

Examen 3I010

L3 – Licence d’Informatique -Année 2019

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

1. SYNCHRONISATIONS PAR SEMAPHORES (6 POINTS)

On veut synchroniser un calcul réparti. Tous les processus sont cycliques et bouclent indéfiniment.

N processus de calcul (C_0, C_1, \dots, C_{N-1}) appellent une fonction de calcul local qui retourne un entier.

Toutes les valeurs produites sont accumulées dans une variable partagée *part* (*part* somme les valeurs produites par les C_i). Lorsque tous les processus C_i ont ajouté leur valeur à *part*, un processus X est réveillé.

Une fois réveillé X fait un calcul global sur *part* et réinitialise *part* à 0.

Les processus C_i ne peuvent continuer que lorsque le calcul global est terminé.

Le pseudo-code des processus est le suivant

```
#define N 10
void C(int i) {
    int c ;
    while(1) {
        ...
        c = calcul_local() ;
        ...
        part +=c;
        ...
    }
}
void X() {
    while (1) {
        ...
        calcul_global(part) ;
        part = 0;
        ...
    }
}
```

1.1. (3 points)

Complétez les codes de fonctions C et X.

Spécifiez les sémaphores et/ ou variables partagées utilisés ainsi que leur initialisation.

Un compteur partagé cpt initialize à 0

Un tableau de sémaphores de taille N, initialisé à 1 pour bloquer les Ci

Un sémaphore SX initialisé à 0 pour bloquer X

Deux sémaphores MUTEX1 et MUTEX2 initialisé à 1

```
void C(int i) {
    int c ;
    while(1) {
        P(SC[i]);
        c = calcul_local() ;
        P(MUTEX1);
        part +=c;
        V(MUTEX1);
        P(MUTEX2);
        cpt++;
        if (cpt==N)
            V(SX);
        V(MUTEX2);
    }
}

void X(){
    int i;
    while (1) {
        P(SX);
        /* Mutexs inutiles ici car X est forcément seul ici */
        calcul_global(part) ;
        part = 0;
        cpt = 0;
        for (i=0 ; i<N ; i++)
            V(SC[i]) ;
    }
}
```

1.2. (3 points)

On souhaite modifier la synchronisation pour que X soit réveillé lorsque les K premiers ($K < N$) calculs locaux ont été effectués. Les valeurs produites par ces K calculs locaux sont accumulées dans la variable partagée *part*. Comme précédemment, tous les processus C_i doivent attendre que tous aient fini leur calcul local avant de continuer.

Modifiez vos programmes en conséquence.

Ajout d'un sémaphore Attente initialisé à 0.

```

void C(int i) {
    int c ;
    while(1) {
        P(SC[i]);
        c = calcul_local() ;
        P(MUTEX1);
        cpt++ ;
        if (cpt <= K) {
            part +=c;
            if (cpt == K)
                V(SC);
        }
        else {
            if (cpt == N) {
                for (c=0 ; c<N-1 ; c++)
                    V(SC[c]) ;
                cpt = 0;
                part = 0;
                V(Attente);
            }
        }
        V(MUTEX1);
    }
}

void X(){
    while (1) {
        P(SX);
        P(MUTEX1);
        calcul_global(part) ;
        V(MUTEX1);
        P(Attente);
    }
}

```

2. MEMOIRE (4,5 POINTS)

On dispose d'une machine simplement paginée avec des pages de 1024 octets.

2.1. (0,5 point)

La traduction des adresses virtuelles en adresses physiques est-elle réalisée par le processeur ou par le système d'exploitation ? Justifiez votre réponse.

Traduction est faite par la MMU pour des raisons d'efficacité

2.2. (0,5 point)

La table des pages d'un processus est-elle stockée en RAM ? A quel moment la table des pages d'un processus est créée ?

La table est stockée en RAM et est créée à la création du processus (fork).

2.3. (0,5 point)

La TLB est-elle stockée en RAM ? Justifiez.

Non, c'est une mémoire interne de la MMU, utilisée pour éviter les accès RAM lors de la traduction.

Soit un processus P, dont la table des pages est la suivante :

| case | Présence | LEX | Référence |
|------|----------|-----|-----------|
| 0 | 0 | 101 | 0 |
| 1 | 200 | 101 | 0 |
| 2 | 0 | 110 | 0 |
| 3 | 0 | 110 | 0 |

(L : lecteur, E : écriture, X : eXécution)

On dispose d'une seule case libre, la case 300.

2.4. (2 points)

Que se passe-t-il lorsque P fait :

a) un accès en lecture à l'adresse 2019 ?

b) un accès en écriture à l'adresse 3000 ?

Vous préciserez les actions faites par le matériel et celles faites par le système.

Vous donnerez l'adresse physique correspondant à l'accès.

a) MMU : accès page <1,995>, droit OK, Présence OK, adresse physique = $200 \cdot 1024 + 995 = 205795$

b) MMU : accès page <2,952>, IT défaut de page , Système : traite défaut de la page 2, alloue la case 300, adresse physique = $300 \cdot 1024 + 952 = 308152$

2.5. (1 point)

Donnez le nouvel état de la table des pages de P après ces 2 accès.

| | case | Présence | LEX | R |
|---|------|----------|-----|---|
| 0 | | 0 | 101 | 0 |
| 1 | 200 | 1 | 101 | 1 |
| 2 | 300 | 1 | 110 | 1 |

3. REMPLACEMENT DE PAGES (4,5 POINTS)

On dispose d'une machine simplement paginée. Soit deux processus P1 et P2 n'exécutant pas le même code.

P1 fait les accès suivant à ses pages :

0 1 2 2 1 0 1 2 0

P2 fait les accès suivant à ses pages :

1 2 3 3 4 1 2 1 4

On dispose en tout de **6** cases libres pour exécuter P1 et P2.

P1 et P2 s'exécutent **en temps partagé** suivant l'algorithme du tourniquet simple avec **une commutation toutes les 3 références** (on considère qu'initialement P1 est élu). On considère dans la suite qu'**'aucune page n'est initialement chargée en mémoire**.

On fait une stratégie de remplacement de page de type FIFO appliqué localement à chaque processus : le système alloue 3 cases à P1 (les cases 10, 20 et 30) et trois cases à P2 (les cases 40, 50 et 60). Le remplacement de page d'un processus ne se fait qu'au sein des pages du processus : par exemple si P1 a besoin d'une nouvelle page alors qu'il a déjà utilisé ses 3 cases, on choisira parmi les pages de P1 la plus anciennement chargée.

3.1. (3 points)

a) Quels sont les défauts de page engendrés par cette séquence ? Faites un tableau pour indiquer à quels moments ces défauts ont lieu et la case utilisée lors de chacun de ces défauts.

b) Quel est le nombre total de défauts de pages ?

| | | P1 | | | P2 | | | P1 | | | P2 | | | P1 | | | P2 | | |
|---------|---|----|----|----|----|----|----|----|---|---|----|----|----|----|---|---|----|---|---|
| | | 0 | 1 | 2 | 1 | 2 | 3 | 2 | 1 | 0 | 3 | 4 | 1 | 1 | 2 | 0 | 2 | 1 | 4 |
| P1 | 0 | 10 | | | | | | | | | | | | | | | | | |
| | 1 | | 20 | | | | | | | | | | | | | | | | |
| | 2 | | | 30 | | | | | | | | | | | | | | | |
| P2 | 1 | | | | 40 | | | | | | | X | 50 | | | | | | |
| | 2 | | | | | 50 | | | | | | X | | | | | 60 | | |
| | 3 | | | | | | 60 | | | | | | | | | | X | | |
| | 4 | | | | | | | | | | | 40 | | | | | | | |
| Défauts | D | D | D | D | D | D | | | | | D | D | | | | | D | | |

9 défauts

3.2. (1,5 points)

Donnez l'état des tables des pages à la fin de ses séquences. Pour chaque page, vous indiquerez : la case éventuelle, les bits de présence.

TP de P1

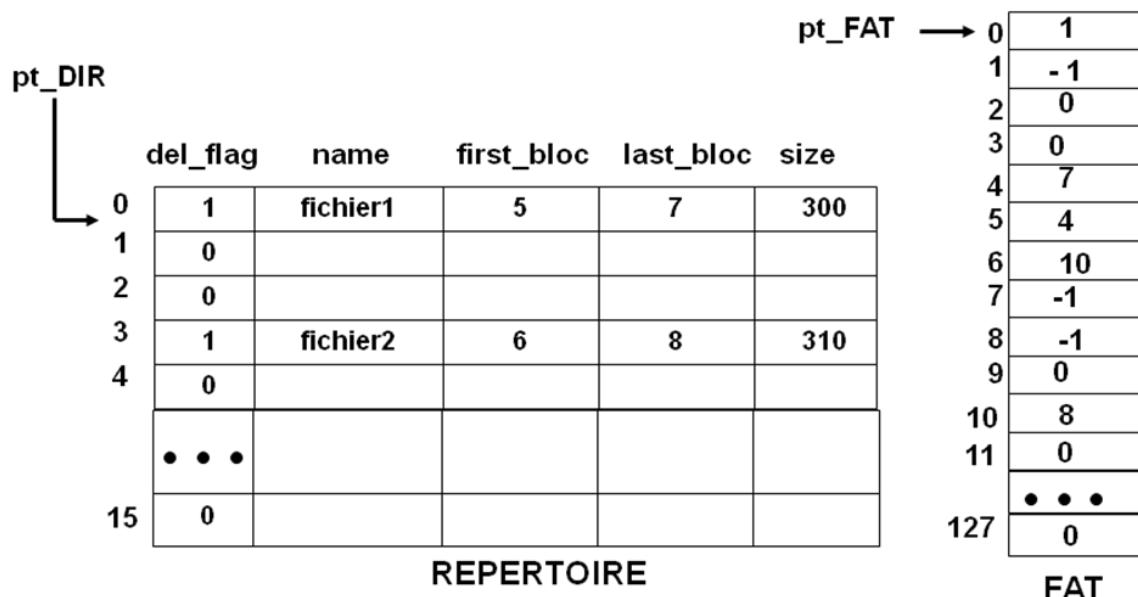
```
<page,case,P>
<0,10,1>
<1,20,1>
<2,30,1>
```

TP de P2

```
<page,case,P>
<1,50,1>
<2,60,1>
<3,-,0>
<4,40,1>
```

4. 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. 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 11 à 127 de la table pt_FAT contiennent 0.

4.1. (1 point)

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

fichier1 : 5 4 7

fichier2 : 6 10 8

Libres : 2, 3, 9, 11 à 127

On veut écrire une fonction **int fusion_file(char *f1, char *f2)** qui fusionne les blocs des fichiers f1 et f2. Après l'appel à la fonction, le fichier f1 sera composé des blocs de f1 et f2 de façon alternative ; le fichier f2 est lui détruit. La fonction retourne 0 si la fusion s'est faite correctement et -1 sinon. Pour simplifier, on suppose que f1 et f2 ont **le même nombre de blocs** et que la taille des fichiers est un multiple de 128.

Par exemple, si avant l'appel, f1 est composé des blocs 23, 15 et 17 et f2 des blocs 30, 40 et 35, après l'appel f1 sera composé, dans cet ordre, des blocs 23, 30, 15, 40, 17 et 35.

4.2. (4 points)

Donnez le programme C de la fonction fusion_file.

```
int fusion_file(char *f1, char *f2){
    int i, auxb1, auxb2, b1, b2;
    struct ent_dir *pt = pt_DIR, *pt1, *pt2;

    pt1 = pt2 = NULL;
    for (i=0; i< NB_DIR; i++) {
        if ((pt->del_flag) && (!strcmp (pt->name, f1)))
            pt1 = pt;
        else
            if ((pt->del_flag) && (!strcmp (pt->name, f2)))
                pt2 = pt;
        pt++;
    }

    if (!pt1 || !pt2) {
        return -1;
    }
```

```
b1 = pt1->first_bloc;
b2 = pt2->first_bloc;

while (b1 != -1 && b2 != -1) {
    auxb1 = pt_FAT[b1];
    auxb2 = pt_FAT[b2];
    pt_FAT[b1] = b2;
    pt_FAT[b2] = auxb1;
    b2 = auxb2;
    b1 = auxb1;
}
pt1->size += pt2->size;
p1->last_bloc = p2->last_bloc;
pt2->del_flag = 0;
return 0 ;
}
```