Maîtrise d'Informatique **14 Décembre 99**

Systèmes d'exploitation (tronc commun) **Devoir surveillé**

Exercice 1. Cours

Question 1 Sous unix, quelles sont les différences entre «lien symbolique» et «lien en dur»?

Question 2 Quels sont leurs avantages respectifs?

Question 3 Quelle stratégie unix utilise-t-il pour éviter que les utilisateurs communs introduisent des cycles dans le système de fichier?

Exercice 2. Le problème des lecteurs / rédacteurs version moniteur de Hoare.

On considère des processus lecteurs, qui consultent une base de données, éventuellement simultanément, et des processus rédacteurs, qui modifient la base de données; une modification (écriture) en cours exclut toute autre lecture ou modification simultanée.

De plus on souhaite qu'un rédacteur puisse modifier la base dès que les lectures en cours sont terminées, sans être bloqué par de nouvelles arrivées de lecteurs.

Question 4 A l'aide d'un moniteur de Hoare, écrire, outre le code d'initialisation du moniteur, les procédures *debut_lecteur, fin_lecteur, debut_redacteur, fin_redacteur*, qui permettent la synchronisation des processus; par exemple un lecteur exécutera:

```
debut_lecteur;
consulter la base;
fin_lecteur;
```

Rappels sur les moniteurs Le type moniteur est une structure de synchronisation de haut niveau, ayant la même puissance de synchronisation que les sémaphores. Un moniteur comprend des variables, des conditions, des procédures et un code d'initialisation. Seules les procédures peuvent être manipulées par les processus, et le mécanisme du moniteur garantit l'exécution de chaque procédure en exclusion mutuelle.

La synchronisation des processus est obtenu à l'aide des procédures *wait* et *signal* appliquées sur les variables de type condition. L'action X.signal provoque le réveil de tous les processus qui ont auparavant effectué X.wait.

Exercice 3. Pagination et multiplication de matrices.

Les algorithmes suivants placent dans la matrice resultat (initialement nulle) la multiplication des matrices a et b de dimensions $n \times n$:

```
ALGO 1
 int i, j, k ;
  for(i = 0; i < n ; i++)
      for(j = 0; j < n; j++)
          for(k=0; k < n ; k++)
            resultat[i][j] += a[i][k] * b[k][j] ;
ALGO 2
int i, j, k ;
  for(i = 0; i < n ; i++)
      for(j = 0; j < n; j++)
          for(k=0; k < n ; k++)
            resultat[i][k] += a[i][j] * b[j][k] ;
```

En supposant que

- la mémoire primaire est constituée de 4 cadres de n mots,

– un cadre contient en permanence l'ensemble du code (l'accès au code ne provoque pas de défaut de page),
– chaque ligne de chaque matrice occupe exactement une page [les matrices sont rangées par lignes, le premier élément de chaque matrice étant aussi le premier mot de sa page],

Question 5 Calculer le nombre minimal de défauts de page générés par l'exécution de chacun des algorithmes.

Exercice 4. MMU

Le processeur Alpha AXP 21064 dispose d'**adressage virtuelle sur 64 bits**. Cette adresse est traduite en **adresse physique de 34 bits** en utilisant la technologie des tables de pages à trois niveaux. L'adresse virtuelle est décomposée de façon suivante :

bits complémentaires	indice niveau 1	indice niveau 2	indice niveau 3	offset

où offset donne le décalage (le numéro de l'octet) dans la page physique et « indice niveau i » représente l'indice dans la table de pages de niveau i.

Sachant que :

– la taille d'une page est de 8K octets,
– la table de pages de chaque niveau est exactement de la taille d'une page,
– une entrée de la table de pages (PTE) est de longueur de 8 octets.

Question 6 Donner le nombre de bits nécessaires au codage l'offset.

Question 7 Donner le nombre de bits nécessaires au codage de chaque indice.

Une entrée dans la table de page (PTE) a la structure suivante :

PTE:

adresse	info

La partie « adresse » de la table de niveau i donne, pour $i < 3$, l'adresse physique de la table de niveau $i + 1$. Pour le niveau 3, la partie « adresse » donne les bits de poids fort de l'adresse physique de la page.

Question 8 Donner le nombre de bits nécessaires au codage de la partie « adresse » du PTE.

Pour un processus donné, la table de niveau 1 est organisée de la façon suivante:

– le segment de code et celui du tas sont codés à partir de l'entrée 0,
– la pile se termine à l'entrée $n - 1$, la dernière entrée (comme souvent pour empiler on décremente le pointeur de pile).

Soit un processus dont le segment de code et le tas occupent 28832K et dont la pile occupe 1608K.

Question 9 Donner le nombre de tables de pages de niveau 2, puis de niveau 3, occupées par ce processus. Dessiner la structure de tables de pages pour ce processus.

NB. Contrairement au processeur SPARC, le processeur Alpha ne supporte pas l'adressage linéaire – il faut donc toujours passer par les différents niveaux pour obtenir une adresse physique.

Correction

Question 1

Sous Unix, les fichiers sont identifiés par un unique i-node (noeud d'information contenant le propriétaire, les droits, la taille, des pointeurs vers la zone de données...), ces structures étant gérées par le système. Un lien est une entrée (article, enregistrement) dans un répertoire, chaque entrée possédant un type (répertoire, tube, lien en dur, lien symbolique,...).

Liens symbolique et liens en dur permettent tous deux le "partage" des fichiers, c'est à dire l'accès à un même fichier à partir de liens différents. Un lien en "dur" contient un numéro d'i-node désignant le fichier cible, un lien symbolique contient un i-node pointant vers une zone de donnée contenant une chaine de caractères décrivant (ou non) un chemin donnant accès au lien ciblé.

Question 2

Avantages des liens en dur : ils consomment moins de ressources système (temps cpu, accès disque, espace mémoire) puisque l'accès au numéro d'i-node est direct; de plus chaque i-node possède un compteur de référence dénombrant le nombre de lien en dur le désignant, le fichier et l'i-node sont restitués au système lorsque ce compteur passe à 0.

Avantages des liens symboliques : ils permettent de pointer sur des répertoires, sur des fichiers n'appartenant pas au système de fichiers local, sur des fichiers n'existant pas (ce qui permet la destruction temporaire du fichier cible).

Question 3 Un utilisateur commun ne peut pas ajouter un lien en dur sur un fichier pré-existant et désignant un répertoire. Ainsi, l'ensemble des répertoires liés en dur forment un arbre, l'ensemble des liens en dur forment un DAG. Liens symboliques et liens en dur sont donc complémentaires.

Question 4

Précision : un moniteur permet d'assurer l'exécution mutuelle sur l'ensemble des procédures et variables du moniteur.

```
Moniteur lecteur_redacteur :

condition lecteurs_en_attente ;
condition redacteurs_en_attente ;

boolean ecriture_en_cours ;
integer nombre_redacteurs ;
integer nombre_lecteurs ;

init:
  nombre_redacteurs = 0 ;
  nombre_lecteurs = 0 ;
  boolean ecriture_en_cours = 0 ;
end ;

entry debut_lecture :
  while (nombre_redacteurs <> 0)
    lecteur_en_attente.wait
  nombre_lecteurs++ ;
end ;

entry fin_lecture :
  nombre_lecteurs-- ;
```

```
  if (nombre_redacteurs <> 0 && nombre_lecteurs == 0)
     redacteurs_en_attente.signal
end ;

entry debut_ecriture :
  redacteurs_en_attente++ ;
  while (ecriture_en_cours || nombre_lecteur <> 0 )
    redacteurs_en_attente.wait
  ecriture_en_cours = true ;
end ;

entry fin_lecture :
  nombre_redacteur-- ;
  ecriture_en_cours = false ;
  if (nombre_redacteur == 0)
    lecteur_en_attente.signal
  else
    redacteur_en_attente.signal
end ;
```

Question 5

Configuration initiale:

- le code occupe le premier cadre
- le résultat[i] occupe le second cadre
- la ligne a[i] occupe le troisième cadre
- la ligne b[k] (b[j] pour le second algo) occupe le quatrième cadre

On remarque alors que l'on aura systématiquement et uniquement un défaut de page lorsque deux accès consécutifs à une même matrice ne concerne pas la même ligne.

Ainsi, on observe que chaque accès à la variable `b[j][k]` provoque un défaut de page, cet accès provoque donc n^3 défauts ; i étant modifié n fois, `resultat[i][j]` et `a[i][k]` provoquent n défauts chacun, on a donc $n^3 + 2n$ défauts.

Pour le second algorithme on observe que la variable `j` n'est plus modifiée que n^2 fois, la variable `i` étant modifiée n fois, on a donc $n^2 + 2n$ défauts.

Question 6 L'offset donne le décalage à l'intérieur d'une page. Puisque $8K = 2^{13}$ nous avons 13 bits d'offset.

Question 7

La taille d'une table est de $8K$ mais chaque entrée de la table est de taille 8 octets, ainsi une table (pour chaque niveau) possède $8K : 8 = 1K = 2^{10}$ entrées; aussi il faut 10 bits pour coder l'indice.

Question 8 L'adresse physique est sur 34 bits. De ces 34 bits, les 13 bits de poids faible proviennent soit de l'offset, soit l'indice correspondant de niveau i. Pour connaître l'adresse physique il nous manque encore $34 - 13 = 21$ bits, ces 21 forment la partie “adresse” de PTE.

Question 9

Le code et le tas occupent $28832 : 8 = 3604$ pages. Une table de niveau 3 permet de adresser $1K = 1024$ pages (c.f. Question 7) alors il faut 4 tables de niveau 3, trois premières complétement occupées et la quatrième qui utilise $3604 - 3 \times 1024 = 532$ entrées. Il faut aussi ajouter une table de niveau 2.

Pour la pile il suffit une table de niveau 3 et une de niveau 2.

Enfin une table de niveau 1 pour tout le processus.

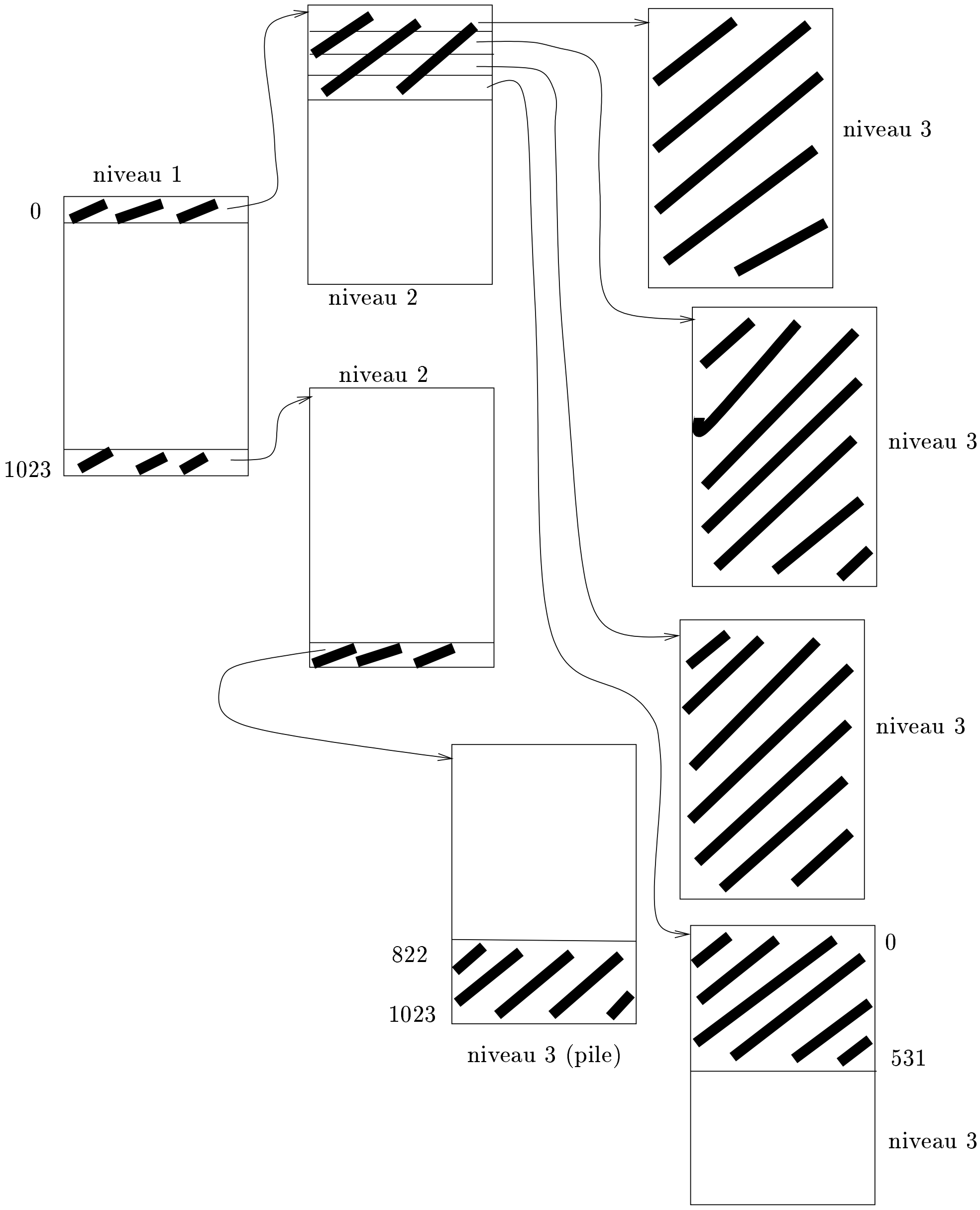

Fig. 1 – *Tables de pages.*