Examen 3I010

L3 – Licence d'Informatique -Année 2018

2 heures - Aucun document autorisé

Barème donné à titre indicatif

1. SYNCHRONISATIONS (7 POINTS)

Nous considérons un programme où le processus principal crée N processus fils P_0, P_1, P_2 …., P_{N-1}. Chaque processus fils $P_{id,}$ possède un identifiant *id* unique (0, …, N-1), affecté lors de sa création. $P_{id,}$ appelle alors la fonction *int calcul (int i)* en passant comme paramètre son *id* puis affiche la valeur renvoyée par la fonction *calcul* ainsi que la valeur de son identifiant *id*. Après il se termine. Ce que fait la fonction *calcul* n'est pas important pour l'exercice mais sa valeur de retour et sa durée varient selon le paramètre d'entrée.

1.1. (1 point)

Expliquez pourquoi les N affichages ne seront pas forcement faits dans l'ordre croissant des identifiants de processus.

- On ne peut pas dire quel processus fils aura en premier la CPU, si tous les processus fils se trouvent dans l'état prêt. De plus, si c'était le cas, comme la durée de *calcul* varie, il se peut que le processus fils en exécution perde la main (fin quantum) et un autre, dans la durée de *calcul* est petit, prend la main et fasse l'affichage.

1.2. (3 points)

Donnez le code en C d'une version de ce programme qui garantit que les affichages seront faits dans **l'ordre croissant des identifiants des processus,** autrement dit, dans l'ordre P_0, P_1, P_2 …., P_{N-1}. Notez que seuls les affichages doivent se faire dans l'ordre, l'appel à la fonction calcul par les processus fils peut se faire dans n'importe quel ordre.

Votre programme devra créer les processus et gérer les synchronisations en utilisant des sémaphores et/ou variables partagées (pas d'attente active, sleep ( ), etc.). Vous indiquerez l'initialisation des sémaphores et/ou variable partagées utilisés. Vous pouvez considérer N comme une constante.
Les fonctions pour manipuler des sémaphores sont rappélées en annexe.

```
#define N 5
SEM* sem [N];

int main (int argc, char* argv[ ]) {
 int i, p, v;
  for (i=1;  i < N; i++)
    sem [i] = C([0);
```

```
 sem [0] = CS(1);

  for (i=0; i < N; i++ && (p=fork ()) !=0 )
        ;
   if ( p== 0 )  {
       v = calcul(i);
       P(sem[i]);
       printf ("id: %d, valeur:%d \n", i, v);
       V(sem[i+1]);
}
```

Barème
Boucle fork : 20 %
Definition des sémaphores + valeurs initiales : 20 %
Synchronisation 50 %
Affichage correct : 10%
Si calcul synchronisé : – 10 %

Nous voulons modifier le programme ci-dessus en considérant un processus fils de plus P_N. Comme dans la question précédente, les processus P_0, P_1, …, P_{N-1} appellent la fonction *int calcul (int i)* en passant comme paramètre leur *id* respectif. Cependant, ils ne font pas l'affichage qui sera fait par P_N. Par conséquent, le processus P_N, qui n'appelle pas la fonction *calcul,* est dédié à l'affichage qui doit toujours être fait dans l'ordre des créations des autres processus fils (P_0, P_1, puis P_2 …., P_{N-1}).
Pour assurer une telle synchronisation des affichages, votre programme ne peut utiliser que des sémaphores et, si nécessaire, des variables partagées.

1.3. (3 points)

Donnez le code en C du programme demandé en spécifiant l'initialisation des sémaphores et/ou variables partagées utilisés..

```
#define N 5
SEM* sem [N+1]; /* un sémaphore pour le processus affichage */
shared x;

int main (int argc, char* argv[ ]) {
 int i, j, p, v;
  for (i=1;  i < N+1; i++)
    sem [i] = CS(0);
 sem [0] = CS(1);

  for (i=0; i < N; i++ && (p=fork ()) !=0 )
       ;

   if ( p== 0 )  {
     /* processus calcul */
     if (i  < N) {
```

```
        v= calcul(i);
        P(sem[i]);
        x=v;
       V(sem[N]); /*révéiller le processus affichage */
        exit (1);
     }
      /* processus affichage */
         for (j=0; j< N; j++ ) {
            P(sem[N]);
           printf ("id: %d, valeur:%d \n", j, x);
           if (j !=N)
            V(sem[j+1]);
         }
    }
}
```

Barème
Boucle fork : 20 %
Definition des sémaphores + valeurs initiales : 20 %
Synchronisation 50 %
Affichage correct : 10%
Si calcul synchronisé : – 10 %

2. MÉMOIRE (7 POINTS)

On dispose d'une machine simplement paginée avec des pages de 1024 mots. Soit un processus P. Son code commence à l'adresse 1000 et se termine à l'adresse 3000 mots. Sa pile est située entre les adresses 10000 et 15000. La zone contenant les données est située entre les adresses 3100 et 5000.

2.1. (1 point)

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

Code : pages 0, 1 et 2 (1000 div 1024 = 0, 3000 div 1024 = 2)
pile : pages entre 9 et 14
données: pages 3 et 4

Barème : 33% code, 33% pile, 33% données

2.2. (0,5 point)

Les variables *x* et *y* situées respectivement aux adresses 12000 et 3500 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.

Barème : 50 % par variable

Initialement **la première page des données est chargée dans la case 300**.

2.3. (0,5 point)

Faites un schéma de la table des pages du processus P en précisant les cases, les droits et le bit de présence.

TP
<page,case,P,droits>
<0,-,0,LX>
<1,-,0,LX>
<2,-,0,LX>
<3,300,1,LE>
<4,-,0,LE>
<9,-,0,LE>
…
<14,-,0,LE>

Barème : appréciation libre

2.4. (1, 5 points)

Que se passe-t-il lors :
a) d'un accès en écriture à l'adresse 2018 ?
b) d'un accès en écriture à l'adresse 3150?
c) d'un accès en lecture à l'adresse 10 050?

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 P
a) MMU : accès page <3,78>, adresse physique = 300 * 1024 + 78 = 307 278
b) MMU : accès page <9,834> , IT défaut de page , Système : traite défaut de la page 9

Barème : 33% par cas

Le système réserve uniquement pour les pages de code les cases 100 et 200. Pour la pile et les données, les cases 300, 400 et 500 sont réservées.

On applique pour les pages de code un algorithme de remplacement de pages de type **FIFO** et pour les pages de piles et de données un algorithme **LRU** (Least Recently Used).

On considère dans la suite qu'**aucune page n'est initialement chargée en mémoire**.

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

0 9 1 3 2 10 0 9 1 4 0 11 2 11

2.5. (3,5 points)

a) Quels sont les défauts de page engendrés par cette séquence ? Faites un tableau pour indiquer à quels moments ces défauts ont lieu.
b) Quel est le nombre total de défauts de pages ?
c) Donnez l'état de la table des pages à la fin de la séquence. Pour chaque page, vous indiquerez: la case éventuelle, les bits de présence, les droits.

pages 0, 1 et 2 en FIFO
les autres pages en LRU

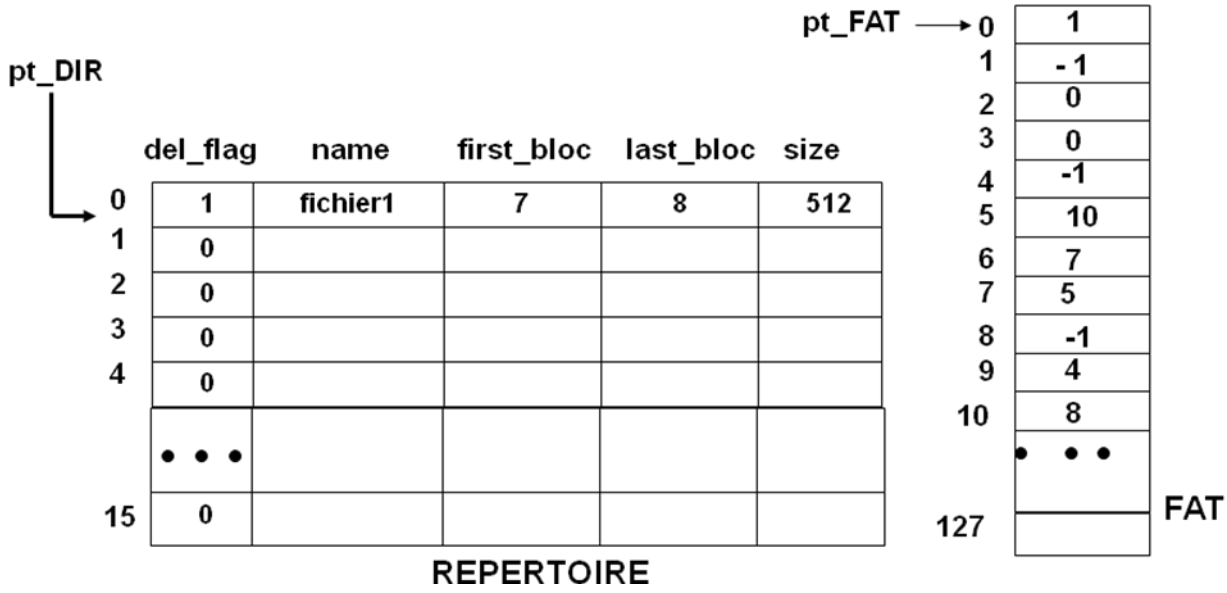
	0	9	1	3	2	10	0	9	1	4	0	11	2	11
0	100				X		200						X	
1			200				X		100					
2					100				X				200	
3				400						X				
4										400				
9		300						300						
10						500						X		
11												500		500
Déf	D	D	D	D	D	D	D		D	D		D	D	

11 défauts
TP <page,case,P, droits>
<0,-,0,LX>
<1,100,1,LX>
<2,200,1,LX>
<3,-,0,LE>
<4,400,1,LE>
<9,300,1LE>
<10,-,0,LE>
<11,500,1,LE>

Barème :
a) 65 %
b) 15 %
c) 20 %

3. FICHIERS (5 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. **Les blocs ont une taille de 128 octets.**

3.1. (1 point)

- Quels sont les blocs composant le fichier « fichier1 »?
- Quels sont les blocs libres

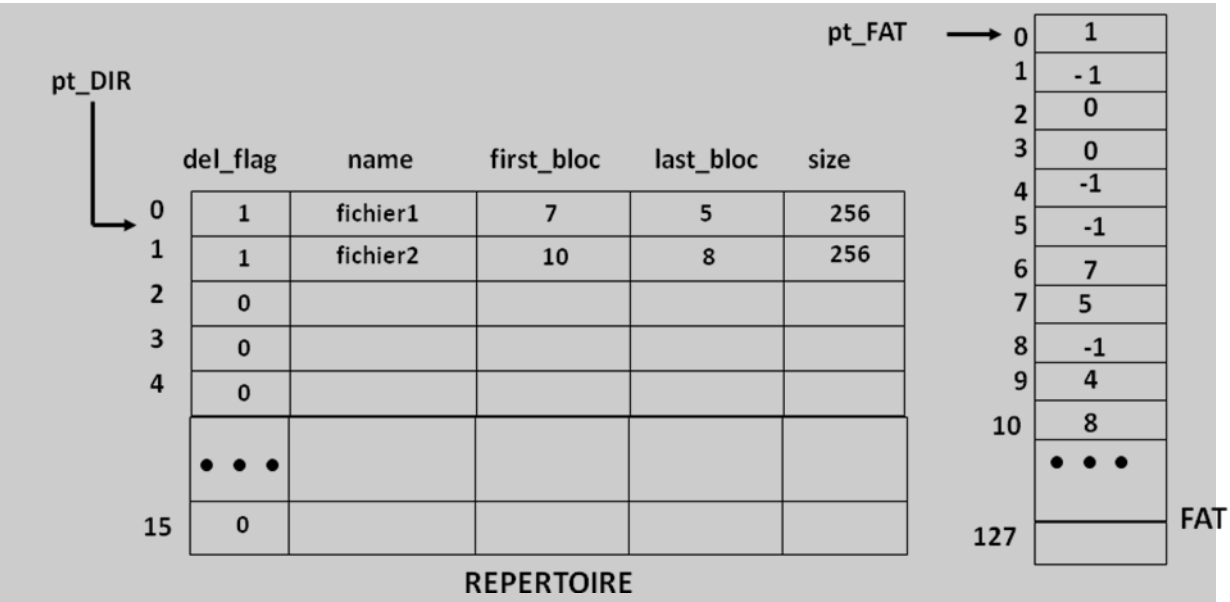
Fichier1 : 7,5,10,8
Libres : 2,3

Barème :
50 % blocs fichier
50% blocs libres

On veut écrire une fonction int SplitFile(char *fich1, char *fich2) qui permet de découper le fichier fich1 en deux de la façon suivante : le fichier fich1 possède les n/2 premiers blocs (n étant le nombre de blocs initial de fich1) et le nouveau fichier fich2 possède les n/2 blocs suivants. Pour simplifier on fera l'hypothèse que le nombre de blocs d'un fichier est pair et supérieur à 1.

3.2. (1 point)

Faites un schéma du répertoire suite à l'appel à SplitFile(«fichier1 », « fichier2 »).



Barème
45% fichier 1
45 % fichier 2

3.3. (3 points)

Ecrire la fonction int SplitFile. La fonction retourne 1 si fich1 n'existe pas ou s'il n'y a plus suffisamment d'entrées libres dans le répertoire pour créer le fichier fich2, sinon la fonction retourne 0.

```
int SplitFile(char *fich1, char *fich2) {
 int i;

 short cpt = 0, bloc, nb;
 int i=0
 struct ent_dir * pt1 = pt_DIR;
 struct ent_dir * pt2 = pt_DIR;

/* rechercher l'entrée de fichier 1*/
 while (i < NB_DIR && pt1->del_flag != 0 && !strcmp(pt1->name,fich1)) {
    i++;
   pt1++;
 }
 if (i == NB_DIR)
  return 1; /* Fichier inexistant */
/* rechercher une entrée libre dans le repertoire */
 i = 0 ;
 while (i < NB_DIR && pt2->del_flag != 0) {
    i++;
   pt2++;
 }

 if (i == NB_DIR)
   return 1;  /* Plus de place dans le repertoire */
```

```
 /* rechercher le bloc du milieu */
  nb = pt1->size/128;
  bloc = pt1->first_bloc;
  i=0;
  while (i <nb/2) {
       bloc = *(pt_FAT + bloc);
       i++;
  }
   /* mettre à jour les 2 entrées du repertoire */
  pt2->del_flag=1;
  strcpy(pt2->name, fich2) ;
  pt2->first_bloc = *(pt_FAT + bloc);
  pt2->last_bloc = pt1->last_bloc;
  pt2->size = p1->size/2;
  pt1->size /=2 ;
  pt1->last_bloc = bloc;
 *(pt_FAT + bloc) = -1;
  return 0;
}
```

Barème :
Recherche entrée fich1 : 15 %
Recherche entrée libre pour fich2 : 15 %
Recherche bloc milieu : 30 %
Mise à jour entre fich1 , fich2 : 40%
-10 % si par de retour erreur

ANNEXE

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

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

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