Premier examen réparti LI324

L3 – Licence d'Informatique -Année 2015

1h45 - Tout document papier autorisé

Barème donné à titre indicatif

I. ORDONNANCEMENT (6,5 POINTS)

On considère un algorithme d'ordonnancement en **mode batch** hybride avec deux types de tâches :

- Les tâches de type A suivent un algorithme premier arrivé, premier servi (FCFS).
- Les tâches de types B suivent l'algorithme SJF qui choisit la tâche ayant le plus petit temps d'exécution **restant**.

Les tâches de type A sont plus prioritaires que les tâches de type B.

Soit la configuration suivante :

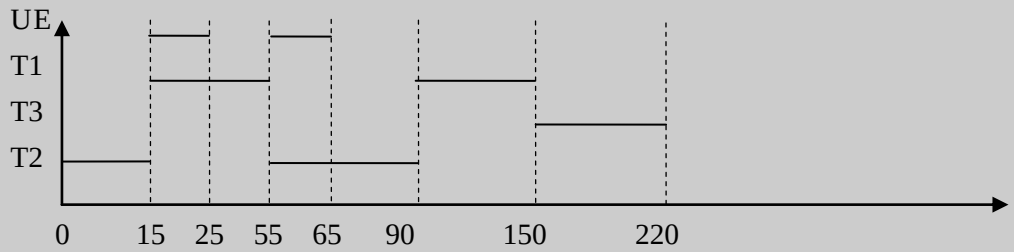
Tâches	Type	Instant création	Durée d'exécution sur le processeur	E/ S
T1	A	10 ms	100ms	Une seule après 40 ms
T2	B	0 ms	50ms	Une seule après 15 ms d'exécution
T3	B	0 ms	70ms	aucune

Les entrées/ sorties (E/ S) durent **10 ms** et se font sur **le même disque** (i.e., il y a une seule unité d'échange)

I.1. (3 points)

On considère une stratégie **sans réquisition** en cas de création ou de fin d'Entrée/ Sortie.

- Faites un diagramme temporel (Gantt) de l'évolution des tâches
- Indiquez les temps de réponse des tâches.

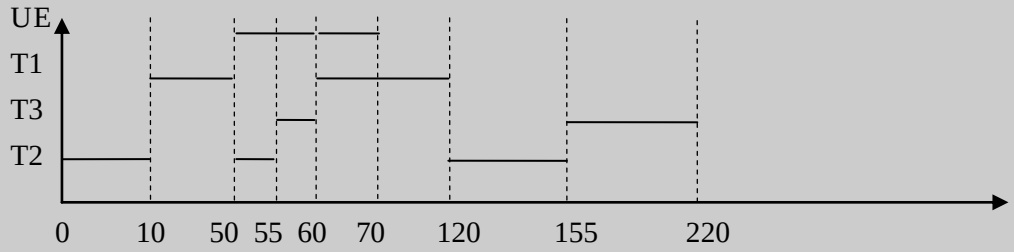


TR T1 = 150 – 10 = 140 ms
TR T2 = 90 -0 = 90 ms
TR T3 = 220 -0 = 220 ms

I.2. (3,5 points)

On considère maintenant une stratégie **avec réquisition** en cas de création ou de fin d'Entrée/ Sortie.

- Faites un diagramme temporel (Gantt) de l'évolution des tâches
- Indiquez les temps de réponse des tâches.
- Combien y-a-t-il eu de réquisitions et à quels moments ?



TR T1 = 120 – 10 = 110 ms
TR T2 = 155 -0 = 155 ms
TR T3 = 220 -0 = 220 ms

2 réquisitions à t = 10 et t = 60

II. PROCESSUS UNIX (7 POINTS)

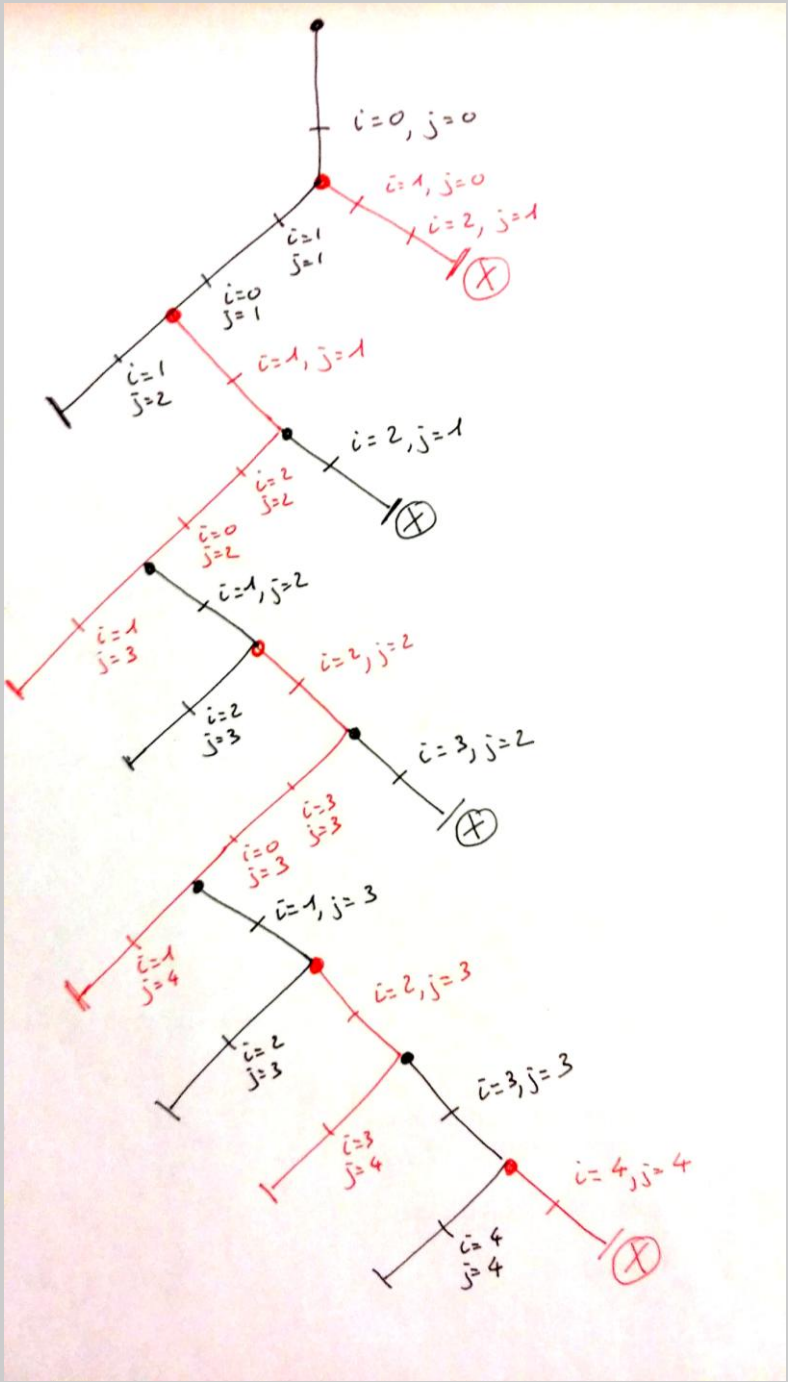
Soit le programme suivant :

```
1: #define  N  3

2: int main (int argc, char** argv) {
3:   int i=0; int j=0; int save_i=0; int save_j=0;
4:   pid_t p;

5:    printf ("i:%d, j:%d \n", save_i, save_j );

6:    while ((j <=N) && (i==j)) {
```

```
7:        for (i=0; i<=j; i++)
8:               if((p=fork ()) !=0)
9:                     break;

10:       i++;
11:       j++;

12:       if (p == 0){
13:          save_i=i; save_j=j
14:          printf ("i:%d; j:%d \n",save_i,save_j);
          }
15:   } /* while */

16:   exit (0);
17: }
```

II.1. (2,5 points)

Combien de processus sont créés par ce programme lors de son exécution (y compris le processus principal) ? Donnez l'arbre des dépendances avec l'affichage des processus.



11 processus

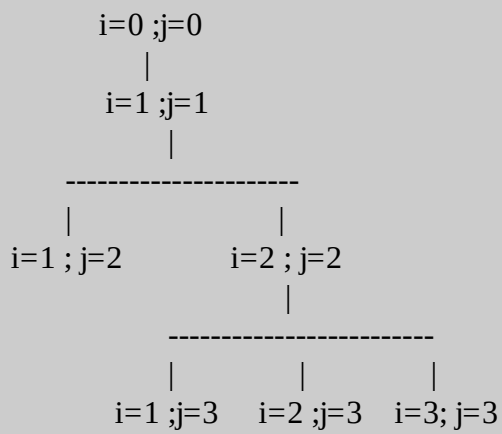
Nous voulons maintenant changer la ligne 14 : si j>=3, le programme utilise pour afficher la commande *echo* qui se trouve dans le répertoire */bin* ; sinon il garde la primitive *printf*

II.2. (2,5 points)

Changez le programme en conséquence.

Combien de processus sont-ils créés (y compris le processus principal)? Donnez l'arbre des dépendances avec l'affichage des processus.

```
char buffer [20];
Entre les lignes 11 et 12 ;

11:      save_i=i; save_j=j;
         if (j >=3) {
            sprintf(buffer, "i:%d; j:%d \ n",i, j );
            execl ("/bin/echo", "echo",buffer ,NULL);
         }
         else
12 :        printf ("i:%d; j:%d \ n",i,j);
```



7 processus créés

II.3. (2 points)

Modifiez encore une fois le programme original pour que le processus *main* ne se termine qu'après tous le autres processus.

NB: Dans votre solution, un processus ne doit pas attendre pour un nombre de fils plus important qu'il en a.

Ajouter entre la ligne 15 et 16 :

```
if (save_i==save_j)
   for ( i =0 ; i<save_j + 1  ; i++)
      wait (NULL) ;
```

III. SYNCHRONISATION (6,5 POINTS)

Soient les trois processus suivants. Le compteur du sémaphore S1 est initialisé à 1 tandis que les compteurs des sémaphores S2 et S3 sont initialisés à 0. La variable var est partagée entre les trois processus et initialisée à 0.

P1	P2	P3
(1.1) P(S2)	(2.1) P(S1)	(3.1) P(S1)
(1.2) P(S3)	(2.2) var=var+2	(3.2) var=var*3
(1.3) P(S1)	(2.3) V(S1)	(3.3) V(S1)
(1.4) var--	(2.4) V(S3)	(3.4) V(S2)
(1.5) V(S1)		

III.1. (3 points)

Donnez l'évolution des compteurs et de la file d'attente de chacun des sémaphores et de la variable partagée au cours de l'exécution représentée dans le tableau suivant. Dans la colonne « Remarque », vous indiquerez l'effet des appels P et V sur les processus (par exemple bloqué, réveillé etc.). Vous recopierez et complèterez le tableau suivante sur votre copie.

	var	Compteur S1	File S1	Compteur S2	File S2	Compteur S3	File S3	Remarque
--								
1.1								
2.1								
2.2								
3.1								
2.3								
3.2								
2.4								
3.3								
3.4								
1.2								
1.3								

1.4								
1,5								

	var	Compteur S1	File S1	Compteur S2	File S2	Compteur S3	File S3	Remarque
--	0	1	-	0	-	0	-	
1.1				-1	P1			P1 bloqué
2.1		0	-					P2 passe
2.2	2							
3.1		-1	P3					P3 bloqué
2.3		0	-					P3 débloqué
3.2	6							
2.4						1	-	
3.3		1	-					
3.4				0	-			P1 débloqué
1.2						0	-	P1 passe
1.3		0	-					P1 passe
1.4	5							
1,5		1	-					

III.2. (1 points)

Décrivez toutes les valeurs possibles de la variable partagée var à la fin de l'exécution de ces trois processus. Justifiez votre réponse.

P2 et P3 s'exécutent en premier et bloquent P1. Ils exécutent leur sections critiques en exclusion mutuelle. Selon l'ordonnancement, var vaut alors 6 (P2 d'abord, P3 ensuite) ou 5 (l'inverse). P1 est alors débloqué et exécute sa section critique. Selon la valeur de var, deux valeurs sont possibles pour var : 5 ou 6.

III.3. (2,5 points)

Soient les trois processus suivants (les points de suspensions représentent des instructions quelconques mais qui ne manipulent pas de sémaphores).

P1	P2	P3
…	V(S2)	P(S1)
P(S2)	…	...

P(S1)	P(S2)	P(S3)
…	V(S1)	...
V(S2)	...	V(S1)
….	V(S3)	V(S2)

L'exécution de ces trois processus présente-t-elle un risque d'interblocage si les compteurs des sémaphores S1, S2 et S3 sont respectivement initialisés à 1, 0 et 0 ? Justifiez votre réponse.

Oui, il y a l'interblocage suivant :
P3 : P(S1) →passant, cpt=0
P2 : V(S2) →cpt=1
P1 : P(S2) →passant,cpt=1
P1 : P(S1) →bloquant, cpt=-1
P2 : P(S2) →bloquant, cpt=-1
P3 : P(S3) →bloquant, cpt=-1