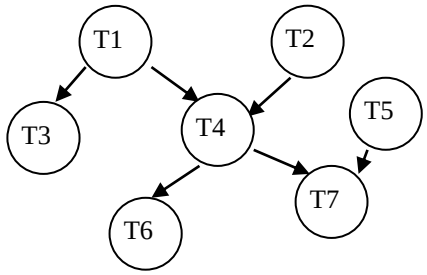
USTHB
Faculté d'Electronique et Informatique
Département Informatique
Epreuve Finale – Durée : 1h30
(Systèmes d'exploitation- M1: HPC+IV) Le 08/01/2018 - Année 18/19

Exercice 1 (6 pts= 1+5)

Soit le graphe de précédences suivant :



a-Vérifier si le graphe est proprement lié.
b-Exprimer ce graphe à l'aide de Fork/Join.

Exercice 2 : (6 pts= 2 + 4)

On suppose un système constitué de plusieurs processus et uniquement de trois événements mémorisés e1, e2 et e3. On considère les deux primitives suivantes :

- **Attendre (e1 et e2 ou e3)**: Elle permet d'attendre que : soit les événements e1 et e2 se produisent soit l'événement e3 se produise.
- **Déclencher (ei)** : Elle permet de réveiller tous les processus en attente de cet événement si leurs conditions sont satisfaites et tous les événements attendus sont acquittés. Si aucun processus ne l'attend, l'événement ei est mémorisé.
  On désire implémenter ces deux primitives à l'aide des sémaphores :

1- Donner les structures de données communes nécessaires et indiquer leurs rôles.
2- Donner une solution à ce système.

Exercice 3 : (8 points= 1+1+1 + 5)

On s'intéresse à la gestion, à l'aide de sémaphores, d'une classe de ressources à M exemplaires partagées par N processus numérotés de *1* à N. Chaque processus peut demander k exemplaires de ressources à la fois et est satisfait au *mieux*, ce qui veut dire qu'il peut continuer son exécution même si on satisfait partiellement sa demande. Par exemple, si P_i demande *10* exemplaires et le système ne peut lui donner que *4*, P_i se contente de ces ressources allouées (tout en étant informé du nombre alloué) et continue son exécution (sa requête est alors satisfaite). A la libération de ressources, les processus bloqués en attente de ressources sont aussi satisfait au mieux.

A/ Décrire les structures principales à utiliser.
B/ Expliciter les procédures nécessaires avec leurs paramètres.
C/ Donner la forme générale d'un processus.
D/ Réaliser cette gestion à l'aide des sémaphores.

Bon Courage

Correction de l'Epreuve Finale

Exercice 1 (6 pts= 1+5)

a-Le graphe est non proprement lié car il n'est pas possible de le décomposer en sous-graphes ni parfaitement parallèles ni parfaitement séquentiels même au départ.

b-Expression à l'aide de Fork/ Join.

Begin
n12 :=2 ; n45 :=2; ;
Fork et2; Fork et5; T1; Fork et4; T3; Aller à F;
et2 : T2 ;
et4 : Join n12 ; T4 ; Fork et7 ; T6 ; Aller à F ;
et5 : T5 ;
et7 : Join n45 ; T7 ; Aller à F ;
F :
Fin.

Exercice N°2 : (6 pts= 2 + 4)

1- Structure de données communes et leurs rôles

- Remarquons que la condition est satisfaire au même temps pour tous les processus bloqués ; un seul sémaphore; (S : sémaphore :=0) est donc nécessaire pour bloquer tous les processus puisqu'ils vont tous être réveillés en même temps.
- e:Tableau [1..3] de entier :=0 ; ils indiquent, chacun, l'état (mémorisé ou non) de l'événement correspondant.
- cpt : entier :=0 ; compteur du nombre de processus bloqués.
- mutex : sémaphore :=1 ; il permet de rendre les primitives en exclusion mutuelle afin de protéger les variables de synchronisation utilisées.

2- Implémentation

S : sémaphore :=0 ; e:Tableau [1..3] de entier :=0 ; cpt : entier :=0 ; mutex : sémaphore :=1

```
Primitive Attendre (e1 et e2 ou e3) ;
Début
P(mutex)
Si (((e1<>1)ou (e2<>1)) et (e3<>1)) Alors
  cpt :=cpt+1 ;V(mutex) ;
  P(S) ;
Sinon
e1:=0; e2:=0; e3 :=0 V(mutex)
Fsi
Fin ;
```

```
Primitive Déclencher (e[j]) ;
Début
P(mutex) ;
Si (e[j]<>1) alors
        e[j]:=1;
  Si (j=3) ou (((j=1) et (e[2]=1)) ou ((j=2)et (e[1]=1)))
        Alors
                Si (cpt)<>0) Alors
                    Tant que (cpt<>0) Faire
                            V(S) ; cpt :=cpt-1 ;
                    Fait ;
                   e[1] :=0 ; e[2] :=0 ; e[3] :=0
                Fsi
    Fsi
Fsi
Fin ;
```

Exercice 3: (8 points= 1+1+1+5)

A- Structures de données

- *S : Tableau [1..n] de sémaphore :=0 ;* - Chaque sémaphore est associé à un processus, il sert à le faire attendre si sa requête n'est pas satisfaite.
- *mutex: sémaphore :=1*;- Sert à protéger les différentes variables de synchronisation.
- *nbr: entier :=M;* - Nombre de ressources dans la réserve à tout moment.
- On utilise aussi une file explicite *f* constituée d'une suite d'identités de processus. Elle permet de garder l'ordre d'arrivée des processus en attente de ressources.
  Les procédures de manipulation de *f* supposées existantes sont :

Premier (f) : fonction qui retourne le premier élément de f.
Insérer (f, proc) : insère un élément proc dans f en fin de file.
Extraire (f): supprime le premier élément de *f.*

B- Les procédures nécessaires:

Procédure demander (k: entier, ***var*** *nbacquit: entier, i : entier) :* permet de demander *k* ressources, *nbacquit* est le nombre de ressources allouées effectivement retourné au processus demandeur et *i* est l'identité du processus demandeur.
Procédure libérer (m : entier): Permet de libérer *m* ressources.

C- Forme générale d'un processus

Processus Pi ;
-
Procédure demander (k, nbacquit, i);
<Utiliser ressources>
Procédure libérer (m) ;
-
Fin.

D/ Implémentation

```
Const N=… ;M =… ;
Var S : Tableau [1..N] de sémaphore :=0 ; mutex: sémaphore :=1;
nbr: entier :=M; réveillé : booléen :=faux;
<Déclaration de f et ses procédures d'accès>
Procédure demander (k: entier, var nbacquit: entier, i :entier) ;
Début
P(mutex)
Si (nbr=0) Alors  insérer (f, i)  V(mutex);  P(S[i]) Fsi;
Si (nbr<=k) Alors nbacquit :=nbr; nbr :=0 Sinon nbacquit :=k; nbr :=nbr-k fsi;
Si reveillé  Alors reveillé :=faux; V(attente)
             Sinon V(mutex)
Fsi
Fin;
Procédure libérer (m : entier) ;
Var j: entier;
Début
 P(mutex) ; nbr :=nbr+m;
 Tant que non vide (f) et (nbr<>0)
Faire j :=premier(f).id; extraire (f); réveillé :=vrai; V(S[j];  P(attente) Fait;
V(mutex)
Fin ;
```