intra - ift2245 Nom:_____ Matricule:_____

Examen Intra

IFT 2245

26 février 2020

Directives

— Aucune documentation autorisée.
— Calculatrice est autorisée.
— Répondre dans le cahier fourni.
— Le pointage pour chaque question est entre parenthèses (total = 20).
— Les traductions en anglais sont en *italics*.
— Notez clairement toutes les suppositions que vous faites.

Question 0. *Nom et prénom (1 point de bonus)*

Écrire son nom et prénom et son matricule en haut et aussi sur la page couverture du cahier réponse.

Question 1. *Réponse courte (3 points au total)*

(a) (1 point) Après qu'un processus ait été **interrompu** (*interrupted*) dans quel état est-il :

1. prêt (*ready*)
2. nouveau (*new*)
3. en exécution (*executing*)
4. en attente (*waiting*)

(b) (1 point) Quelles sont les deux méthodes de communication inter-processus (*inter-process communication* - IPC) ? passage de message / memoire partagé

(c) (1 point) Dans le cas où toutes les ressources n'ont qu'une seule instance, un cycle dans le graphe d'allocation des ressources (*resource allocation graph*) correspond à quel type de condition de interblocage (*deadlock*) :

1. nécessaire et suffisant
2. pas nécessaire ou suffisant
3. nécessaire mais pas suffisant
4. pas nécessaire mais suffisant

Question 2. *Interblocage (3 points au total)*

Considérez un système d'exploitation qui gère les interblocages en garantissant qu'il ne s'en produise pas (évitement). Il n'y a qu'une seule ressource R avec 12 instances, et l'allocation actuelle de la ressource entre 3 processus est la suivante :

Processus	Maximum	Actuel
P_0	10	4
P_1	5	2
P_2	8	2

(a) (1 point) le système est-il dans un état sûr ? pourquoi ou pourquoi pas ? oui → séquence P_1, P_2, P_0 ou P_1, P_0, P_2

(b) (1 point) P_1 fait une demande de 4 ressources, doit-elle être donnée ? pourquoi ou pourquoi pas ? Non $4+2 > 5$

(c) (1 point) P_2 fait une demande pour 1 ressource, doit-elle être donnée ? pourquoi ou pourquoi pas ? oui → séquence P_1, P_2, P_0

Question 3. *Forks et Threads (5 points au total)*

(a) (3 points) Qu'est qui est imprimé quand ce code est exécuté :

```
int value = 0;
void *runner(void *param);
int main(int argc, char *argv[])
{
  pid_t pid;
  pthread_t tid;
  pthread_attr_t attr;
  pid = fork();
  if (pid == 0) {
    pthread_attr_init(&attr);
    pthread_create(&tid,&attr,runner,NULL);
    pthread_join(tid,NULL);
    printf("LINE A value = %d\n",value);
  }
  else if (pid > 0) {
    wait(NULL);
    printf("LINE C value = %d\n",value);
  }
  printf("LINE D value = %d\n", value);

}

void *runner(void *param) {
  value += 5;
  printf("LINE B value = %d\n",value);
  pthread_exit(0);
}
```

LINE B value = 5
LINE A value = 5
LINE D value = 5
LINE C value = 0
LINE D value = 0

(b) (2 points) Écrivez une équation pour décrire le nombre de fois que le "bonjour" sera imprimé en fonction de la variable N pour le code suivant :

```
int main()
{
  pid_t pid;
  int i = 1;
  int n = N; // N est remplacé par un entier
  for (;i<=n;){
    pid = fork();
    printf("bonjour \n"); // combien de fois cela est-il exécuté?
    i++;
  }
  return 0;
```

$$f(N) = \sum_{i=1}^{N} 2^i$$

Question 4. *Synchronisation (4 points au total)*

Puisqu'un seul processus est autorisé à l'intérieur d'un moniteur, il existe deux options pour gérer le cas où une variable de condition est libérée, soit le processus qui signale la condition doit quitter le moniteur (signal et attendre *signal and wait*) ou le processus qui signale la condition doit être autorisé à se terminer, mais un accès prioritaire doit être accordé au processus qui a été bloqué à la condition qui a été signalée (signaler et continuer *signal and continue*). Voici une implémentation d'un moniteur pour le cas **signaler et attendre** à l'aide de sémaphores :

Implémentation moniteur :

```
semaphore mutex;  // (initialisé à 1 - contrôle l'access au moniteur)
semaphore next;   // (initialisé à 0 - utilisé pour redonner le
contrôle a un processus qui a été bloquer après un x.signal)
int next_count = 0; // (nombre de processus suspendus par les conditions)

// Chaque fonction F  vais être remplacer par:

wait(mutex);
...
body of F;
...
if (next_count > 0)
    signal(next);
else
    signal(mutex);
```

```
int signal_count = 0;
semaphore next_signals[];
int next_count = 0; // même
semaphore next; // même
semaphore mutex; // même.
exit() {
  while (signal_count > 0){
    next_count ++;
    signal next_signals[signal_count
    -1]
```

Implémentation condition :

```
// Pour chaque variable condition x:
semaphore x_sem;   // (initialisé à 0)
int x_count = 0;

// L'opération x.wait():

x_count++;
if (next_count > 0)
    signal(next);
else
    signal(mutex);
wait(x_sem);
x_count--;

// L'opération x.signal():

if (x_count > 0) {
    next_count++;
    signal(x_sem);
    wait(next);
    next_count--;
}

// else fait rien
```

Comment modifieriez-vous l'implémentation pour le cas de **signaler et continuer**.

Question 5. *L'ordonnancement (5 points au total)*

Pour un ensemble de processus $P_{1..4}$, nous anticipons la situation suivante :

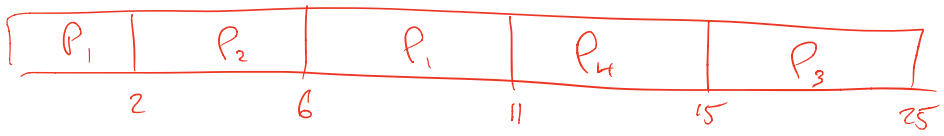
Processus	Arrivée* (ms)	CPU *burst time*(ms)
P_1	0	7
P_2	2	4
P_3	4	10
P_4	8	4

* indique le moment où le processus est d'abord prêt à être exécuté.

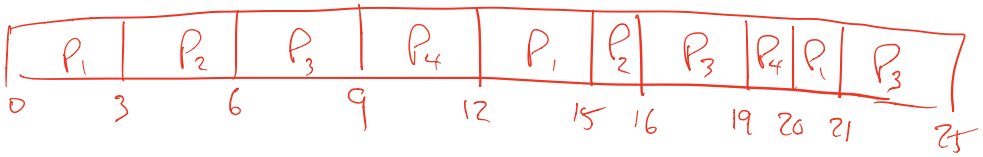
(a) (1 point) Dessinez le diagramme de Gantt pour l'algorithme "shortest remaining time first" (préemptif)
(b) (1 point) Dessinez le diagramme de Gantt pour l'algorithme "Round Robin" avec quantum = 3ms
(c) (1 point) Quel est le temps de réponse moyen (*average response time*) pour (b) et (c)

Considérons un système en temps réel avec deux processus qui s'exécutent sur une période fixe (ils doivent s'exécuter complètement une fois au cours de chaque période). Le processus P_1 a une période $p_1 = 40ms$ et un temps d'exécution de $t_1 = 15ms$. Le processus P_2 a une période de $p_2 = 65$ et un temps d'exécution de $t_2 = 45ms$.
(d) (1 point) Dessinez le diagramme de Gantt pour l'algorithm d'ordonancement "deadline plus proche" jusqu'à l'instant $t = 150ms$.
(e) (1 point) Quelle est l'utilisation du processeur (*CPU utilization*) pour (d) ?

(a)



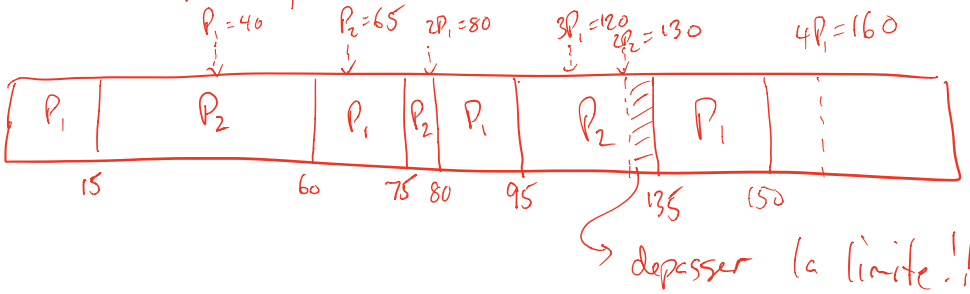
(b)



(c) (i) $\frac{1}{4}(0 + 0 + 11 + 3) = 14/4$

(ii) $\frac{1}{4}(0 + 1 + 2 + 1) = 1$

(d) version pre-emptif



(e) $15/40 + 45/65 > 100\% \longrightarrow 100\%$