3IN010- Système - Durée : 1 h30min
Mars 2024 - Les documents ne sont pas autorisés
Partiel - Barème indicatif

1. INTERRUPTION (2,5 PTS)

1.1 (0,5 point)

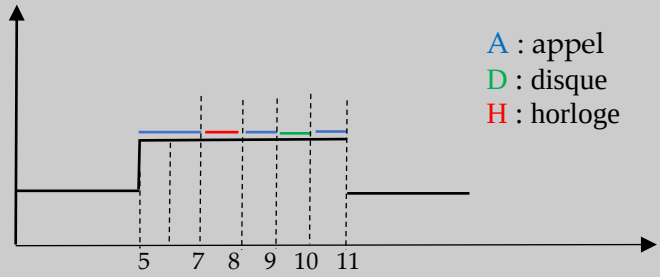
Le traitement d'interruptions utilise la pile utilisateur ou la pile système ?

Système

1.2 (1 point)

Soit le scénario suivant avec un seul processus P : on suppose qu'à t=5ms un appel système est appelé par P, cet appel doit s'exécuter pendant 4 ms, à t=7ms une interruption horloge a lieu dont le traitement dure 1ms, à t=9ms une interruption disque est déclenchée dont le traitement dure 1ms.

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

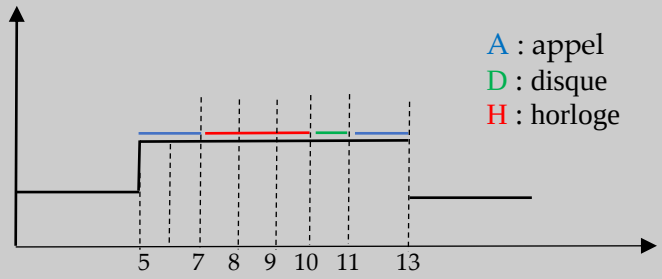


fin appel : 11ms
fin disque : 10 ms
fin horloge : 8 ms

1.3 (1 point)

On considère maintenant que le traitement de l'interruption horloge dure 3ms.

Dessinez le nouveau diagramme temporel.



fin appel : 13ms
fin disque : 11 ms
fin horloge : 10 ms

2. ORDONNANCEMENT (6 PTS)

On considère un algorithme d'ordonnancement en **mode** hybride avec deux types de tâches. Chaque type de tâche à sa file d'attente pour l'accès au processeur. **Les tâches de type A sont plus prioritaires que les tâches de type B**.

- Les tâches de type A suivent un algorithme batch FIFO.

- Les tâches de type B suivent l'algorithme du tourniquet (RR) en temps partagé avec un **quantum de 100 ms**. Une tâche est ré-insérée en queue de file uniquement en fin de quantum. Dans le cas d'une stratégie avec réquisition, si une tâche de type B est interrompue par une tâche de type A avant la fin de son quantum, elle reste en tête dans sa file et on lui attribuera à sa prochaine élection son quantum restant.

Soit la configuration suivante :

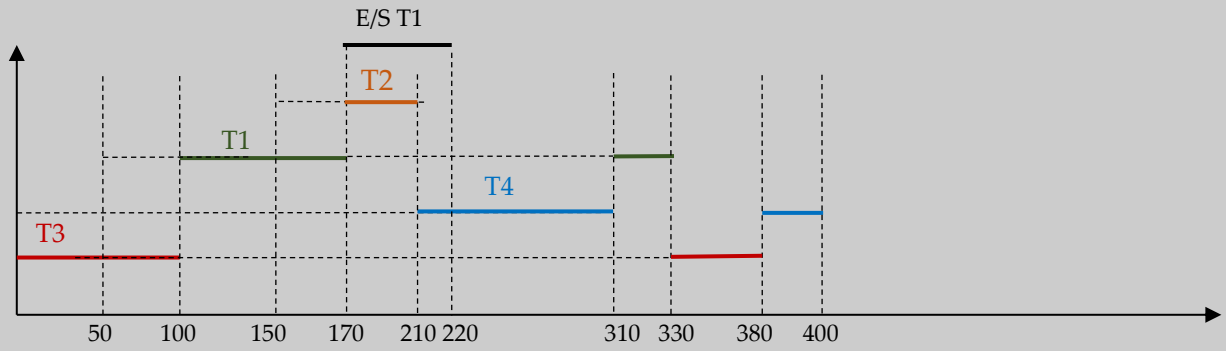
Tâches	Type	Instant création	Durée d'exécution sur le processeur	E/S
T1	A	50 ms	90ms	Une seule de 50ms après 70 ms
T2	A	150 ms	40ms	aucune
T3	B	0 ms	150ms	aucune
T4	B	0 ms	120ms	aucune

On suppose que T3 est créée avant T4

2.1 (2,5 points)

On considère une stratégie **sans réquisition**.

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

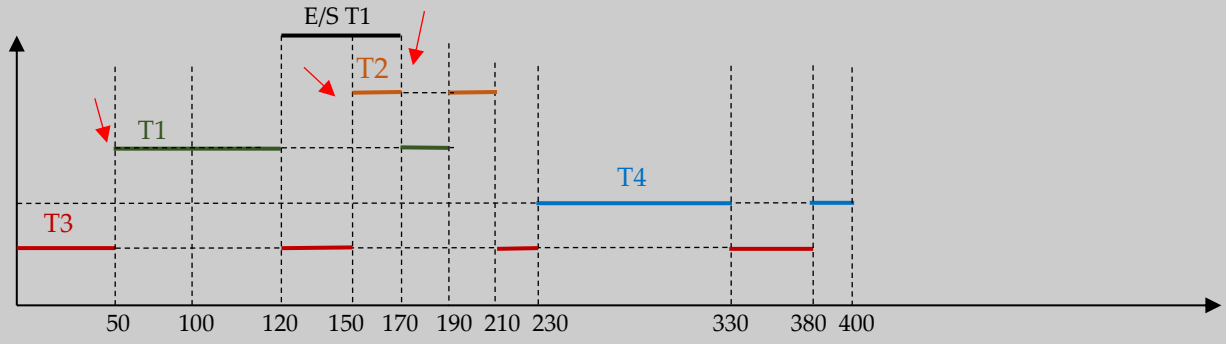


TR_1 = 330 - 50 = 280 ms
TR_2 = 210 – 150 = 60ms
TR_3 = 380-0 = 380ms
TR_4 = 400-0 = 400ms

2.2 (3,5 points)

On considère maintenant qu'une **réquisition** de tâche est possible lors d'une création de tâche ou d'une fin d'E/S.

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



TR_1 = 190 - 50 = 140 ms

TR_2 = 210 – 150 = 60ms
TR_3 = 380-0 = 380ms
TR_4 = 400-0 = 380ms

Réquisition à 50, 150ms et 170 ms.

3. PROCESSUS (4 PTS)

Soit le programme prog1 dont le code est le suivant :

```
1:   int main() {
2:     int e;
3:     int a = 10;
4:     if (fork() == 0) {
5:           printf("A  - %d\n", a);
6:           if (fork() == 0) {
7:                a = a + 2;
8:                printf("B  - %d\n", a);
9:                exit(1);
10:          }
11:          if (fork() == 0) {
12:               a  = a *2;
13:               printf("C  - %d\n", a);
14:               exit(2);
15:          }
16:          wait(&e);
17:          printf("D  - %d, %d \n", a, WEXITSTATUS(e));
18:          wait(&e);
19:          printf("E  - %d, %d \n", a, WEXITSTATUS(e));
20:          exit(3);
21:    }
22:    wait(&e);
23:    printf("F - %d, %d\n", a, WEXITSTATUS(e));
24:    return 0;
25:  }
```

On suppose que les appels à fork n'échouent pas.

3.1 (1 point)

Donnez l'arbre des processus fait par prog1.

```
prog1  ── fils ── petit-fils 1
               ── petit-fils 2
```

3.2 (2 points)

Donnez les affichages possibles fait par chacun des processus. En utilisant les lettres A, B, C, D, E et F, donnez les ordres possibles des affichages.

père
F – 10, 3

fils
A – 10

D – 10, 1
E – 10, 2
ou
D – 10,2
E – 10,1

petit-fils 1
B – 12

petit-fils 2
C - 20

Ordre :
A BCDE F
A BDCE F
A CBDE F
A CDBE F

Soit le programme prog2.c (ayant pour exécutable ./prog2) dont le code est le suivant :

```
26:  int main(int argc, char * argv[]) {
27:    printf("B, %d\n", atoi(argv[2]));
28:    return atoi(argv[1]);
29:  }
```

On remplace la ligne 8 de prog1.c par la ligne suivante

execl("./prog2", "prog2", "5", "6", NULL);

3.3 (1 points)

Quels sont les modifications dans les affichages induites par ce changement ?

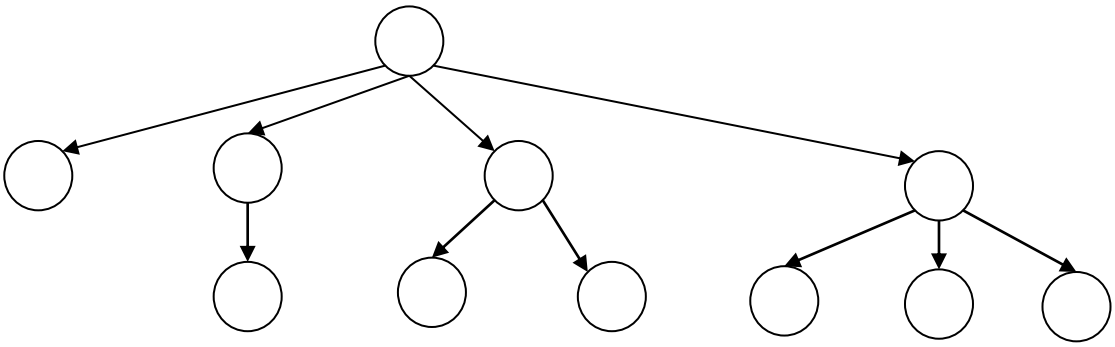
petit fils 1
B – 6

fils
D – 10, **5**
E – 10, 2

ou
D – 10, 2
E – 10, **5**

4. PROGRAMMATION PROCESSUS (4,5 PTS)

On veut écrire une fonction *void creer_processus(int nb)* qui crée *nb* processus fils. Le premier processus fils ne crée pas de fils, le 2ème en crée un, le 3ème deux et ainsi de suite. Par exemple, l'appel à **creer_processys(4)** entraînera l'arbre suivant :



4.1 (2,5 points)

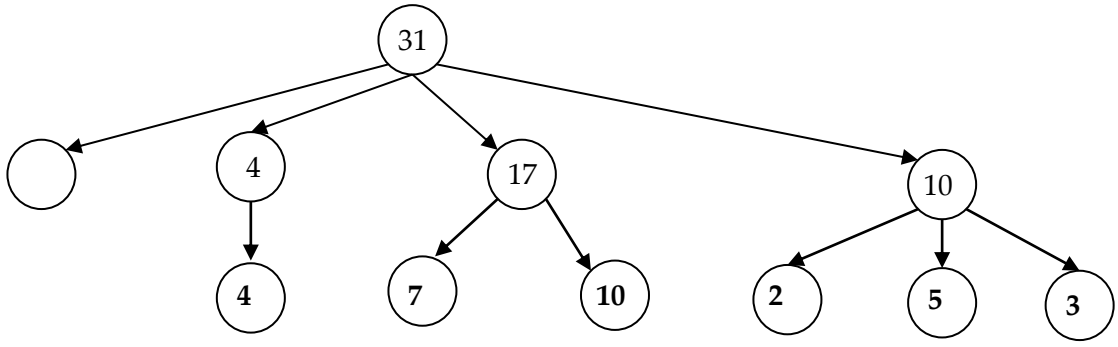
Donnez le code de la fonction *void creer_processus (int nb)*.

```
void creer_processus(int nb) {
 int i;
 for (i=0; i < nb; i++) {
  if (fork() == 0) {
    for (int j = 0; j < i; j++) {
      if (fork() == 0)
        return;
    }
    return;
  }
```

```
  }
}
```

Maintenant chaque processus petit-fils, appelle une fonction *int calcul()* qui retourne une valeur calculée puis se termine. Un processus qui a des fils doit faire la somme de toutes les valeurs calculées par ses fils en utilisant la fonction wait. La fonction *creer_processus* retourne enfin au père initial la somme totale calculée.

Par exemple, dans le schéma suivant on représente en gras les valeurs produites par la fonction calcul et les sommes calculées. Dans cet exemple, *creer_processus(4)* retournera 31.



4.2 (2 points)

Modifiez le code de la fonction *int creer_processus (int nb)* en conséquence

```
int creer_processus(int nb) {
  int i,v;
  int s = 0;

  for (i=0; i < nb; i++) {
    if (fork() == 0) {
      for (int j = 0; j < i; j++) {
        if (fork() == 0) {
          v=calcul();
          exit(v);
        }
      }
      for (int j = 0; j < i; j++) {
        int e;
        wait(&e);
        s += WEXITSTATUS(e);
      }
      exit(s);
    }
```

```
 }
 for (i = 0; i < nb; i++) {
      int e;
      wait(&e);
      s += WEXITSTATUS(e);
 }
 return(s);
}
```

5. SYNCHRONISATION (3 PTS)

On considère les quatre tâches suivantes :

A	B	C	D
P(S1); printf("a"); V(S2); P(S3); V(S4);	P(S1); printf("b"); V(S2);	P(S2); P(S2); printf("c"); V(S3);	P(S4); printf("d"); V(S1); V(S1);

Le sémaphore *S1* est initialisé à 2, les sémaphores *S2; S3* et *S4* sont initialisés à 0.

5.1 (1,5 points)

A la fin des 4 processus, quels sont les affichages possibles ?

abcd
bacd

5.2 (1,5 points)

Sans modifier les affichages, modifiez les synchronisations de ce programme pour commencer par l'affichage *d* puis avoir les mêmes séquences d'affichage possible pour les processus A, B et C que dans la question précédente (n'oubliez pas d'indiquer leurs valeurs initiales des sémaphores).

Plusieurs solutions

1) Initialiser S1 à 0 et S4 à 1

2) S1,S2 initialisé à 0

A	B	C	D
P(S1); printf("a"); V(S2);	P(S1); printf("b"); V(S2);	P(S2); P(S2); printf("c");	printf("d"); V(S1); V(S1);