3IN010- Système - Durée : 1 h30min
Mars 2025 - Les documents ne sont pas autorisés
Partiel - Barème indicatif

1. INTERRUPTION (2,5 PTS)

1.1 (1 point)

Si un processus P a été interrompu par une interruption, est-il possible qu'une fois cette interruption traitée, un autre processus soit élu ? Si oui dans quels cas.

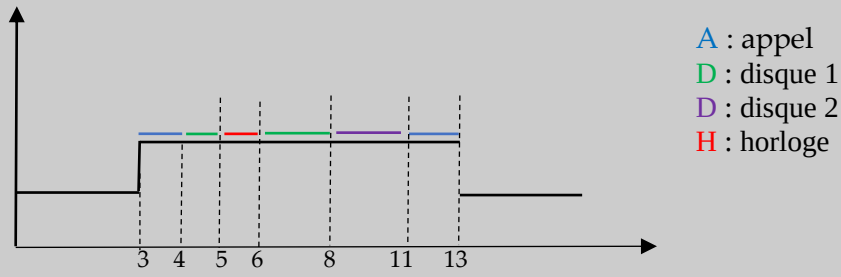
Un autre processus peut être élu dans les cas suivants :
- retour d'une IT horloge, lors d'une fin de quantum,
- retour d'une fin E/S, en cas de réquisition, si le processus réveillé est plus prioritaire,
- lors d'un appel système de création de processus, en cas de réquisition, si le nouveau processus est plus prioritaire.

1.2 (1,5 point)

Soit le scénario suivant avec un seul processus P. On suppose que :

- à t=3ms un appel système est appelé par P, cet appel doit s'exécuter pendant 3 ms,
- à t=4ms une interruption disque a lieu dont le traitement dure 3ms,
- à t=5ms une interruption horloge est déclenchée dont le traitement dure 1ms,
- à t=7ms une seconde interruption disque a lieu dont le traitement dure 3ms.

Dessinez un diagramme temporel où sont indiqués les traitements de l'appel système, les interruptions disque et de l'interruption horloge, ainsi que les instants où ils se terminent.



fin appel : 13ms
fin disque 1 : 8ms
fin disque 2 : 11ms
fin horloge : 6 ms

2. ORDONNANCEMENT (7 PTS)

On considère un ordonnancement en mode batch. Il y a une tâche « temps-réel » TR de plus haute priorité et 3 tâches T1, T2 et T3 de priorité inférieure et gérées selon la stratégie SJF (Shortest Job First) : on élit la tâche ayant le plus petit temps **restant** d'exécution sur le processeur.

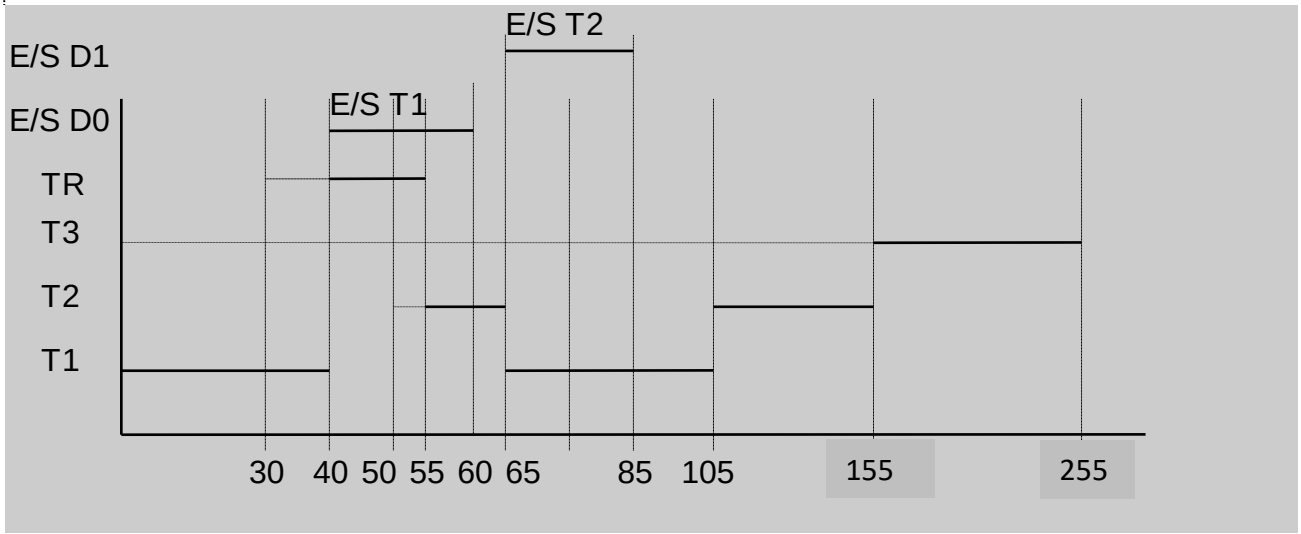
On dispose d'un ordinateur ayant **deux disques** D0 et D1 chacun géré par **un contrôleur différent** travaillant en parallèle avec le processeur.

Soit la configuration suivante :

Tâches	Instant création	Durée d'exécution sur le processeur	E/S	Durée E/S
TR	30 ms	15 ms	-	-
T1	0 ms	80 ms	Après 40 ms d'exécution sur le disque **D0**	20 ms
T2	50 ms	60 ms	Après 10 ms d'exécution sur le disque **D1**	20 ms
T3	0 ms	90 ms	-	-

2.1 (2 points)

On considère une stratégie sans aucune réquisition. Faites un diagramme temporel (Gantt) de l'évolution des tâches



2.2 (1 point)

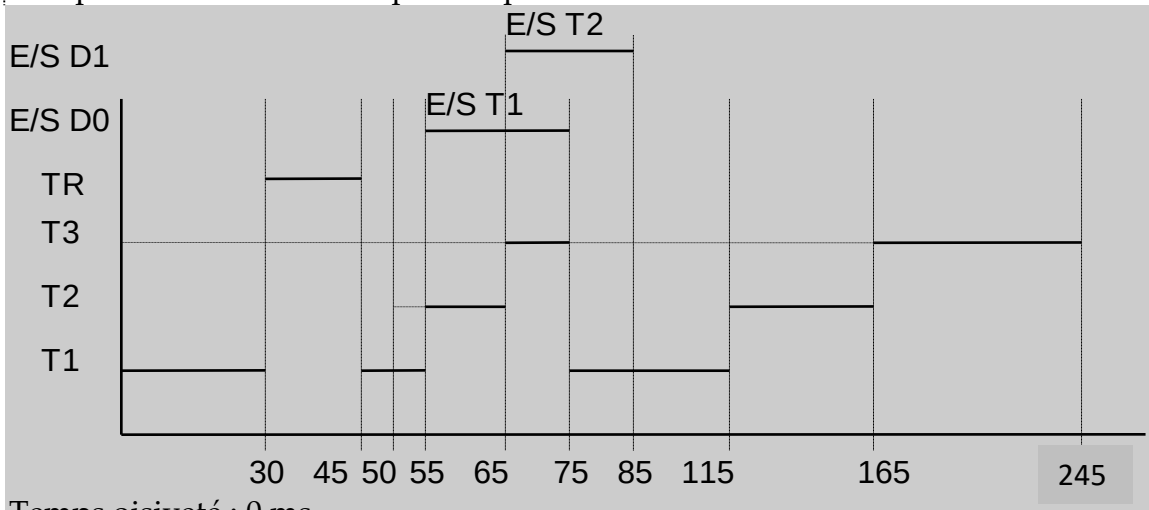
Indiquez le temps d'oisiveté du processeur et le temps de réponse des tâches.

Temps oisiveté : 0 ms
Tr TR = 55 – 30 = 25 ms
Tr T1 = 105ms
Tr T2 = 155 -50 = 105 ms

Tr T3 = 245 ms

2.3 (3 points)

On considère maintenant la même stratégie d'ordonnancement avec réquisition en cas de création ou de fin d'Entrée/Sortie. Refaites le diagramme temporel en conséquence et indiquez les nouveaux temps de réponse.



Temps oisiveté : 0 ms
Tr TR = 45 – 30 = 15 ms
Tr T1 = 115ms
Tr T2 = 165 -50 = 115 ms
Tr T3 = 245 ms

2.4 (1 point)

On souhaite modifier la stratégie batch SJF pour que les tâches ayant fait des entrées/sorties soient avantagées. On suppose que l'on ne connaît pas à l'avance les temps des E/S. En quelques lignes proposez comment réaliser cette modification.

Une idée simple est (1) de mesurer le temps des E/S faites par chaque tâche et (2) de soustraire ce temps au temps restant

3. PROCESSUS (6,5 PTS)

Considérez le programme suivant :

```
1: #define N 3
2: int level=1;
3: int i=1;
4: int j;
5: pid_t pid_var =0;


6: int main (int argc, char * argv[]) {
7:    printf ("i=%d ; level=%d \n",i,level);
```

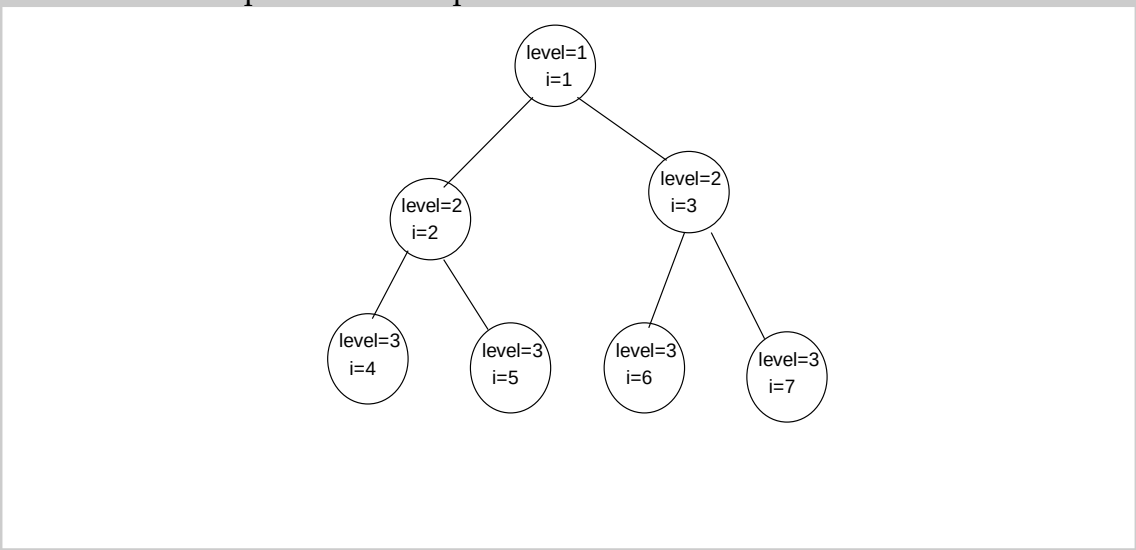
```
8:    while ((pid_var ==0) && (level <N)) {
9:       for (j=0; j<2 && ((pid_var= fork ()) != 0); j++)
10             ;
11:      if (pid_var == 0) {
12:         level ++;
13:         i = i*2 +j;
14:         printf ("i=%d ; level=%d \n",i,level);
15:      }
16:   }
17 :}
```

Observation : Nous considérons que la création d'un processus fils n'échoue pas.

3.1 (2,5 points)

Combien des processus sont créés ? Représentez l'arborescence des processus créés en donnant les affichages de chaque processus.

Arbre binaire complet : $2^N - 1 = 7$ processus créés



.

3.2 (1,5 points)

Modifiez le programme pour que le père initial attende la fin de tous les processus créés.

Observation :

- Un processus père ne doit pas attendre plus de fils qu'il en a.

- La modification doit être ajoutée après la ligne 16.

```
17:   if (level != N)
18:     for (j=0; j<2; j++)
19:       wait (NULL);
```

3.3 (2,5 points)

Modifiez le programme pour que chaque processus créé affiche : le pid de ses fils (s'ils existent), les pid des frères créés avant lui.

Observation :

- Vous ne pouvez pas ajouter des variables partagées au code.

```
#define N 3
int level=1;
int i=1;
int j;
pid_t pid_var=0;
pid_t frere, pid_fils[2];

int main (int argc, char * argv[]) {
  printf ("i=%d ; level=%d pid:%d\n",i,level, getppid());

  while ((pid_var ==0) && (level <N)) {
    for (j=0; j<2 && ((pid_var= fork ()) != 0); j++)
      pid_fils[j]=pid_var;

    if (pid_var == 0) {
      if (j==1)
        frere = pid_fils[0];
        level ++;
        i = i*2 +j;
        printf ("i=%d ; level=%d \n",i,level);
    }

  }
  if ( level != N)
    for (j=0; j<2; j++)
    wait (NULL);

  if ((i > 1) && (i%2 != 0)){
      printf ("i:%d : frere:%d \n", i, frere);
      if (level != N)
          printf ("i:%d, fils : %d, fils2:%d \n", i, pid_fils[0],
                pid_fils[1]);
    }

}
```

4. SYNCHRONISATION (4 PTS)

Nous considérons un processus coordinateur et deux processus de calcul, C1 et C2, qui manipulent deux variables partagées entières *a* et *b*. Pour simplifier, on suppose

que les instructions "*a = a +1*" et "*b = b+1*" sont indivisibles (correspondent à une instruction du processeur).

Le pseudo code des processus est le suivant :

```
#define N 5

/* Sémaphores */
1 : Sem S1 = CS(0);
2 : Sem S2 = CS (0) ;

/* Variables Partagées */
3 : int a=0, b=0;

/* Processus Coordinateur*/
4 : void Coordinateur(){
5 :         for (int i=0; i < N; i++)  {
6:              V(S1);
7:              V(S1);
8:              P(S2);
9:              P(S2);
10:     }
11:     printf("valeurs finales a=%d, b=%d\n",a,b);
12: }


/* Processus de calcul C1 */
13 :void calcul1(){
          /* Boucle infinie */
14:         while (1) {
15 :                    P(S1) ;
16 :                    a = a + 1;
17 :                    V(S2)
18 :        }
19 :}

/* Processus de calcul C2 */
20 :void calcul2(){
          /* Boucle infinie */
21:         while (1) {
22 :                    P(S1) ;
23 :                    b = b + 1;
24 :                    V(S2)
25 :        }
26 :}
```

4.1 (2 points)

- Dans quels cas les valeurs *a* et *b* sont différentes dans l'affichage de la ligne 11 ?
- Dans quels cas sont-elles identiques ?
- Donnez en **fonction de N** les valeurs maximales et minimales que peuvent avoir les variables *a* et *b* à la fin de la boucle for du coordinateur.

- a différent de b si de un des 2 processus s'exécute plus de N fois
- a = b si les deux processus boucle N fois
- Minimum : a= 2N, il suffit que P1 boucle 2 N fois; idem pour b Minimum: a= 0, il suffit que P2 boucle 2 N fois

4.2 (2 points)

On souhaite maintenant que *a* soit systématiquement égale à *b* à la fin de la boucle.

Modifiez les synchronisations en conséquence sans modifier les opérations sur *a* et *b*. Vous indiquerez clairement les valeurs initiales des sémaphores éventuellement ajoutés.

Définir un nouveau sémaphore: S3=CS(0)
Remplacer le P(S1) de C2 par un P(S3) et le V(S1) du coordinateur par V(S3)