

Premier examen réparti 3I010
L3 – Licence d’Informatique -Année 2017
1h45 - aucun document autorisé
Barème sur 20 points donné à titre indicatif

I. MULTIPROGRAMMATION BATCH (4 POINTS)

On considère deux tâches T_1 et T_2 , soumises simultanément à l’instant $t = 0$. T_1 et T_2 effectuent respectivement N et M fois le traitement suivant ($M > N$):

- Lecture d'une donnée sur le disque (durée l)
- Opération de calcul sur la donnée (durée c)
- Ecriture du résultat sur le disque (durée e)

Nous considérons une multiprogrammation en mode batch et qu'il existe qu'une unité de change, autrement dit les E/S ne peuvent pas être faites concurremment. On supposera $c > l$ et $l > e$ et qu'il n'y a que ces deux tâches à exécuter dans le système.

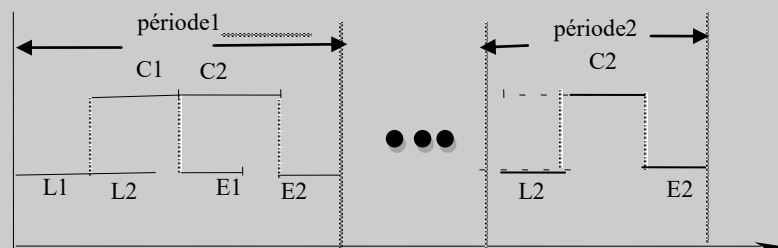
II.1 (2 points)

Représentez sur un diagramme de Gantt l'exécution des tâches T_1 et T_2 . Calculez le temps total T pour exécuter les deux tâches. Application numérique : $l = 1$ ms, $c = 4$ ms, $e = 5$ ms, $N = 2$ et $M = 5$.

2 réponses acceptées :

1)

$e < l < c$



période1 exécutée N fois ; periode2 exécutée $M-N$

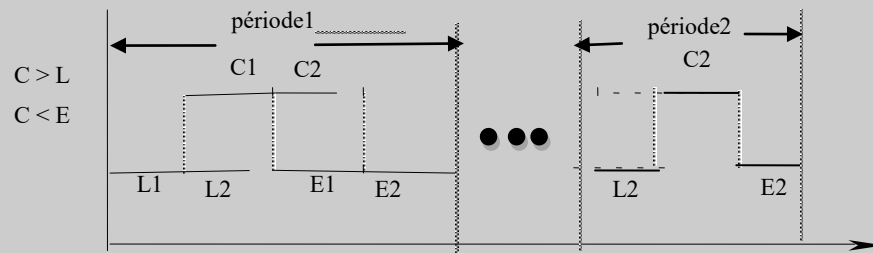
$$T = N. (l + 2c + e) + (M-N) (l + c + e) = M(l+c+e) + Nc$$

Application numérique:

$$T = 5 \cdot 10 + 2 \cdot 4 = 58$$

2)

$l < c < e$



période1 exécutée N fois ; periode2 exécutée M-N

$$T = N. (l + c + 2e) + (M-N) (l + c + e) = M(l+c+e) + Ne$$

Application numérique:

$$T = 5 \cdot 10 + 2 \cdot 5 = 60$$

III.1 (2 points)

Donnez les différentes formules du taux d'occupation de la CPU τ_p et du taux d'occupation de l'unité d'échange τ_u . Est-ce que ces taux restent toujours constant jusqu'à la fin de l'exécution des 2 tâches ? Justifiez vos réponses.

Non.

$\tau_p = 2.c / (l + 2c + e)$ lorsque les deux tâches s'exécutent; $\tau_p = c / (l+c+e)$ lorsque T_2 est seule

$\tau_u = 2.(l+e) / (l + 2c + e)$ lorsque les deux tâches s'exécutent; $\tau_u = (l+e)/(l+c+e)$ lorsque T_2 est seule

Non.

$\tau_p = 2.c / (l + c + 2e)$ lorsque les deux tâches s'exécutent; $\tau_p = c / (l+c+e)$ lorsque T_2 est seule

$\tau_u = 2.(l+e) / (l + c + 2e)$ lorsque les deux tâches s'exécutent; $\tau_u = (l+e)/(l+c+e)$ lorsque T_2 est seule

II. ORDONNANCEMENT (11 POINTS)

On considère un algorithme d'ordonnancement à **temps partagé** inspiré de Linux 2.4.

- A chaque processus est associé une priorité allant de 80 à 0 (80 étant plus prioritaire que 0).
- Lorsqu'un processus est créé, il a la priorité 80 (la plus élevée).

- Le quantum est égal à 10 ticks. 1 tick dure 10 ms.
- A chaque tick, la priorité du processus élu est diminuée de 1.
- A chaque fin de quantum, le système élit le processus ayant la priorité la plus élevée.
- Lorsqu'un processus a fini son quantum, la prochaine fois qu'il sera élu, on lui attribut de nouveau un quantum de 10 ticks.
- Lorsqu'un processus est créé via fork, le quantum attribué au fils est égal **au quantum restant du père divisé par 2**. Le quantum restant du père est lui aussi divisé par 2. Par exemple si un processus crée un fils après 4 ticks, il ne lui restera que 3 ticks et son fils aura également un quantum de 3 ticks.

II.1. (0,5 point)

Quel est l'intérêt de diminuer la priorité du processus élu et d'attribuer aux nouveaux processus la priorité maximum ?

Cela permet de donner une priorité plus forte aux processus courts.

Soit un programme exécutant le code suivant :

```
int main() {  
    a();  
    if (fork() == 0) {  
        a();  
        fork();  
    }  
    b();  
    wait(NULL);  
}
```

Les appels aux fonctions a() et b() génèrent respectivement des calculs de 20ms et 50ms. Pour simplifier vous considérerez qu'il n'y a pas de commutation au moment du fork(). On rappelle qu'un wait est non bloquant pour un processus qui n'a jamais eu de fils.

II.2. (1,5 points)

Combien de processus sont créés ?

Représentez les processus créés par ce programme sous forme d'un arbre

main → fils → petit-fils

3 processus

II.3. (1 point)

Pour chaque processus, indiquez le nombre de fois où les fonctions a() et b() sont appelées.

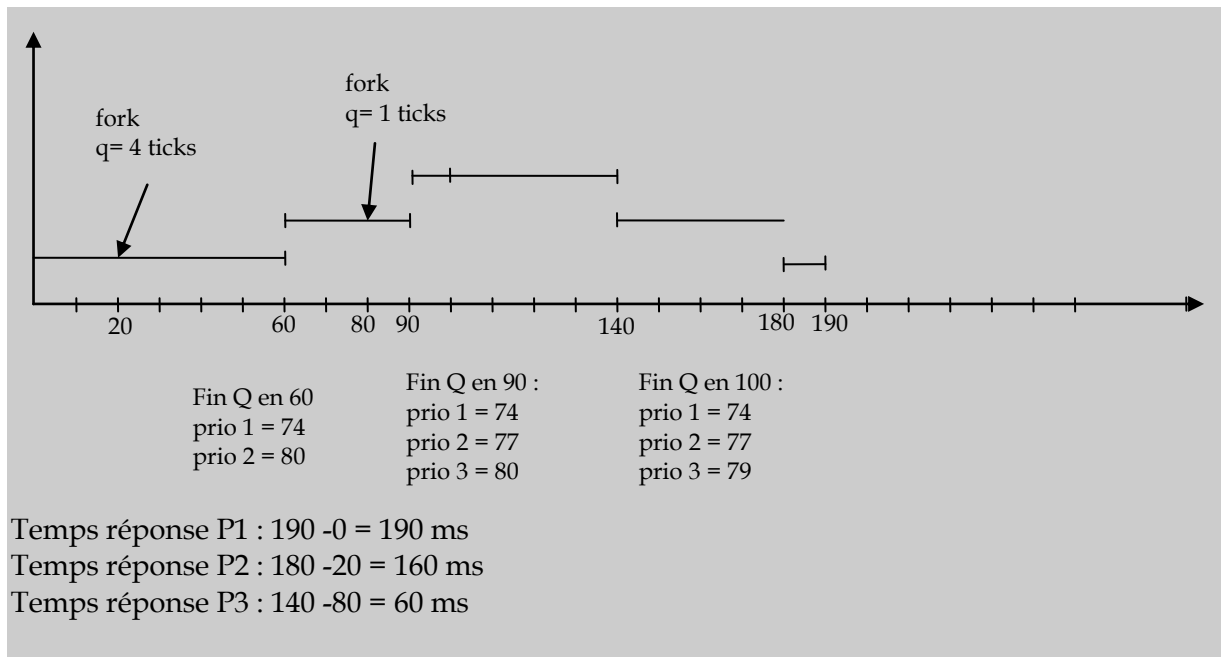
Père : a() et b()

Fils : a() et b()

Petit-fils : b()

II.4. (4 points : 3+0,5+0,5)

- Faites un diagramme temporel (Gantt) de l'évolution des processus en considérant que le programme est lancé à l'instant 0.
- Indiquez à chaque fin de quantum la priorité des processus.
- Indiquez les temps de réponse des processus.



II.5. (1 points)

Pour les processus créés, indiquez la période de temps où ils ont été à l'état Zombie.

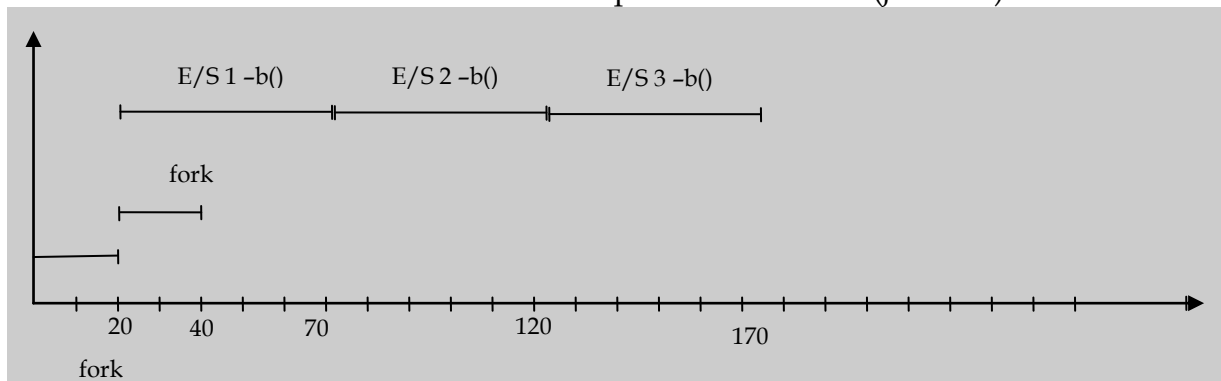
P2 est zombie de 180 à 190

P3 est zombie de 140 à 180

II.6. (3 points : 2+0,5+0,5)

La fonction b() plutôt que faire des calculs réalise désormais une entrée/sortie (E/S) de 50ms. Les E/S ont toutes lieu sur le même disque (il ne peut pas y avoir 2 E/S en parallèle).

- Refaites le diagramme de Gantt en conséquence en indiquant en plus l'activité de l'unité d'échange (le contrôleur).
- A quels moments les trois tâches se terminent ?
- Y a-t-il à un moment de l'exécution des processus zombie (justifiez) ?

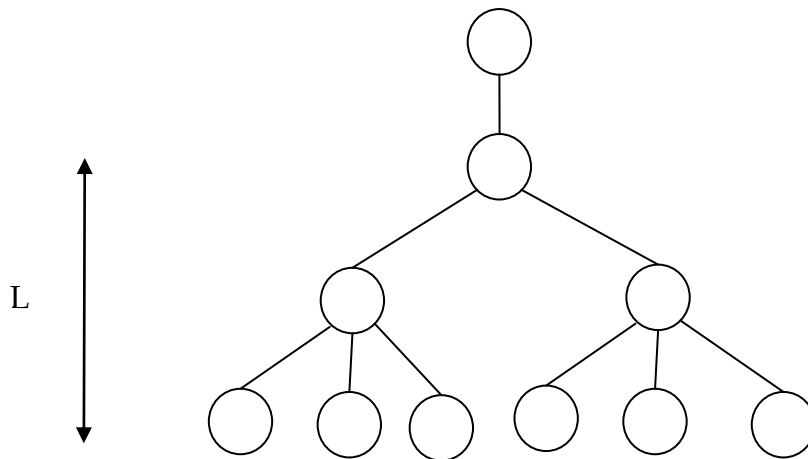


Les 3 processus se terminent à 170.

Il n'y a pas de zombie car les wait ont lieu avant la fin des processus (P1 fait wait en 70, P2 en 120)

III. PROCESSUS (5 POINTS)

Considérez l'arborescence de processus ci-dessous. Le programme principal crée 1 fils ; le fils créé à son tour 2 fils, les 2 nouveaux fils créent 3 fils chacun et ainsi de suite jusqu'à ce que l'arborescence de processus ait une hauteur L . Pour simplifier on suppose que les `fork` ne font pas d'erreur.



III.1. (2,5 points)

En utilisant l'appel système `fork()`, donnez le code du programme qui crée une telle arborescence. Avant de se terminer un processus affiche son `pid`. La valeur de L est définie comme une constante.

```
#define L 4
int main () {

    int i, j, k;
    pid_t p;
    int cpt = 0;

    i = 0;

    while (i < L) {
        j = 0;
        while (j <= i && (p = fork()) != 0)
            j++;
        if (p != 0) {
            break;
        }
        i++;
    }
}
```

```
printf("%d\n", getpid());
return 0;
}
```

III.2. (2,5 points)

Chaque feuille de l'arbre (i.e., les processus au niveau L) fait un `exit(1)`. On souhaite que la racine de l'arbre (i.e., le processus "main") attende la fin de toutes les feuilles et affiche la somme des valeurs de sortie retournées par les feuilles (valeurs fournies en paramètre à `exit`). Modifiez le programme de la question précédente en conséquence.

```
#define L 4
int main () {

    int i, j, k;
    pid_t p;
    int cpt = 0, a;

    i = 0;

    while (i < L) {
        j = 0;
        while (j <= i && (p = fork()) != 0)
            j++;
        if (p != 0) {
            break;
        }
        i++;
    }
    if (i == L)
        exit(1);
    else {
        for (j=0; j <= i; j++) {
            wait(&a);
            cpt += WEXITSTATUS(a);
        }
        if (i > 0)
            exit(cpt);
        else
            printf("cpt = %d\n", cpt);
    }

    return 0;
}
```