NOM LAVAUD

Prénom Florian

Promo 2016 MIL

Date 10/04/15

LAVAUD Florian
M1 - 2014

20,5

MATIÈRE Systèmes temps réel

Questions de cours

1) Un système temps réel est un système qui gère les tâches asynchrones issus du monde physique. délai ? (0,5) (1)

2) Déterminisme : Dans un cas similaire, l'exécution sera toujours semblable
Predictibilité : L'exécution des tâches est gérée dans l'intégralité des cas possibles.
Fiabilité : Le système est maintenable et ne perd/détériore pas ses ressources. (1)

3) L'envoi d'une rafale d'interruptions peut provoquer l'ignorement et la perte de ces interruptions car le handler n'a pas le temps de traiter l'interruption avant qu'une nouvelle interruption survienne (1 octet seulement). Nous pouvons réduire ce problème en limitant les exécutions dans le handler mais nous ne pouvons pas le corriger intégralement. (1)

4) Si l'architecture, pour l'exécution d'un programme en boucle, est plus performante, il est possible qu'une tâche s'exécute avant sa fenêtre temporelle. Nous pouvons régler ce problème avec des pauses et signaux de réveil. (1)

5) Le temps réel strict à l'inverse du souple ne permet pas le dépassement de fenêtre temporelle pour certaines tâches. (0,5)

6) Ce problème intervient entre autre lors de partage de ressources. Une tâche exploite une ressource partagée et s'interrompt pour laisser la place à une tâche prioritaire, mais des tâches ne peuvent s'exécuter à cause du blocage des ressources partagées et lorsque la tâche initiale débloque ses ressources, l'échéance est déjà dépassée. Le protocole d'héritage de priorité permet de corriger ce problème en élevant la priorité de la tâche ayant les ressources partagées à la priorité de la tâche qui doit s'exécuter mais qui n'a pas accès aux ressources. (1)

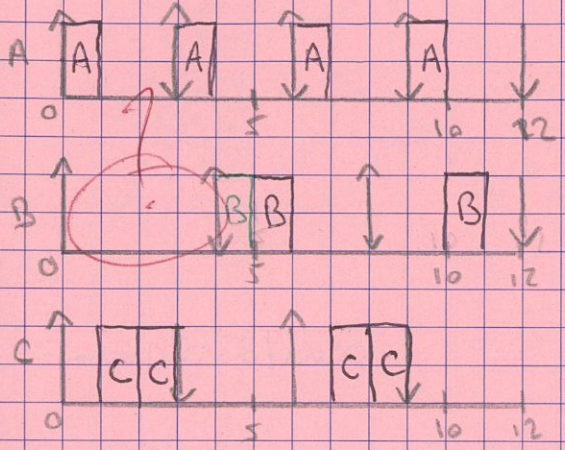
Ordonnancement

7) $U = \frac{\sum C}{T} = \frac{1}{3} + \frac{1}{4} + \frac{2}{6} = \frac{11}{12}$ Période d'étude : [0; PPCM] [0; 12] (1)

8) • EDF

Priorité : $\frac{1}{\text{Echéance relative}}$

Prio (A) = Prio (C) > Prio (B)
A → C → B
ou C → A → B



↑ : activation
↓ : echeance

(Q5)

Le B (marqué en vert) montre un non-respect de l'échéance B pour la première période.

• LLF

Priorité : $\frac{1}{\text{laxité (charge dynamique)}}$

$Md = (Tg - Tc) - R$

On va calculer la laxité à chaque activation, c'est-à-dire à :
t = 0 s, 3s, 4s, 6s, 8s et 9s

t = 0 s :

$Md_A = (3-0) - 1 = 2$
$Md_B = (4-0) - 1 = 3$
$Md_C = (3-0) - 2 = 1$
Prio (C) > Prio (A) > Prio (B)

t = 3s :

$Md_A = (6-3) - 1 = 2$
$Md_B = (4-3) - 1 = 0$
Prio (B) > Prio (A)

t = 4s :

$Md_A = (6-4) - 1 = 1$
$Md_B = (8-4) - 1 = 3$
Prio (A) > Prio (B)

t = 6s :

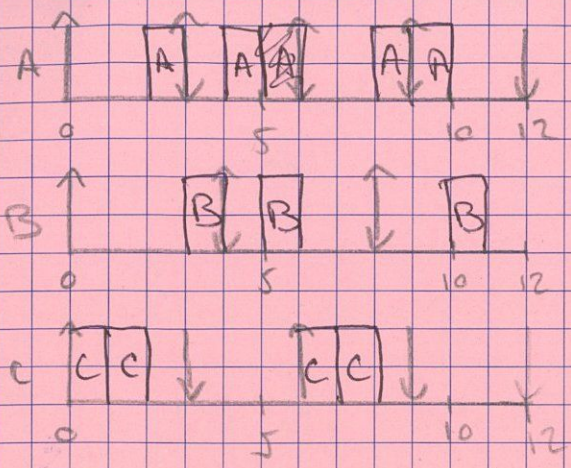
$Md_A = (9-6) - 1 = 2$
$Md_C = (9-6) - 2 = 1$
Prio (C) > Prio (A)

t = 8 s :

$Md_A = (9-8) - 1 = 0$
$Md_B = (12-8) - 1 = 3$
Prio (A) > Prio (B)

t = 9 s :

$Md_A = (12-9) - 1 = 2$
$Md_B = (12-9) - 1 = 2$
Prio (A) = Prio (B)



9) Pour que RMS soit optimal, il faut 2 conditions :

- Condition nécessaire : $W \leq 1$
- Condition suffisante : $W \leq U(n)$

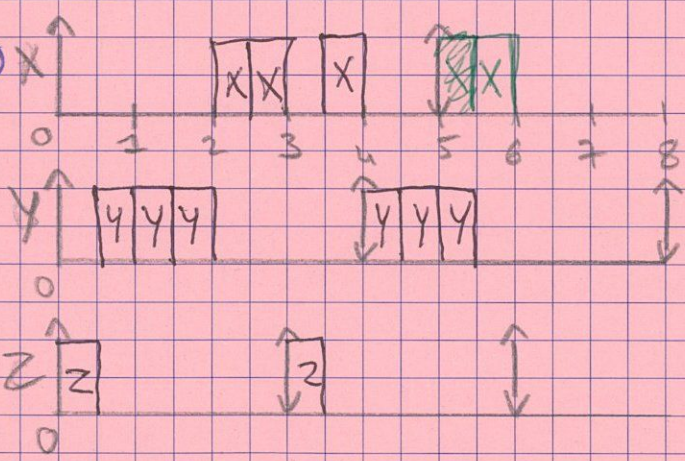
Dans notre cas : $W = \frac{56,5}{60} \leq 1 \Rightarrow$ CV

$W \not\leq U(3)$

Regardons à 5s, dans le pire des scénarios il y a $1X+2Y+2Z = 6s$. Nous ne pouvons pas affirmer que un plan RMS est possible.

Il faut donc essayer avec le plan d'ordonnancement.

10a)



Priorité : $\frac{1}{T}$

$Prio(Z) > Prio(Y) > Prio(X)$

$\Rightarrow$ Non respect de l'échéance X (en vert)

11) Nous faisons la dernière demi-seconde de X, une seconde après l'échéance. Il faut donc augmenter la vitesse d'une seconde.

difficile à exploiter...

12) EDF et LLF sont optimaux si $W \leq 1$. Dans notre cas c'est juste mais nous remarquons que la période équivaut à l'échéance donc EDF ne fonctionnerait pas.

13)

$\rightarrow$

	T_g	T_d	C	Ms
T_0	2-k	0	1	1-k
T_1	3	1	1	1
T_2	3	1	2	0
T_3	2	1	k	1-k
T_4	4	2	1	1
T_5	4	3	1	0
T_6	4	1+k	2	1-k
T_7	7	3	3	1
T_8	7	4	2	1
T_9	7	3+k	3	1-k

(2)

13) 7 unités de temps ; $k > 0$ (1)

On doit passer à l'exécution 16+k sec, ce qui signifie que pour 7 unités de temps, il faut au moins $\frac{16}{7} \leq 2,2 \Rightarrow$ 3 processeurs.

14) $Ms \geqslant 0 \Rightarrow 1-k \geqslant 0 \Rightarrow k = 1$ (énoncé $k > 0$)

0						7
T_0	T_1 T_2 T_3	T_4 T_2 T_6	T_7 T_5 T_6	T_7 T_8 T_9	T_7 T_8 T_3	T_9

(1)

Synchronisation de processus

15) x valeurs possibles : 25, 39 (1)

16) Les processus A et B peuvent se bloquer mutuellement. (0,5)

17) • On utilise un seul sémaphore (équivalent à un mutex ici)

A
```
P(mutextest)
...
V(mutextest)
```

B
```
P(mutextest)
...
V(mutextest)
```

(1)

• On inverse l'ordre des sémaphores dans B

A
```
P(sem1)
P(sem2)
...
V(sem1)
V(sem2)
```

B
```
P(sem2)
P(sem1)
...
V(sem2)
V(sem1)
```

(3)

18) Nous utilisons :

• 2 sémaphores
  - occ : nombre de cases occupées dans la file
  - lib : nombre de case lib

occ = 0 ; lib = N

• 1 mutex qui bloque l'accès d'écriture à un écrivain

```
Ecrivain () {
    P(lib);
    P(mutex_ecr);
    ecriture_tampon(&donnee);
    V(mutex_ecr);
    V(occ);
}
```

```
Lecteur {
    P(occ);
    lecture_tampon(&donnee);
    V(lib);
}
```

LAVAUD Florian
M1 IL

3 Synchronisation de processus

Exercice 3.1

Squelette de programme des processus `A` et `B` :

`A`	`B`
P(sem1)	P(sem2)
P(sem2)	P(sem1)
...	...
x=x+4	x=x+5
...	...
x=x*3	x=x+2
...	...
V(sem2)	V(sem1)
V(sem1)	V(sem2)

Question 15. Quelles sont les valeurs possibles de "x" (valeur initiale : x=2) suivant l'entrelacement des processus `A` et `B` après une exécution en parallèle ?

Question 16. Quel problème peut survenir lors de l'exécution des processus `A` et `B` ?

Question 17. Proposez deux solutions différentes pour résoudre ce problème. Quels sont les nouveaux squelettes de programme de chaque processus pour chacune des solutions ?

Exercice 3.2 : Producteurs / Consommateur

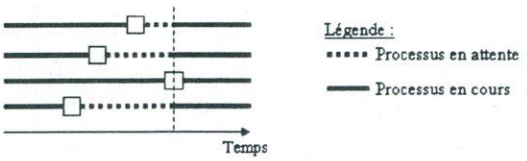
Le problème est composé d'un ensemble de processus et d'un tampon partagé de taille `N`. **Plusieurs** threads `Ecrivain( )` pourront écrire des données dans un tampon partagée. **Un seul** thread `Lecteur()` pourra lire une donnée de ce tampon. La gestion du tampon ne sera pas à faire, vous utiliserez une fonction `ecrire_tampon(donnee)` pour écrire une donnée dans le tampon et une fonction `lecture_tampon(&donnee)` pour lire une donnée dans celui-ci.

Contraintes du système :
- Les threads `Ecrivain()` effectueront (chacun) M écritures dans le tampon (M > N) ;
- Le thread `Lecteur()` lira des données dans le tampon tant qu'il y a en ;
- Les lectures et les écritures pourront être effectuées **en parallèle**.

Question 18. Ecrire les programmes du thread `Lecteur( )` et du thread `Ecrivain( )` en respectant bien les contraintes du système. Les synchronisations entre les threads se feront via des mutex et/ou sémaphores et les primitves P( ) et V( ). **Précisez l'utilité de chacun des mutex et/ou sémaphore(s) que vous avez utilisés.**

Exercice 3.3 : Barrière de synchronisation pour N processus

Une barrière de synchronisation permet de garantir qu'un certain nombre de processus ait passé un point spécifique. Ainsi, chaque processus arrivant sur cette barrière devra se mettre en attente jusqu'à ce que tous les processus soient arrivés au niveau de cette barrière. La figure ci-dessous illustre le fonctionnement de la barrière de synchronisation pour 4 processus.



Question 19. Implantez un mécanisme de barrière de synchronisation entre N processus à l'aide de sémaphores et/ou mutex. Un squelette de programme vous est fourni en figure 19.

Notes :
- Notez que certaines cases du squelette de programme fourni peuvent être vides et certaines peuvent contenir plusieurs lignes de code si le besoin s'en fait sentir ;
- La variable globale « nb_process »représente le nombre restant de processus à synchroniser. Elle est initialisée à N.

```
void barriere(void)
{
    int i = 0;
    P(thread_arrive);
    nb_process--;
    
    if(nb_process == 0)
    {
        for(i=0 ; i<N ; i++)
        {
            V(thread_suite)
        }
        nb_process = N;
    }
    
}
```

// Les threads donnent un jeton quand ils atteignent la barrière à thread_arrive et peuvent continuer grâce à thread_suite

(9,5)