Institut Supérieur d'Informatique

Université de Tunis el Manar

TD5 : Synchronisation des processus avec des Sémaphores (2)

Systèmes d'Exploitation Avancés – 1ère année ING.

Année Universitaire : 2011/2012

MME. LILIA SFAXI

TD5 : Synchronisation des processus avec des Sémaphores (2)

Systèmes d'Exploitation Avancés

Exercice 1 : Tirage d'examens

On se propose de modéliser le système de tirage d'examen mis en place à l'ISI. L'ISI met en place une salle réservée pour le tirage d'examens (où sont faites les photocopies du sujet d'examen), dans laquelle opère un seul agent, Mr. K. Pour garantir la confidentialité du sujet, seuls sont présents lors du tirage, l'agent K et l'enseignent responsable du module, Mr. E.

On supposera les hypothèses suivantes :

- A tout moment de la journée, l'agent est ou bien en train de tirer un examen (un seul examen à la fois), ou bien en attente non active d'un nouvel examen à tirer.
- Un enseignant qui souhaite faire le tirage de son sujet d'examen, doit se présenter devant la salle de tirage :
  - Si l'agent K est libre (c.à.d en attente d'un nouveau sujet à tirer), l'enseignant et l'agent procèdent au tirage.
  - Si l'agent K est déjà occupé à tirer un autre sujet, l'enseignant :
    - S'il trouve une chaise libre (il y a quatre chaises en tout) devant la salle de tirage, attend l'agent K (attente non active).
    - Si toutes les chaises sont occupées, l'enseignant préfère alors partir et revenir plus tard (on considérera dans ce cas qu'il est sorti du système)

  Il faut remarquer que les quatre chaises devant la salle de tirage ne peuvent être occupées que par des enseignants qui veulent faire des tirages des sujets d'examens.
- Une fois le tirage d'un sujet d'examen fini, l'enseignant responsable du module part (sort du système). Pour l'agent K, si d'autres enseignants attendent (sur les chaises), l'un d'entre eux procède avec l'agent K au tirage de son examen. Sinon, l'agent K se met en attente d'un nouvel examen à tirer.

On se propose de modéliser ce système en utilisant un processus agentK et un autre processus EnseignantE.

Correction:

Sémaphore	**Rôle**	**Valeur initiale**
enseignant	Indique s'il y a des enseignant en attente	0
agentK	Indique si l'agent K est accessible ou non	0

mutex	Pour protéger la modification du nombre des enseignants en attente	1

```
// Agent K
void AgentK() {
        while(1){
                down(&enseignant);
                down(&mutex);
                enAttente --;
                up(&agentK);
                up(&mutex);
                tirer_examen();
        }
}
```

```
// Enseignant E
void EnseignantE() {
        while(1){
                down(&mutex);
                if (enAttente < 4){
                        enAttente ++;
                        up(&enseignant);
                        up(&mutex);
                        down(&agentK);
                        tirage();
                }else{
                        up(&mutex);
                }
        }
}
```

Exercice 2 : Trains

Deux villes A et B sont reliées par une seule voie de chemin de fer. Les trains peuvent circuler dans le même sens de A vers B ou de B vers A. Mais, ils ne peuvent pas circuler simultanément dans les sens opposés. On considère deux classes de processus: les trains allant de A vers B (Train AversB) et les trains allant de B vers A (Train BversA). Ces processus se décrivent comme suit :

Train AversB :	*Train BversA :*
Demande d'accès à la voie A;	Demande d'accès à la voie B;
Circulation de A vers B;	Circulation de B vers A;
Sortie de la voie par B;	Sortie de la voie par A;

Représenter les programmes des deux types de processus.

Correction :

On propose les variables suivantes :

- **NbAB** : Le nombre de trains circulant de A vers B.
- **NbBA** : Le nombre de trains circulant de B vers A.

- **mutex1** : un sémaphore sur la variable NbAB.
- **mutex2** : un sémaphore sur la variable NbBA.
- **autorisation** : un sémaphore pour l'accès à voie du train.

//Demande d'accès par AversB down(mutex) Si (NbAB = 0) down(autorisation) NbAB++ up(mutex) *//Sortie de la voie par B* down(mutex) Si (NbAB = 1) up(autorisation) NbAB -- up(mutex)	*//Demande d'accès par BversA* down(mutex) Si (NbBA = 0) down(autorisation) NbBA++ up(mutex) *//Sortie de la voie par B* down(mutex) Si (NbBA = 1) up(autorisation) NbBA -- up(mutex)

Exercice 3 : Poste (DS 2010-2011)

Le tri du courrier à la poste se fait en deux étapes : la première dans le bureau de poste régional, et la deuxième dans le bureau de poste principal. On dénombre quatre bureaux régionaux et un bureau principal. Dans chaque bureau de poste régional, trois agents procèdent au tri (fonction *trier_courrier_régional*). Si l'un des agents termine avant les autres, il doit les attendre (fonction *attendre*). Une fois que tous les agents ont terminé, un unique agent doit être désigné pour se rendre au bureau de poste principal (fonction *designer_transporteur*). Au bureau de poste principal, les quatre agents régionaux désignés pour transporter le courrier doivent procéder au tri final (fonction *trier_courrier_principal*). Le courrier n'est enfin expédié (fonction *expédier_courrier*) que si tous ces agents ont fini leur tri. L'expédition sera faite par un unique agent (choisi par la même fonction *designer_transporteur*). Un processus représente un agent de la poste. On se propose de synchroniser ces processus avec les sémaphores. On définit les constantes suivantes :

Constante	Valeur	Rôle
nb_reg	3	Nombre de fonctionnaires dans un bureau régional
nb_princ	4	Nombre de fonctionnaires dans un bureau principal

Chaque réponse doit être précédée par une partie initialisations qui doit définir les variables partagées et les sémaphores utilisés.

1. Donner le code de la méthode *attendre (num, nbAgentTotal)* qui va permettre à un agent de numéro *num* d'attendre les autres s'il n'est pas le dernier (le nombre d'agents total à attendre est *nbAgentTotal*).

```
Initialisations :
nbAgent = 0;
semaphore mutex = 1;
semaphore rdv = 0;
attendre (num, nbAgentTotal){
        down(mutex);
        nbAgent++;
        if ( nbAgent < nbAgentTotal){
              up(mutex);
              down(rdv);
        }else{
              up(mutex);
              for ( int i = 0; i < nbAgentTotal ; i++){
                    up(rdv);
              }
        }
```

2. Donner le code de la méthode *désigner_transporteur(num, nbAgentTotal)* qui va choisir un unique agent comme transporteur. L'agent choisi sera le dernier agent parmi les *nbAgentTotal* agents ayant appelé la fonction *désigner_transporteur*. Les agents qui ne sont pas choisis quitteront la poste (fonction *exit()*).

```
Initialisations :
nbAgent = 0;
semaphore mutex = 1;
désigner_transporteur (num, nbAgentTotal){
        down(mutex);
        nbAgent++;
        if ( nbAgent < nbAgentTotal){
              up(mutex);
              exit( );
        }else{
              up(mutex);
        }
}
```

3. Donner le code de la fonction principale agent*(num)* qui définit le comportement de chaque agent.

```
Initialisations :
nbAgent = 0;
semaphore mutex = 1;
semaphore rdv = 0;
agent (num){
        trier_courrier_régional();
        attendre(num, nb_reg);
        désigner_transporteur(num, nb_reg);
        trier_courrier_principal();
        attendre(num, nb_princ);
        désigner_transporteur(num, nb_princ);
        expédier_courrier();
}
```