Examen 3I010

L3 – Licence d'Informatique -Année 2016

Tout document papier autorisé, barème donné à titre indicatif

I. SYNCHRONISATION (7,5 POINTS)

Nous considérons deux types de processus : 1 **producteur** et M **consommateurs**. Les processus producteur et consommateurs sont cycliques, autrement dit, chaque processus est **une boucle infinie**. Ils ont accès au vecteur partagé **int vet [N],** avec M<N.

En appelant la fonction *int random ( )*, le producteur génère des valeurs aléatoire en les stockant dans **vet**. Les consommateurs ne peuvent commencer à consommer les valeurs stockées dans **vet** que lorsque celui-ci est **plein.** Chaque valeur doit alors n'être consommée qu'une seule fois par un des M consommateurs qui l'affichent. Lorsque toutes les N valeurs ont été consommées, le producteur peut produire N nouvelles valeurs en les stockant dans **vet.**

Observation : Tous les consommateurs doivent avoir la même chance de consommer les valeurs, vous ne pouvez donc pas choisir quels sont les consommateurs qui iront consommer les valeurs.

I.1. (4,5 points)

Donnez le pseudo-code des processus *Producteur ( )* et *Consommateur (int id),* où *id* (0≤ id <M) est l'identifiant du processus consommateur. Indiquez les sémaphores et variables partagées utilisés dans votre solution ainsi que leur initialisation.

```
Semaphores :
  init (PROD,1) ;
  init (CONS,0) ;
  init (MUTEX,1) ;

variables partagées
int vet [N] ;
int nb_cons ;

Producteur ( )
{   int i ;
        while (1) {
            P(PROD);
            for (i=0 ; i <N ; i++)
                vet[i]= random ( );
            /* libérer N lectures */
            for (i=0; i<N; i++)
              V(CONS);
        }
```

```
}


Consommateur (int id)
{   int val ;

   while (1) {
     P(CONS) ;
     P(MUTEX) ;
     val=vet[nb_cons] ;
     nb_cons++ ;

     if (nb_cons== N) {
     /* réveiller le producteur */
       nb_cons=0;
        V(PROD);
     }
    V(MUTEX);
   printf ("%d",val);
  }
```

Supposons maintenant que N est divisible par M est que chaque processus doit consommer exactement N/M valeurs chaque fois que le vecteur **vet** est rempli.

I.2. (3 points)

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

```
Semaphores :
 init (PROD,1) ;
 SEM CONS [M] ;
  for (i=0; i<M; i++)
        init (CONS[i],0);
 init (MUTEX,1) ;

variables partagées
int vet [N]
int nb_cons ;

Producteur ( )
{   int i,j ;
        while (1) {
            P(PROD);
            for (i=0 ; i <N ; i++)
               vet[i]= random ( );
            /* libérer M/N lecture par consommateur */
            for (i=0; i<M; i++)
```

```
            for (j=0; j<N/M; j++)
              V(CONS [i]);
      }
}

 Consommateur (int id)
{   int val ;

   while (1) {
     P(CONS [i]) ;
     P(MUTEX) ;
     val=vet[nb_cons] ;
     nb_cons++ ;

     if (nb_cons== N) {
     /* réveiller le producteur */
       nb_cons=0;
        V(PROD);
     }
    V(MUTEX);
   printf ("%d",val);
  }
```

II. MEMOIRE (6,5 POINTS)

On dispose d'une machine monoprocesseur simplement paginée avec des pages de 512 mots. On considère le programme suivant :

Adresse	Instruction	Signification
C	`Loada X`	RA = X
	`Loadb Y`	RB = Y
	`Storeb X`	X = RB
	`Storea Y`	Y = RA

Où:

- RA et RB sont des registres internes,
- C est l'adresse virtuelle du début du programme. C = 511
- X et Y sont deux variables situées respectivement aux adresses virtuelles 2570 et 5200

II.1. (1 point)

Quels sont les numéros des pages associées au code et les numéros de pages associées aux variables X et Y?

Code : 0 et 1, Variables 5 et 10

II.2. (1, 5 points)

Donnez la séquence (dans l'ordre) des pages accédées par un processus exécutant ce programme.

0, 5, 1, 10, 1, 5, 1, 10

II.3. (2 points)

On dispose de 3 cases libres, 100, 200 et 300. On suppose qu'une fois en mémoire les pages de codes sont "clouées", c'est-à-dire qu'elles ne peuvent pas être choisies comme victimes.

Indiquez le nombre de défauts de pages engendré par la séquence de la question II.2 en appliquant un remplacement de page FIFO. Donnez l'état de la table des pages du processus à la fin de cette séquence. Pour chaque page, vous indiquerez, le bit de présence P, le numéro de case, les droits et le bit de modification M.

Seq.	0	5	1	10	1	5	1	10
0	100							
1			300					
5		200		X		200		X
10				200		X		200
Def.	D	D	D	D		D		D

6 défauts

Tables de pages :
Page, P , case, droits, M
0,1,100, RX,0
1,1,300, RX ,0
5,0, -, RW, 1
10,1,200,RW, 1

II.4. (1 point)

A la fin de cette séquence connaît les adresses physiques de C, X et Y. Si oui donnez leur adresse physique en détaillant la formule permettant de l'obtenir.

C : adresse physique = 100*512+511 = 51 711
X : adresse physique inconnue
Y : adresse physique = 200*512 + 5200%512 = 102 480

II.5. (1 point)

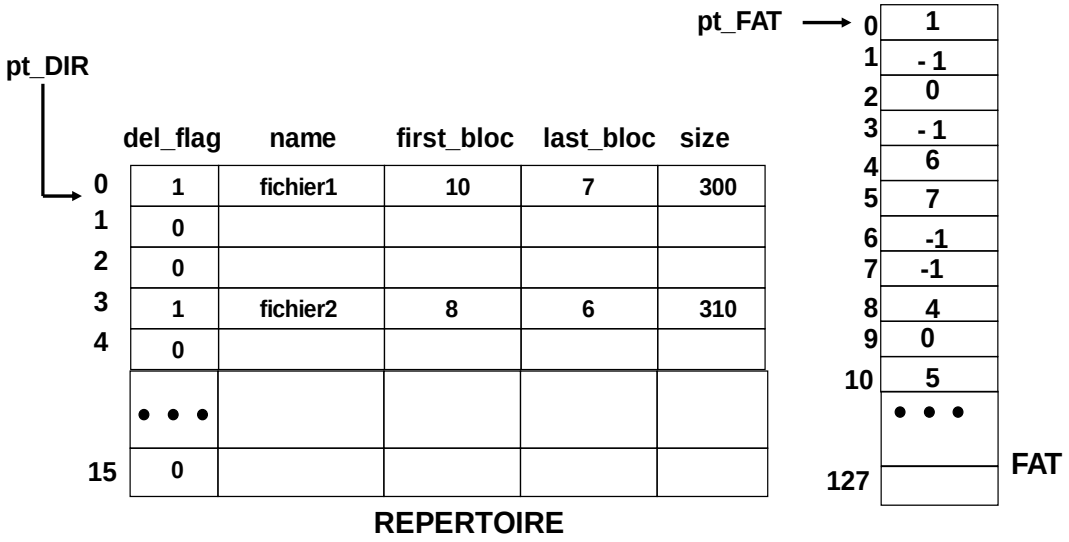
Dans cette configuration, toujours en considérant des pages de code clouées en mémoire, obtiendra-t-on le même nombre de défauts de pages et la même table des

pages si on appliquait une autre stratégie de remplacement de pages (par exemple LRU) ? Justifiez.

Même séquence, car il ne reste qu'une seule case libre pour les variables. Il n'y a pas de choix dans la victime

III. FICHIERS (6 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 (**14 octets**) 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**: premier bloc du fichier.
- **size** : taille du fichier.

III.1. (1 point)

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

Fichier1 : 10, 5,7
Fichier2 : 8,4,6
Libres : 2,9

III.2. (2,5 points)

Ecrire une fonction *int PhysBlock(char *fich, int n)* qui retourne le numéro de bloc physique correspondant au bloc numéro *n* du fichier *fich*. Si ce bloc n'existe pas, la fonction doit retourner -1. Par exemple si un fichier "toto" est composé dans l'ordre des blocs 30, 12, 15, 20, PhysBlock("toto", 2) retournera 15 et PhysBlock("toto", 4) retournera -1.

```
int PhysBlock(char *fichier, int n) {
  int i , cont =0;

  short bloc, cpt = 0;
  struct ent_dir * pt = pt_DIR;
  for (i=0; i< NB_DIR; i++) {
    if (pt->del_flag && !strcmp(pt->name, fichier)) {
      if (pt->size / 128 <= n)
         return -1;
      bloc = pt->first_bloc;
      while (cpt <n) {
        bloc = *(pt_FAT + bloc);
        cpt++;
      }
      return bloc;
    }
    pt++;
  }
}
```

III.3. (2,5 points)

Ecrire une fonction *void ReversePrint(char *fich)* qui utilise la fonction *PhysBlock* pour afficher la liste des numéros de bloc du fichier *fich* dans **l'ordre inverse**. Par exemple pour le fichier "toto" composé dans l'ordre des blocs 30, 12, 15, 20, ReversePrint("toto") affichera "20 15 12 30". On fera l'hypothèse que le fichier *fich* existe.

```
void ReversePrint(char *fich) {
   int i;
  for (i=0; i< NB_DIR; i++) {
    if ((pt->del_flag) && !strcmp (pt->name,fich)) {
        if (pt->size % 128 == 0)
             n = pt>size/128 -1;
        else n = pt>size/128;
        while (n>=0) {
             printf("%d ", PhysBlock(fich,n))
            n--;
        }
}
```