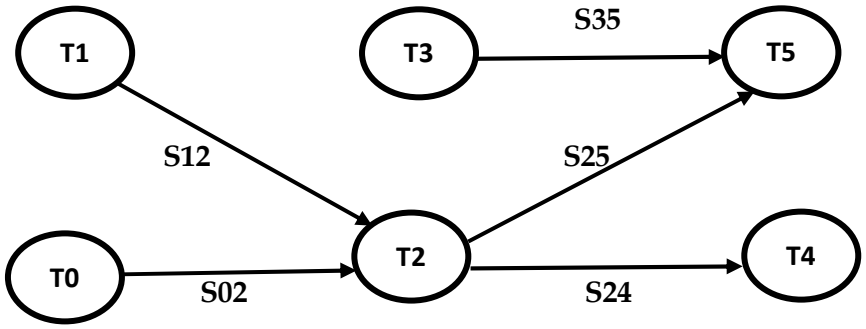
Examen de fin de semestre / Corrigé

Exercice 1 : (4 points)

L'outil « sémaphore » peut être utilisé pour garantir des ordonnancements entre tâches. Une solution possible consiste à utiliser un sémaphore -initialisé à 0- par arête dans le graphe d'ordonnancement des tâches :



Ordonnancement avec cinq (05) sémaphores

Avec ces sémaphores, l'ordonnancement est obtenu, si chaque tâche exécute les codes suivants :

T0	T1	T2	T3	T4	T5
Code V(S02)	Code V(S12)	P(S02) P(S12) Code V(S24) V(S25)	Code V(S35)	P(S24) code	P(S25) P(S35) code

Expliquer le fonctionnement de la tâche T2.

T2 effectue les actions suivantes :

- **Elle attend la fin de T0 du fait du sémaphore S02,**
- **Elle attend la fin de T1 du fait du sémaphore S12,**
- **Elle exécute son propre code,**
- **Elle libère le sémaphore S24, ce qui débloque éventuellement T4,**
- **Elle libère le sémaphore S25, ce qui débloque T5.**

Exercice 2 : (4 points)

Le pseudo-code suivant montre une solution incomplète du problème Producteur/Consommateur à l'aide d'un moniteur.

```
Processus Producteur
Début
    TantQue (Vrai) faire
        Produire_Element ;
        Producteur_Consommateur.mettre ;
    FinTantQue
Fin
```

```
Processus Consommateur
Début
    TantQue (Vrai) faire
        Producteur_Consommateur.retirer ;
        Cosommer_Element ;
    FinTantQue
Fin
```

MONITOR Producteur-Consommateur

Début

// mettre, retirer : procédures exportées

compteur : entier ;

plein, vide : condition ;

```
Procédure mettre
Début
  Si  compteur = n Alors
      wait(plein)
  Finsi
  compteur = compteur + 1 ;
  Si compteur = 1
      Alors signal(vide)
  finsi
Fin mettre
```

```
Procédure retirer
Début
    Si  compteur = 0 Alors
        wait(vide)
  Finsi
  compteur = compteur - 1 ;
  Si compteur = n - 1
      Alors signal(plein)
  finsi
Fin retirer
```

Compteur = 0 ;

Fin Producteur-Consommateur

Ecrire le pseudo-code des procédures **mettre** et **retirer**.

Exercice 3 : (6 points)

a) (2,5 pts) Comment peut-on prévenir l'occurrence d'interblocages dans un système ?

Pour qu'un interblocage se produise, il faut que chacune des 4 conditions nécessaires soient vérifiées En s'assurant qu'au moins une de ces conditions ne peut pas se vérifier, nous pouvons prévenir l'occurrence d'interblocages.

- Exclusion mutuelle : Elle doit être vérifiée pour des ressources non partageables (Par eg., une imprimante ne peut pas être partagée par plusieurs processus simultanément) En général, il n'est pas possible de prévenir les interblocages en niant la condition de l'exclusion mutuelle car certaines ressources sont intrinsèquement non partageables.

- Occupation & attente : Forcer les processus à demander toutes leurs ressources avant l'exécution ou bien une seule à la fois Inefficace et risque de famine (i.e. : un processus ayant besoin de plusieurs ressources prisées peut attendre indéfiniment, car au moins l'une des ressources dont il a besoin est toujours allouée à un autre processus).

- Aucune préemption : Si un processus requiert une ressource occupée alors toutes ses ressources sont implicitement libérées et il passe en attente Généralement, non utilisable pour certains types de ressources (Eg. : les imprimantes & les unités de bandes magnétiques).

- Attente circulaire : Imposer un ordre total sur l'allocation des ressources et forcer chaque processus à demander les ressources dans un ordre d'énumération croissant Difficile à mettre en œuvre.

b) (2 pts) Donnez la définition algorithmique des primitives P et V pour un sémaphore binaire **s**.

```
P(s)
Si n(s) = 1
   Alors
        n(s) ← 0
    Sinon
        Etat(p) ← bloqué ;
        Entrer(p, f(s))
 Finsi
```

```
V(s)
Si f(s) est vide
   Alors
        n(s) ← 1
    Sinon
        Choisir q dans f(s) ;
        Etat(q) ← actif ;
        Sortir(q, f(s))
Finsi
```

c) (1,5 pts) La famine est un problème inhérent à la gestion des ressources. Comment peut-on éviter ce problème.

Pour éviter le problème de la famine :

a) Mémoriser les demandes dans 1 file pour les traiter selon la politique FIFO ;
b) Limiter le temps d'allocation de chaque ressource ;
c) Augmenter progressivement la priorité d'1 processus avec le temps d'attente (si la priorité est 1 critère d'ordonnancement).

Exercice 4 : (6 points) Répondez par **VRAI** ou **FAUX** :

Q1	En multiprogrammation, les tâches ne peuvent pas entrer en concurrence pour demander des ressources dont ils ont besoin.	**Faux**
Q2	La solution « masquage des ITs » pour le problème de la Section Critique est dangereuse en mode noyau.	**Faux**
Q3	Le paradigme des Lecteurs-Ecrivains modélise la compétition cohérente.	**Vrai**
Q4	Un changement de contexte peut interrompre une instruction atomique.	**Faux**
Q5	Il est primordial de maîtriser les bons comportements et les bonnes méthodes de programmation et d'utilisation des processus concurrents.	**Vrai**
Q6	Un outil de type TSL (Test And Set Lock) ne génère pas de l'attente active.	**Faux**
Q7	La valeur initiale d'un sémaphore d'exclusion mutuelle est toujours égale à 1.	**Vrai**
Q8	Un Operating System est un intermédiaire entre un ordinateur et les applications qui utilisent cet ordinateur.	**Vrai**
Q9	Les sémaphores sont considérés comme des outils de bas niveau.	**Vrai**
Q10	Un THREAD est un flot d'exécution dans le code du processus, doté d'un CO, de registres système et d'une pile qui lui sont propres.	**Vrai**
Q11	Un sémaphore d'exclusion mutuelle est un sémaphore binaire.	**Faux**
Q12	Les systèmes d'exploitation conventionnels qui sont multiprogrammés dès le départ continuent à se développer et sont de plus en plus souvent multiprocesseurs.	**Vrai**