (int m, int n, int *dispo, int **requete, int *exec) {
  int i, j;

  for (i = 0; i < m; i++)
    if (!exec[i]) {
      for (j = 0; j < n; j++)
        if (requete[i][j] > dispo[j])
          break;
      if (j >= n) return i;
    }
  return -1;
}
```

Note : les fonctions ne doivent évidemment pas modifier les matrices `requete` et `tenu` passées en argument.

La complexité de la fonction `executable` est comprise entre n (si le premier processus est exécutable) et mn (si c'est le dernier); donc la complexité de la fonction `interblocage` est comprise entre mn et m^2n.