Examen LU3IN010

L3 – Licence d'Informatique -Mai 2025

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

1. MEMOIRE (4 POINTS)

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

Pour les 2 processus, leur **code** commence à l'adresse virtuelle 1024 et se termine à l'adresse virtuelle 3071 mots, leur **pile** est située entre les adresses virtuelles 40960 et 81919 et leur zone contenant les **données** est située entre les adresses virtuelles 4096 et 8191.

La deuxième page de code de P1 et P2 est stockée dans la case 20 en RAM. La première page de données de P1 est stockée dans la case 10 et la deuxième page de pile de P2 dans la case 30.

1.1. (1 point)

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

Code : pages 1 et 2 (1024 div 1024 = 1, 3071 div 1024 = 2)
pile : pages entre 40 et 79
données: pages 4 et 7

1.2. (0,5 point)

Pourquoi P1 et P2 ont leur deuxième page de code stockée dans la même case mémoire ? Serait-il possible que P1 et P2 stockent leur page 0 de données dans la même case ?

P1 et P2 partagent le même code en lecture
Pas de partage pour les données

1.3. (0,5 point)

Les variables *x* et *y* de P1 situées respectivement aux adresses 50000 et 8000 sont-elles des variables locales ou globales ? Justifiez

x est locale, elle fait partie de la pile.
y est globale, elle fait partie des données.

1.4. (1 point)

Faites un schéma des tables des pages des processus P1 et P2 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>
…
<1,-,0,LX>
<2,20,1,LX>
…
<4,10,1,LE>
<5,-,0,LE>
…
<7,-,0,LE>
…
<40,-,0LE>
…
<79,-,0,LE>

TP P2
…
<1,-,0,LX>
<2,20,1,LX>
…
<4,-,0,LE>
<5,-,0,LE>
…
<7,-,0,LE>
…
<40,-,0,LE>
<41,30,1,LE>
<42,-,0,LE>
…
<79,-,0,LE>

1.5. (1 point)

Que se passe-t-il lorsque :
a) P2 fait un accès en écriture à l'adresse 1500 ?
b) P1 fait un accès en écriture à l'adresse 4101 ?
c) P2 fait un accès en écriture à l'adresse 4101 ?

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

a) MMU : accès page 1, IT faute sur droit d'accès, Système : traite l'erreur et tue P2
b) MMU : accès page <4,5>, adresse physique à 10*1024+5 = 10245
c) MMU : accès page <4,5> , IT défaut de page , Système : traite défaut de la page 4 de P2

2. REMPLACEMENT DE PAGES (3,5 POINTS)

On considère une mémoire paginée. Aucune page n'est initialement chargée en mémoire. Le système dispose de 3 cases libres, les cases : 10, 20 et 30.

L'algorithme de remplacement de pages est type LFU (Least Frequently Used). Il fonctionne de la manière suivante :

Un compteur, initialement à 0, dans chaque entrée de la table des pages compte pour chaque page le nombre d'accès. **Ce compteur est maintenu même si la page se trouve déchargée sur disque** (c'est-à-dire que lorsque une page est déchargée, elle recommencera avec le compteur qu'elle avait avant d'être déchargée).

En cas de remplacement de page, la page à évincer est celle en mémoire dont la valeur du compteur est **la plus faible**. Si plusieurs pages ont la même plus petite valeur, l'ordre FIFO est appliqué.

Soit un processus P faisant la suite de références aux pages suivante :

0, 1, 0, 0, 1, 4, 3, 4, 3, 4, 2, 0

2.1. (3 points)

Quel est le nombre de défauts de page engendré par cette séquence ? Faites un tableau pour indiquer à quel moment ces défauts ont lieu. Indiquez l'état de la table des pages à la fin de cette séquence ainsi que les valeurs des différents compteurs.

Pages :	0	1	0	0	1	4	3	4	3	4	2	0
0	10,1		10,2	10,3								10,4
1		20,1			20,2				X			
2											20,1	
3							30,1	X	20,2		X	
4						30,1	X	30,2		30,3		
	D	D				D	D	D	D		D	

7 défauts

Page	P	Compteur	Case
0	1	4	10
1	0	2	
2	1	1	20
3	0	2	-
4	1	3	30
..			

2.2. (0,5 point)

Entre LFU et LRU lequel des deux algorithmes respecte le plus la localité temporelle ? Justifiez brièvement.

LRU respecte plus la localité temporelle car il prend en compte d'instant d'utilisation de la page. LFU ne prend pas en compte la date d'utilisation

3. SYNCHRONISATIONS PAR SEMAPHORES (6 POINTS)

Dans cet exercice, on veut paralléliser au maximum des processus effectuant un calcul. Il y a plusieurs processus de calcul et un processus coordinateur.

Les processus de calcul exécutent tous la fonction suivante qui met le résultat d'un calcul dans une *variable locale* v :

```
Traitement(){
int v ;
// A COMPLETER
v = Calcul() ;
// A COMPLETER
}
```

Le processus coordinateur exécute la fonction suivante :

```
Coordinateur() {
// A COMPLETER
}
```

3.1. (1,5 points)

Il y a N processus de calcul. Initialement, les processus de calcul doivent être bloqués avant d'effectuer leur calcul. Le processus coordinateur lance le réveil de tous les processus. Le coordinateur connaît N.

Donnez le pseudo-code des fonctions Traitement et Coordinateur en utilisant les opérations **P et V** vues en TD. Vous préciserez les sémaphores utilisés ainsi que leur initialisation.

```
Traitement(){
int v ;
P(S) ;
v = Calcul() ;
}


Coordinateur() {
for (i=0 ; i<N ; i++)
        V(S)
}
```

```
S = CS(0) ;
```

3.2. (1,5 points)

Le nombre de processus de calcul est maintenant quelconque : N est inconnu. Modifiez les deux fonctions en veillant à ce que tous les processus de calcul soient bien réveillés. On considère toujours qu'initialement, les processus de calcul doivent être bloqués avant d'effectuer leur calcul. En revanche, ils ne sont pas nécessairement réveillés par le coordinateur.

```
Traitement(){
int v ;
P(ST) ;
V(ST) ;
v = Calcul() ;
}


Coordinateur() {
        V(ST)
}

ST = CS(0) ;
```

3.3. (3 points)

Le nombre de processus de calcul est toujours inconnu mais le coordinateur, avant de se terminer, doit maintenant se bloquer en attendant que les M premiers processus aient fini leur calcul (on suppose que M est inférieur au nombre total de processus de calcul).

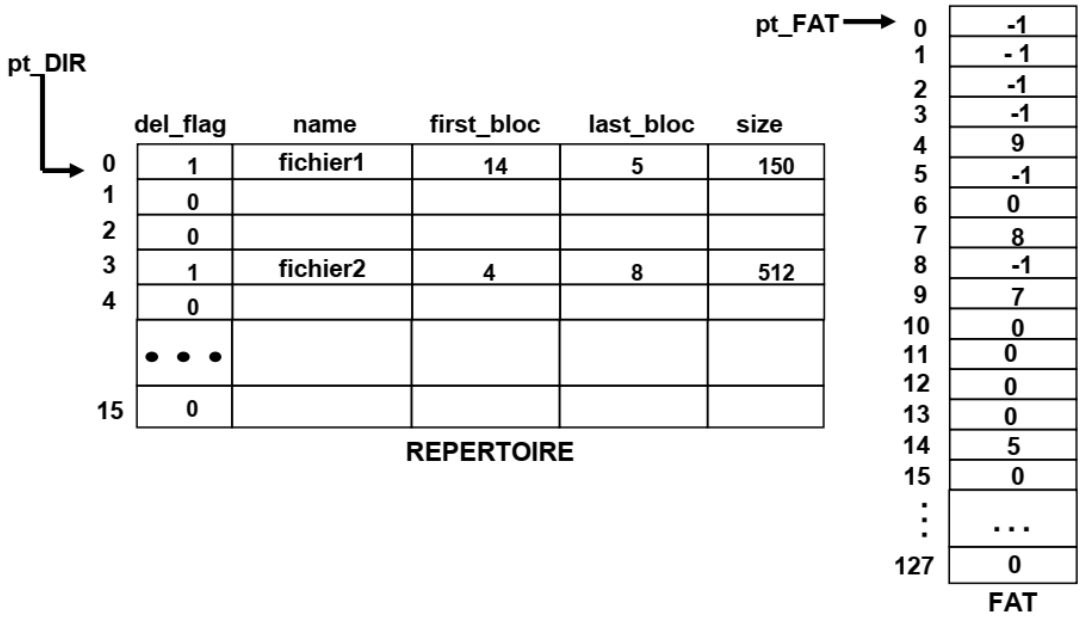
Avant de quitter la fonction Traitement, les M premiers processus de calcul (uniquement eux) doivent attendre que le coordinateur soit débloqué.

Modifiez les fonctions en conséquence en précisant les sémaphores utilisés, leur initialisation ainsi que d'éventuelles variables locales et partagées.

```
Traitement(){
int v ;
bool premier = false ;
P(ST) ;
V(ST) ;
v = Calcul() ;
P(Mutex) ;
cpt++ ;
if (cpt <= M)
        premier = true ;
if (cpt == M)
        V(SC) ;
}
V(Mutex) ;
if (premier)
        P(ST2)
}


Coordinateur() {
        V(ST)
        P(SC) ;
        for (i = 0 ; i < M ; i++) {
                V(ST2)
}

Initialisation :

ST = CS(0) ; SC = CS (0) ;
for (i=0;i<M;i++)
  ST2[i] = CS(0);
cpt = 0 ;
```

4. FICHIERS FAT (4 POINTS)

On considère la gestion de la FAT vue en TME dont le format est le suivant :



Le pointeur short* pt_FAT pointe vers le début de la FAT et le pointeur struct ent_dir* pt_DIR pointe vers le début du répertoire. La taille des blocs (secteurs) est de 128 octets. Chaque entrée du répertoire possède les champs suivants :

- **del_flag** : 0 indique que l'entrée est libre ; 1 indique que l'entrée est occupée.
- **name** : nom du fichier
- **first_bloc, last bloc**: premier et dernier bloc du fichier.
- **size** : taille du fichier.

Les entrées 15 à 127 de la table pt_FAT contiennent 0.

La fonction **int write_DIR_FAT_sectors ( )** permet d'écrire sur le disque les secteurs contenant la FAT et le répertoire.

4.1. (1 point)

- Quels sont les blocs composant les fichiers « fichier1 » et « fichier2 » ?
- Quels sont les blocs libres ?

fichier1 : 14 5
fichier2 : 4 9 7 8
Libres : 6, 10,11,12,13, 15 à 127

Nous voulons écrire une fonction *int FlipFirstSecond(char *fich)* qui remplace le premier bloc de *fich* par le deuxième et vice versa, à condition que le deuxième bloc existe et qu'il soit complètement rempli.

La fonction doit renvoyer -1 en cas d'erreur : si *fich* n'existe pas ou le deuxième bloc n'existe pas ou n'est pas complètement rempli. Dans tous les autres cas, elle renvoie 0.

4.2. (0,5 point)

En considérant le répertoire et la FAT de la figure ci-dessus, donnez la nouvelle configuration des entrées du répertoire et de la FAT à la suite aux appels à *FlipFirstSecond ("fichier1")* et *FlipFirstSecond ("fichier2").*

fichier1 : 14, 5 (le bloc 5 n'est complet)
fichier2: 9 4 7 8

pt_DIR[3].first_bloc = 9
pt_FAT[9] = 4
pt_FAT[4]=7

4.3. (2,5 points)

Donnez le programme C de la fonction *int FlipFirstSecond (char *fich)*.
NB. *int strcmp (char* ch1, char* ch2)* permet de comparer deux chaînes en retournant 0 si les deux chaînes sont égales.

```
int FlipFirstSecond (char *fich) {
 struct ent_dir * pt = pt_DIR;
   short blk ;
   for (i=0; i< NB_DIR; i++) {
     if ((pt->del_flag) && (!strcmp (pt->name, fich))) {
        /* fichier existe */
        if ( pt->size <2*SIZE_SECTOR)
          return -1 ;
     /* remplacer block 1 par block 2 */
     blk = *(pt_FAT + pt->first_bloc);
     *(pt_FAT + pt->first_bloc) = *(pt_FAT+blk) ;
     *(pt_FAT+blk) = pt->fist_bloc ;
    if ( pt->size == 2*SIZE_SECTOR)
      pt->last_bloc = pt->first_bloc;

     pt->first_bloc = blk ;
     write_DIR_FAT_sectors ( ) ;
  }
    pt++;
} /* for */
  /* fichier n'existe pas */
 return -1;
}
```

5. FICHIERS UNIX (3,5 POINTS)

On considère un système de fichier Unix, dont les fichiers sont représentés par une inode contenant un index à 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).

Chaque bloc de contrôle permet de référencer au maximum 128 blocs (de contrôle ou de données).

On considère deux fichiers F1, F2 ayant les caractéristiques suivantes :

F1 20 blocs de données,

F2 267 (10+128+129) blocs de données.

5.1. (0,5 point)

Donnez le nombre d'indirections utilisées pour le stockage des fichiers F1 et F2

Pour F1, simple indirection.
Pour F2, double indirection.

5.2. (1,5 points)

Quel est le nombre total de bloc occupés par F1 et F2 en comptant les blocs de données et les blocs de contrôle ?

Pour F1, 21 blocs = 20 données, 1 contrôle
Pour F2, 271 blocs = 267 données + 1 bloc de contrôle d'index simple + 3 blocs de contrôle d'index double indirection (1 double + 2 simple).

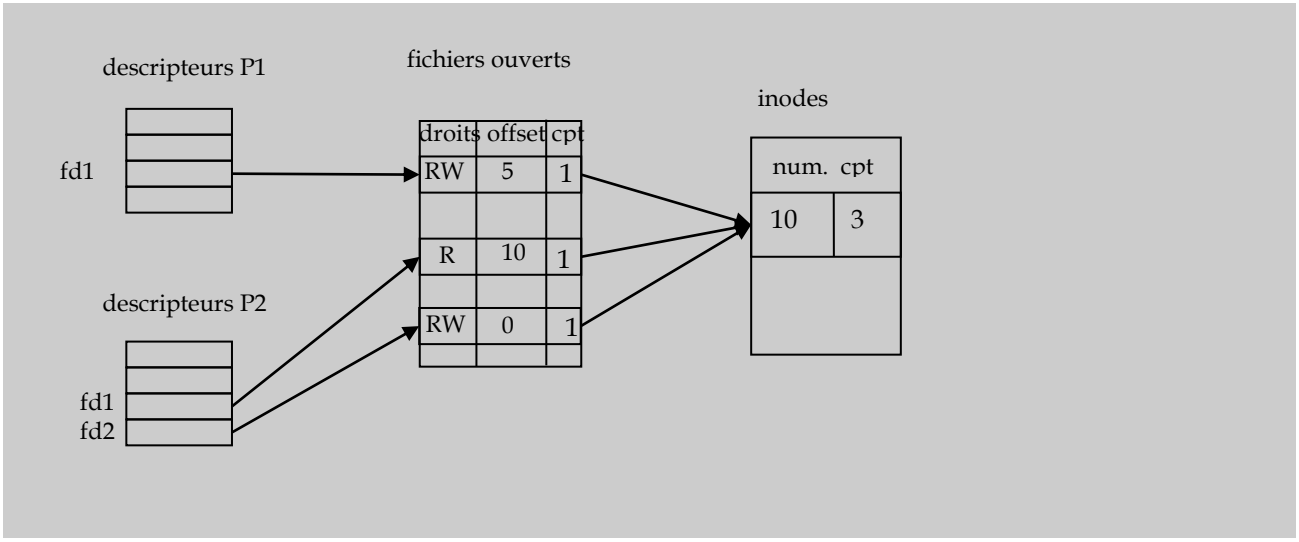
5.3. (1,5 points)

On considère un fichier "fic1", de taille 50 octets pour l'inode numéro 10. Deux processus P1 et P2 effectuent les opérations suivantes :

Processus P1 :	Processus P2 :
fd1 = open("fic1", O_RDWR) ;	fd1 = open("fic1", O_RDONLY) ;
read (fd1, buf, 5) ;	read (fd1, buf, 10) ;
	fd2 = open("fic1", O_RDWR);

où `buf` est un tableau de taille 10, et `read(int fd, void *buf, size_t count)` est un appel système qui lit au maximum `count` octets dans le fichier de descripteur `fd` et stocke les octets lus à l'adresse `buf`.

Représentez les structures de données en mémoire après ces opérations. Vous représentez la table des descripteurs de fichiers des processus, la table des fichiers ouverts (en indiquant le mode l'ouverture et le déplacement/offset), ainsi que la table des inodes.


descripteurs P1
fichiers ouverts
inodes
droits offset cpt
num. cpt
fd1
RW 5 1
10 3
R 10 1
descripteurs P2
RW 0 1
fd1
fd2