Enoncé du partiel ASPD
ESIAL 2, année 2001/2002

Indications : Seuls les documents de cours sont autorisés (cours de Dominique Mery, cours sur algorithmique parallèle, document sur les fonctions PVM).

Il faudra justifier toutes vos réponses. A chaque affirmation, il faudra faire un commentaire pertinent.

Un soin tout particulier devra être apporté à la présentation de la copie.

Exercice : ordonnancement de tâches indépendantes sur p processeurs identiques.

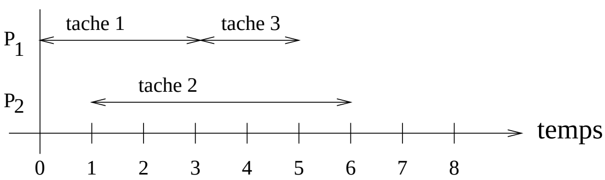
L'objectif de cet exercice est de déterminer la performance d'un ordonnancement de tâches indépendantes sur p processeurs. Nous sommes dans la situation suivante. Il n'y a aucune dépendance entre les tâches entre elles. De plus, un contrôleur reçoit au fur et à mesure (une seule à la fois)[1] les tâches indépendantes : le contrôleur reçoit l'ensemble de tâches $E = \{1, 2, \ldots, n\}$ dans cet ordre. Selon un algorithme, un contrôleur affecte une tâche à un processeur et ce choix est irrévocable. Notons $d(i)$ la date d'arrivée de la tâche i. Remarquons que $d(i) < d(j)$ si $i < j$. Chaque tâche i demande exactement $w(i)$ unités de temps pour s'exécuter.

On définit un ordonnancement σ qui attribue à une tâche i une machine $m(i)$ et une date de début d'exécution $t(i)$. L'algorithme est le suivant : le contrôleur impose à une tâche i de s'exécuter à l'étape $t(i)$ sur la machine $m(i)$ si c'est la machine $m(i)$ qui est disponible à l'étape $t(i)$ le plus tôt possible parmi toutes les p machines.

Voici un exemple :

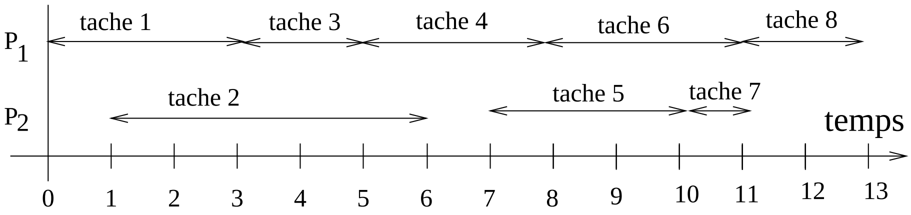
tâche	1	2	3	4	5	6	7	8
durée	3	5	2	3	3	3	1	2
date d'arrivée	0	1	2	3	7	8	9	10

On supposera qu'il y a deux processeurs. La tâche 1 arrive à l'étape 0. Comme les deux processeurs sont libres, la tâche 1 s'exécute à la date 0 sur le processeur P_1. A l'étape 1, la tâche 2 arrive et puisque le processeur P_2 est libre, la tâche 2 s'exécute à la date 1 sur ce processeur. A l'étape 2, la tâche 3 arrive. Comme le processeur P_1 sera libre le plus tôt à l'étape 3 (le processeur P_2 sera libre à l'étape 6), la tâche 3 s'exécute à la date 3 sur le processeur P_1. Et ainsi de suite. . .



[1]autrement dit, le contrôleur reçoit au plus une tâche par unité de temps.

Question 1 : Continuer l'exemple (Ordonnancer toutes les tâches de l'exemple).
Correction :



On notera σ l'ordonnancement obtenu par l'algorithme précédent et $D(\sigma) = max\{t(i)+w(i)|i = 1,\ldots,n\}$ la date où l'ordonnancement σ se termine. Notons D_{opt}, la durée de l'ordonnancement optimal.

Question 2 : Donner la définition formelle de D_{opt}. En déduire un minorant de $D(\sigma)$.
Correction :
- $D_{opt} = min\{D(\sigma')|\ \sigma'$un ordonnancement de $E\}$.
- $D_{opt} \leq D(\sigma)$.

Question 3 : En vous inspirant du cours, donner une borne inférieure de D_{opt} en fonction des durées des tâches.
Correction : $D_{opt} \geq \frac{\sum_{i\in E} w(i)}{p}$.

Question 4 : Donner une borne inférieure de D_{opt} en fonction des dates d'arrivée des tâches.
Correction : $D_{opt} \geq max_{i\in E}(w(i) + d(i))$

Question 5 : Soit j une tâche de E se terminant en dernier.
Question 5.1 : Expliquer pourquoi la tâche j n'a pas été exécuté aux dates $d(j),\ldots,t(j)-1$.
Correction : Durant cette période, aucun des p processeurs est inactif. Si c'était le cas, la tâche j aurait été exécuté plus tôt.
Question 5.2 : Que pouvez-vous en déduire sur $(t(j) - d(j))$ par rapport aux durées de toutes tâches.
Correction :
- $p * (t(j) - d(j)) \leq \sum_{i\in E, i\neq j} w(i)$.
- $t(j) \leq \frac{\sum_{i\in E, i\neq j} w(i)}{p} + d(j)$.

Question 5.3 : Calculer la valeur β de l'inégalité suivante $D(\sigma) \leq \beta D_{opt}$. Que concluez-vous ?
Correction :
- $t(j) \leq \frac{\sum_{i\in E, i\neq j} w(i)}{p} + d(j)$.
- $t(j) + w(j) \leq \frac{\sum_{i\in E, i\neq j} w(i)}{p} + d(j) + w(j)$
- $t(j) + w(j) \leq D_{opt} + D_{opt} \leq 2D_{opt}$

Exercice : Élection d'un leader

On considère un réseau constitué de n sites $P_1,\ldots,P_n$. Chaque site P_i a une valeur val_i avec $\forall i \neq j$, $val_i \neq val_j$. Le leader est le site avec la plus grande valeur val_i. A un moment donné, l'un des sites veut initier une élection, c'est à dire initier un algorithme distribué qui a pour but de déterminer lequel des sites est le leader. Ici, le réseau est organisé en cycle unidirectionnel : les communications élémentaires entre sites voisins sont P_i vers P_{i+1} (modulo n). On propose l'algorithme suivant pour le site i.

```
Lors du reveil spontane du site i ou, a la reception d'un message pour
la premiere fois
     envoyer(<1,val(i)>)
     etat(i) :=actif ;
     max(i):=val(i) ;

Lors de la reception du message <1,V>
      si  (etat(i) ==actif) alors
            si (V!= max(i)) alors
                  envoyer(<2,V>);
                  voisin(i) :=V;
            sinon max(i) est elu, prevenir les autres,
     sinon envoyer (<1,V>);

Lors de la reception du message <2,V>
     si  (etat(i) ==actif) alors
             si (voisin(i)> V) et (voisin(i)>max(i)) alors
                    max(i):=  voisin(i);
                    envoyer (<1,max(i)>);
             sinon etat(i) :=passif ;
     sinon envoyer (<2,V>);
```

Question 1 : Appliquer cet algorithme sur un exemple sur un anneau d'au moins 4 sites sachant qu'un seul processeur initie l'élection.

Question 2 : A quoi servent les variables locales `max(i), etat(i), voisin(i)` du site i ?
Correction : Rappel : En TD, nous avons vu des algorithmes d'élection de leaders sur des cycles bi-directionnels. Le principe de ces algorithmes est le suivant : chacune phase, calcule si pour chaque site actif, il est un maximum local. Si c'est le cas, il reste actif sinon, il devient passif.

la variable $etat(i)$ du site i : si le site i reste $actif(etat(i))$, alors le site i sait que son prédécesseur est un maximum local. Si $etat(i) = passif$, alors le site i participe toujours à l'élection mais il retransmet simplement les messages.

les variables $max(i)$ et $voisin(i)$ du site i : Pour que le site i sache que son prédécesseur est un maximum local, il faut qu'il connaisse la valeur de son prédécesseur, et la valeur du prédécesseur de son prédécesseur. Les variables $max(i)$ et $voisin(i)$ servent à enregistrer ces valeurs.

Question 3 : A quoi servent les deux messages ?
Correction : Le rôle de ces deux messages, servent à un site actif que connaître la valeur de son prédécesseur et du prédécesseur de son prédécesseur.

Question 4 : Par qui peut être proclamé le résultat ?
Correction : Lorsqu'un site reçoit une valeur qu'il a déjà reçu, l'algorithme se termine. C'est ce site qui connaît le résultat. Ce site n'est pas le site ayant la plus grande valeur car il devient inactif dès la première phase.

Question 5 : Quelle est la complexité de l'algorithme en nombre de messages ?
Correction : Ici, il y a $log_2 n$ phases (car pour chaque phase, il y au moins la moitié des noeuds actifs avant cette phase qui deviennent passifs après). Durant chaque phase, chaque arête est traversée par deux messages $< 1, V >$ et $< 2, V >$.

Le nombre de messages : $2n \times log_2 n = O(nlog_2 n)$.

Exemple de programme PVM : calcul de π

Correction : Voir le TP sur PVM.