Contrôle terminal Session 1 - 5/1/2021

Durée : 2h
Cours et/ou travaux dirigés autorisés.

Barème donné à titre indicatif : Ex.1 : 3 - Ex. 2 : 5 - Ex. 3 : 7 - Ex 4 : 5

Exercice 1. Le système de fichiers `FAT` a été introduit dans le passé dans le système MSDOS mais reste utilisé dans divers périphériques multimédia actuels. Supposons un système FAT-32 (28 bits d'adressage de blocs disque). Supposons que la taille d'un bloc disque est de 32 Ko.

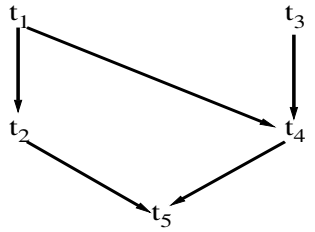
a) Calculer la taille maximale d'une partition (on néglige l'espace disque réservé sur la partition par le système d'exploitation). (1,5 pt)
Taille maximale d'une partition = Nb de blocs × taille_bloc = $2^{28} \times 32$ Ko = $2^{28} \times 2^5 \times 2^{10} = 2^{43}$ octets = 2^{13} Go.

b) Pour accélérer l'accès à la table `FAT`, le système la charge en MC. Si on crée une partition `FAT` de 32 Go, calculer la taille occupée en mémoire par la `FAT`. (1,5 pt)
1 bloc occupe 32Ko, on a une partition de 32 Go ⇒ au total, on a $\frac{32 \times 2^{30}}{32 \times 2^{10}} = 2^{20}$ blocs.
L'adresse est sur 28 bits. Donc, la FAT est une table de 2^{20} lignes et de 28 colonnes, donc elle occupe $2^{20} \times 4$ octets = 4 Mo.

Exercice 2. On considère l'ensemble de tâches suivant qui calcule l'expression $(x + y + z)/(u * v * (x + y))$:

```
t1 : a = x + y
t2 : b = a + z
t3 : c = u * v
t4 : d = c * a
t5 : e = b / d
```

1. Construire le graphe de précédence correspondant à cet ensemble de tâches. (2 pt)



2. Proposer un programme qui exécute cinq processus en parallèle, chacun d'eux exécutant une tâche t_i et qui respecte l'ordre induit par le graphe en utilisant trois sémaphores. (3 pt)

```
begin
init(S, 0) ;  init(U, 0) ; init(R, -1);
t1 ; V(S) ; V(S)
P(S) ; t2; V(R)
t3 ; V(U)
P(U); P(S); t4; V(R)
P(R); t5
end
```

Exercice 3. Soit un système composé de 3 processus cycliques `Acquisition, Exécution` et `Impression` et de 2 tampons `Requête` et `Avis` gérés circulairement, respectivement composés de m et n cases.

a) Le processus `Acquisition` enregistre chacune des requêtes de travail qui lui sont soumises par des clients puis il les place dans le tampon `Requête` à destination du processus `Exécution`. Il utilise la procédure `enregistrer_travail(mess:message)` pour enregistrer une requête de travail ;
b) Le processus `Exécution` exécute chaque requête de travail prélevée depuis le tampon `Requête` et transmet ensuite au processus `Impression` un ordre d'impression de résultats déposé dans le tampon `Avis`. Il utilise la procédure `exécuter_travail(mess:message)`.
c) Le processus `Impression` prélève les ordres d'impression déposés dans le tampon `Avis` et les exécute. Il utilise la procédure `imprimer_resultat(mess:message)`.

1. Programmer la synchronisation des trois processus en utilisant quatre sémaphores : `mvide, nvide, mplein` et `nplein` dont vous préciserez les valeurs d'initialisation et des variables nécessaires à la gestion des tampons. (0,5 +1 + 1 + 1 pt)

   On identifie un schéma producteur-consommateur sur chacun des tampons. On utilise pour chacun de ces schémas un couple de sémaphores :
   - `mvide` initialisé à m et `mplein` initialisé à 0 (tampon `Requête`) ;
   - `nvide` initialisé à n et `nplein` initialisé à 0 (tampon `Avis`).

```
Déclarations globales :
Requête : tampon [0..m-1] de messages ;
Avis : tampon [0..n-1] de messages ;
m, n : entier ;
mvide, nvide, mplein, nplein : sémaphore ;
init(mvide, m); init(nvide, n); init(mplein, 0); init(nplein, 0);
```

procédure Acquisition:	procédure Exécution :	procédure Impression:

mess:message;
i:index := 0
début
while (true) {
enregistrer_travail(mess);
P(mvide);
Requete(i) = mess ;
i = i + 1 % m ;
V(mplein);
}
fin
	mess, res : message;
	j, k : index :=0
	début
	while (true) {
	P(mplein) ;
	mess = Requete(j) ;
	j = j+1 % m ;
	V(mvide);
	exécuter_travail(mess, res) ;
	P(nvide) ;
	Avis(k) = res ;
	k = k+1 % n ;
	V(nplein);
	}
	fin
		mess:message;
		l:index := 0
		début
		while (true) {
		P(nplein) ;
		mess = avis(l) ;
		l = l+1 % n ;
		V(nvide);
		imprimer_resultat(mess)
		}

2. On étend le système à 3 processus `Acquisition`, 3 processus `Exécution` et 3 processus `Impression`. Complétez la synchronisation précédente pour que celle-ci demeure correcte en gérant les accès concurrents aux tampons `Requete` et `Avis` par des sémaphores de type verrou. (0,5 +1 + 1 + 1 pt)

   Les variables i, j, k et l sont maintenant globales et les accès à ces variables doivent se faire en exclusion mutuelle. On ajoute donc 4 sémaphores d'exclusion mutuelle initialisés à 1 (un sémaphore par index).

```
Déclarations globales :
Requête : tampon [0..m-1] de messages ;
Avis : tampon [0..n-1] de messages ;
m, n : entier ;
i, j, k, l : index sur les tampons ;
mvide, nvide, mplein, nplein, muti, mutj, mutk, mutl : sémaphores ;
début
```

```
init(mvide, m); init(nvide, n); init(mplein, 0); init(nplein, 0);
init(muti, 1); init(mutj, 1); init(mutk, 1); init(mutl, 1);
i = j = k = l = 0;
```

procédure Acquisition:	procédure Exécution:	procédure Impression :

mess:message;
début
while (true) {
enregistrer_travail(mess);
P(mvide);
P(muti);
Requete(i) = mess ;
i = i + 1 % m ;
V(muti);
V(mplein);
}
fin
	mess, res : message;
	début
	while (true) {
	P(mplein);
	P(mutj);
	mess = Requete(j) ;
	j = j+1 % m ;
	V(mutj);
	V(mvide);
	exécuter_travail(mess, res) ;
	P(nvide) ;
	P(mutk);
	Avis(k) = res ;
	k = k+1 % n ;
	V(mutk);
	V(nplein);
	}
	fin
		mess:message;
		début
		while (true) {
		P(mutl);
		P(nplein) ;
		mess = avis(l) ;
		l = l+1 % n ;
		V(nvide);
		V(mutl);
		imprimer_resultat(mess);
		}
		fin

Exercice 4. Soit le programme suivant :

```
x=20
i=0
n=0
tant que i < x faire
   si i mod 3 = 0 alors n = n+1 fsi
   i = i+1
ftq
écrire n
```

On pose les hypothèses suivantes :

- les variables `x, i` et `n` sont déclarées dans cet ordre et elles occupent des cellules mémoire consécutives.
- `x, i` et `n` sont rangées en mémoire à partir d'une adresse contenue dans le registre R_{30}.
- les entiers sont stockés sur 4 octets, et la mémoire est adressable en octets.
- la machine dispose de 32 registres, numérotés de 0 à 31.

Traduire le programme en assembleur. Vous devrez commenter soigneusement vos instructions [1]. (4 + 1 pt)

1. un commentaire est une ligne qui commence par ';'.

On rappelle ci-dessous les instructions utiles à la traduction :

Mnémonique	opération	commentaire
`MOV Rx, Ry`	`Rx` $\leftarrow$ `Ry`	`Rx` reçoit les 32 bits contenus dans `Ry`
`MVI Rx, #i`	`Rx` $\leftarrow$ `valeur i`	`Rx` reçoit sur 32 bits la `valeur i`
`STW (Rx), Ry`	`Mem[Rx]`$_{32}$ $\leftarrow$ `Ry`	les 32 bits de `Ry` sont stockés en mémoire à l'adresse contenue dans `Rx`
`LDW Rx, (Ry)`	`Rx` $\leftarrow$ `Mem[Ry]`$_{32}$	`Rx` reçoit les 32 bits mémoire dont l'adresse est dans `Ry`
`ADD Rx, Ry, Rz`	`Rx` $\leftarrow$ `Ry + Rz`	`Rx` reçoit la somme sur 32 bits des valeurs contenues dans `Ry` et `Rz`
`ADD Rx, Ry, #i`	`Rx` $\leftarrow$ `Ry + valeur i`	`Rx` reçoit la somme sur 32 bits de la valeur contenue dans `Ry` et de `i`
`SUB Rx, Ry, Rz`	`Rx` $\leftarrow$ `Ry - Rz`	`Rx` reçoit la différence sur 32 bits des valeurs contenues dans `Ry` et `Rz`
`MOD Rx, Ry, Rz`	`Rx` $\leftarrow$ `Ry modulo Rz`	`Rx` reçoit le reste de la division sur 32 bits de la valeur contenue dans `Ry` par celle contenue dans `Rz`
`JMP adr`	`PC` $\leftarrow$ `adr`	saut inconditionnel à `adr`
`JNZ Rx, adr`	`PC` $\leftarrow$ `adr` si `Rx` $\neq 0$	saut conditionnel à `adr` si la valeur dans `Rx` non nulle
`JGE Rx, adr`	`PC` $\leftarrow$ `adr` si `Rx` ≥ 0	saut conditionnel à `adr` si la valeur dans `Rx` ≥ 0
`OUT Rx`	impression	la valeur contenue dans `Rx` est affichée sur le canal de sortie
`HALT`	stop	fin du programme

```
MOV R0, R30              ; R30 contient l'adresse de début de rangement des variables en mémoire
MVI R1, #20
STW (R0), R1 ; x <- 20

ADD R0, R0, #4
MVI R1, #0
STW (R0), R1  ; i <- 0

ADD R0, R0, #4
STW (R0), R1  ; n <- 0

LDW R3, (R30) ; R3 <- x
MVI R4, #3 ; R4 <- 3

loop :  MOV R0, R30
            ADD R0, R0, #4
         LDW R1, (R0) ; R1 <- i
         SUB R31, R1, R3 ; R31 <- i - x
         JGE R31, end
         MOD R5, R1, R4 ; R5 <- i mod 3
         JNZ R5, fsi
         MOV R6, R30
         ADD R6, R6, #8 ; R6 <- @(n)
         LDW R7, (R6) ; R7 <- n
         ADD R7, R7, #1
         STW (R6), R7 ; n <- n+1
fsi:      MOV R8, R30
         ADD R8, R8, #4 <- R8 <- @(i)
         LDW R9, (R8) ; R9 <- i
         ADD R9, R9, #1
         STW (R8), R9 ; i <- i+1
        JMP loop
end:     MOV R0, R30
        ADD R0, R0, #8
        LDW R1, (R0) ; R1 <- n
        OUT R1
        HALT
```