Examen LU3IN010

L3 – Licence d'Informatique -Mai 2024

2h - Aucun document autorisé - Barème donné à titre indicatif

1. QUESTIONS DE COURS (1,5 POINTS)

Les réponses à ces questions doivent être courtes (1-2 lignes).

1.1. (0,5 point)

Dans une mémoire paginée, les adresses générées par les compilateurs dans les programmes exécutables sont-elles des adresses physiques ou virtuelles ?

Adresse virtuelle, le compilateur ne connait pas l'implantation physique des programmes

1.2. (0,5 point)

Une interruption exécute-t-elle un code du noyau ou de l'utilisateur ?

code noyau

1.3. (0,5 point)

Soit une mémoire simplement paginée où la table des pages d'un processus est contenue dans une case mémoire. On suppose que le temps de moyen d'accès à une case mémoire est de 100 ns et le temps moyen d'accès à la TLB est de 10 ns. Soit une TLB avec un taux de hit de 98%. Quel est le temps moyen d'accès à une donnée ?

temps moyen = 100 + 0.98*(10) + 0.02*(10+100) = 112 ns

2. MEMOIRE (4 POINTS)

On dispose d'une machine **simplement paginée** avec des pages de 1024 octets. On considère 2 processus P1 et P2.

Pour les 2 processus, leur **code** commence à l'adresse virtuelle 0 et se termine à l'adresse virtuelle 4095 octets, leur **pile** est située entre les adresses virtuelles 20480 et 40959 et leur zone contenant les **données** est située entre les adresses virtuelles 10240 et 20479.

La page 0 de P1 et P2 est stockée dans la case 10 en RAM. La page 20 de P1 est stockée dans la case 40 et la page 11 de P2 dans la case 80.

2.1. (0,5 point)

Quels sont les numéros des pages associés au code, à la pile et aux données ?

Code : pages 0 à 3 (0 div 1024 = 0, 4095 div 1024 = 3)
pile : pages entre 20 et 39
données: pages entre 10 et 19

2.2. (0,5 point)

Les variables *r* et s de P1 situées respectivement aux adresses 20500 et 10500 sont-elles des variables locales ou globales ? Justifiez.

r est locale, elle fait partie de la pile.
s est globale, elle fait partie des données.

2.3. (0,5 point)

Pourquoi P1 et P2 ont leur page 0 stockée dans la même case mémoire ?
Serait-il possible que P1 stocke aussi sa page 11 dans la case 80 ?

P1 et P2 partagent le même code en lecture
Ce n'est pas possible pour P1 de partager la case 80 (il n'y a de partage des données)

2.4. (1, 5 points)

On suppose que **la case 20 est la première case libre**. Que se passe-t-il lorsque :
a) P1 fait un accès en écriture à l'adresse 20501 ?
b) P1 fait un accès en lecture à l'adresse 10260 ?
c) P2 fait un accès en écriture à l'adresse 1000 ?

Vous préciserez les actions faites par le matériel et celles faites par le système. S'il n'y a pas d'erreur, vous donnerez l'adresse physique correspondant à l'accès.

a)
MMU : accès <20,21>, droit Ok, Présence Ok, ad physique 40*1024 + 21 = 40981
b)
MMU : accès page <10,20>, IT défaut de page ,
Système : traite défaut de la page 10, case 20 allouée, mise à jour TP
MMU : ad physique =20*1024 +20 = 20500.
c)
MMU : IT faute sur droit d'accès page 0
Système : traite l'erreur et tue P2

2.5. (1 point)

Faites un schéma des tables des pages des processus P1 et P2 à la suite des accès a) et b) de la question 2.4, en précisant les cases, les droits et le bit de présence.

Remarque : en général, il n'y a pas de bit X dans les TP

TP P1
<page,case,P,droits>

<0,10,1, L(X)>
<1,0,0, L(X)>
<2,0,0, L(X)>
<3,0,0, L(X)>

…
<10,20,1,LE>
<11,-,0, LE>
…
<19,-,0, LE>

<20,40,1,LE>
<21,-,0,LE>
…
<39,-,0,LE>

TP P2
<0,10,1, L(X)>
<1,0,0, L(X)>
<2,0,0, L(X)>
<3,0,0, L(X)>
…
<10,-,0,LE>
<11,80,1, LE>
…
<19,-,0, LE>

<20,-,0,LE>
<21,-,0,LE>
…
<39,-,0,LE>

3. REMPLACEMENT DE PAGES (4 POINTS)

On dispose d'une machine simplement paginée avec des pages de 1024 octets. Soit un processus P.

On dispose de 3 cases libres dont les numéros sont 20, 30 et 40. On considère qu'aucune page n'est initialement chargée en mémoire.

La stratégie de remplacement de page est la suivante :

Dans la table des pages, à chaque page est associé un compteur de 2 bits. Ce compteur est initialement à 0. Le matériel incrémente le compteur à chaque accès à une page, après 3 incrémentations le compteur n'est plus augmenté. Lorsqu'une page est évincée (déchargée) de la mémoire son compteur est remis à 0.

En cas de remplacement, la page victime est celle en mémoire dont le compteur a la valeur la plus faible. Si deux victimes ont la même valeur on adopte un ordre FIFO.

Le processus P fait la suite d'accès suivant aux pages :

1 1 10 1 21 0 10 21 10 0

3.1. (3 points)

Quel est le nombre de défauts de pages engendré par cette séquence. Faites un tableau pour indiquer à quel moment ces défauts ont lieu.

Donnez l'état de la table des pages à la fin de la séquence en indiquant le numéro de page, le numéro de case et la valeur du compteur.

Seg/page	1	1	10	1	21	0	10	21	10	0
0						30,1		X		30,1
1	20,1	20,2		20,3						
10			30,1			X	40,1		40,2	
21					40,1		X	30,1		X
Def.	D		D		D	D	D	D		D

Table des pages :
<Page, case, compteur>
0, 30, 1
1, 20,3
10, 40,2

3.2. (1 point)

Pourquoi cette stratégie ne prend pas en compte la localité temporelle des accès ?
En quelques lignes proposez une modification de cette stratégie à compteur pour que la localité temporelle soit considérée.

- Le nombre de défauts est pris en compte (pas l'instant) : une page anciennement active risque de prendre inutilement une case en mémoire (c'est le cas de la page 0 dans l'exemple)
- 1 solution : le système remet régulièrement les compteurs à 0

4. SYNCHRONISATIONS PAR SEMAPHORES (6,5 POINTS)

Nous considérons le problème du Producteurs-Consommateurs étudié en cours et TD mais avec **un seul processus *Consommateur*** et **un nombre infini de processus du type *Producteur*** qui arrivent en continu dans le système.

Soit un tampon **T** constitué de **N cases**. Chaque case est destinée à recevoir une valeur entière, reçue en paramètre par un processus *Producteur*, qui sera consommée par le processus Consommateur. Le tampon est géré de façon circulaire.

Si en arrivant dans le système, le tampon est plein, un processus *Producteur* se termine. Dans le cas contraire, en respectant la gestion circulaire de T, il dépose la valeur qu'il a reçu en paramètre dans une case libre du tampon et se termine également.

Le processus *Consommateur* ne se termine jamais. Si le tampon est vide, il se bloque. Sinon, tant qu'il y a de cases occupées, il doit les consommer (afficher la valeur) en respectant la gestion circulaire de T.

Nous voulons programmer le processus *void Consommateur ( )* et le processus *void Producteur (int val)* en utilisant des sémaphores et des variables partagées. Votre solution doit être aussi concurrente (parallèle) que possible.

Un sémaphore est manipulé à l'aide des opérations indivisibles suivantes :

`SEM *CS(cpt)` : Création d'un sémaphore dont le compteur est initialisé à "cpt".

`P(sem)` : Demande d'acquisition d'une ressource. Si aucune ressource n'est disponible, le processus est bloqué.

`V(sem)` : Libération d'une ressource. Si la file d'attente n'est pas vide, un processus est débloqué.

Note : Un sémaphore n'est jamais initialisé à une valeur négative et la valeur du compteur n'est pas accessible par les processus.

La déclaration d'une variable partagée est faite avec le type SHARED. *N* est une constante.

4.1. (3,5 points)

Donnez le code des processus *void Consommateur ( )* et le code de *void Producteur (int val)*. Spécifiez les sémaphores et/ou variables partagées utilisés ainsi que leur initialisation.

```
#define N 10
SHARED int T[N];
SHARED int ip =0; /* indice de T partagé entre le Producteurs */
SHARED int nb_cases_libres;  /* partagée entre le Producteurs */

*SEM Cons = CS (0) : /* bloquer le Consommateur */
*SEM Mutex= CS(1) ;



void Consommateur ( ) {
```

```
 int ic = 0 ; /* indice de T pour la consommation */
 while (true) {
   P (Cons) ;
   printf ("%d\n", T[ic]);
   ic=(ic+1)%N ;
   P(mutex);
   nb_cases_libres++;
  V(mutex);
}

void Producteur ( int val) {
 int I ;
  P(mutex) ;
  if (nb_cases_libres > 0) {
       I=ip: /* prochaine case libre */
       nb_cases_libres--:
       ip=(ip+1)%N ;
       V(mutex) ;
       T[I]=val;
        V(Cons);
  }
  else /* pas de case libre */
        V(mutex) ;
}
```

Nous souhaitons modifier le code de la question précédente pour qu'un processus *Producteur*, qui a déposé une valeur dans une case de T, ne se termine que lorsqu'il est sûr que la valeur a été consommée par le processus *Consommateur*.

4.2. (3 points)

Modifiez le code des processus de la question précédente en conséquence. Spécifiez les sémaphores et/ou variables partagées utilisés ainsi que leur initialisation.

```
#define N 10
SHARED int T[N];
SHARED int ip =0; /* indice de T partagé entre le Producteurs */
SHARED int nb_cases_libres;  /* partagée entre le Producteurs */

*SEM Cons = CS (0) ; /* bloquer les consommateur */
*SEM Mutex= CS(1) ;
*SEM Attendre[N] /* pour attendre que le message soit consommé */
```

```
  For (j=0, j<N, j++) Attendre [j]= CS(0);

void Consommateur ( ) {
 int ic = 0 ; /* indice de T pour la consommation */
 while (true) {
    P (Cons) ;
    printf ("%d\n", T[ic]);
    V(Attendre [ic]) ;
    ic=(ic+1)%N ;
}
void Producteur ( int val) {
 int I ;
 P(mutex);
   if (nb_cases_libres > 0) {
        I=ip: /* prochaine case libre */
         nb_cases_libres--:
         ip=(ip+1)%N ;
         V(mutex) ;
         T[I]=val;
         V(Cons);
          P(Attendre [I]) ; /* attendre le Consommateur  consommé */
          P(mutex) ;
          nb_cases_libres++;
          V(mutex);
  }
  else /* pas de case libre */
         V(mutex) ;
}
```

5. FICHIERS (4 POINTS)

Soit un disque avec des blocs de 512 octets.

5.1. (1 point)

On considère un fichier f1 composé d'un seul bloc non présent en mémoire. Combien d'octets sont transférés en mémoire si un processus utilisateur demande la lecture d'un octet de f1 ? Ce(s) octet(s) sont transférés dans les espaces système, utilisateur ou les deux ? Justifiez votre réponse.

512 octets du bloc sont copiés (périphérique en mode bloc) dans les l'espace noyau (le buffer ou page cache), l'octet demandé est ensuite copié dans l'espace utilisateur.

Soit un système de fichier Unix avec des blocs de 512 octets dont chaque fichier est associé à une inode. Dans cette structure, on trouve (entre autres) 13 entrées :

• les 10 premières entrées référencent des blocs de données sur le disque,

• la 11ème référence un bloc de contrôle qui référence des blocs de données (simple indirection),

• la 12ème référence un bloc de contrôle qui référence des blocs de contrôle qui référencent des blocs de données (double indirection),

• la 13ème référence un bloc de contrôle qui référence des blocs de contrôle qui référencent des blocs de contrôle qui référencent des blocs de données (triple indirection)

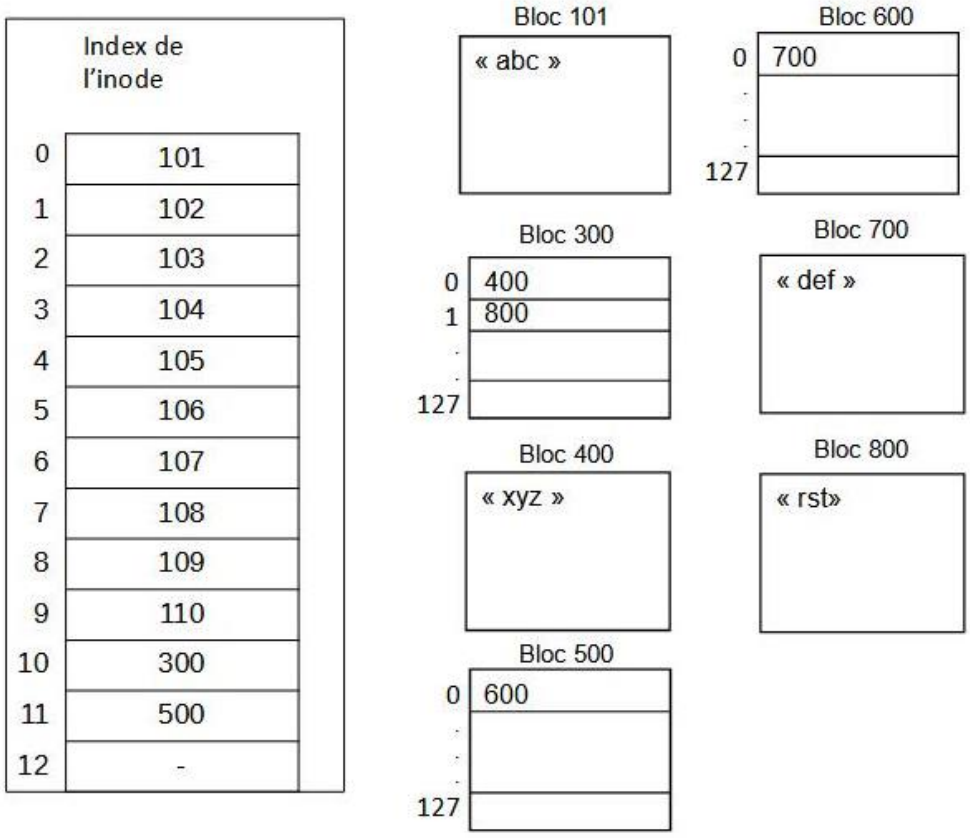
Les blocs de contrôle (index) permettent de stocker 128 numéros de blocs.

5.2. (1 point)

Soit un fichier f1 de taille 102400 octets. Quel le nombre total de blocs occupé sur disque par ce fichier en considérant les blocs d'index ? Justifiez votre réponse en précisant le nombre de blocs de données et le nombre de bloc index ainsi que niveau d'indirection (simple, double ou triple) de ceux-ci.

102400 => 200 blocs de données de 512 octets.
+1 bloc d'indirection simple et 2 blocs d'indirection double soit 203 blocs

Soit un fichier f2. Le schéma suivant représente son inode ainsi que le contenu de plusieurs blocs.



5.3. (1 point)

Quels caractères se trouvent aux déplacements 0, 5122 et 70 657 du fichier (indication : 70 656 = 138*512)

a, z, e

5.4. (1 point)

On suppose qu'initialement l'inode du fichier f2 est en mémoire et qu'aucun de ses blocs (données et index) n'est en mémoire. Une fois accédé, chaque bloc reste en mémoire. Quel est le nombre d'entrées/sorties entrainé par la lecture des caractères numéro 5122 puis 5631 du fichier (5632 = 11*512) ? Vous préciserez dans l'ordre les blocs accédés.

Accès bloc 300 – 1 E/S
Accès bloc 400 – 1 E/S
Accès bloc 300 – 0 E/S
Accès bloc 400 – 0 E/S
Total 2 E/S