TD : Les Sémaphores

L'objectif de ce TP est de mettre en pratique l'utilisation des sémaphores pour le partage de ressources entre plusieurs processus.

Exercice 1 :

Bien que le système Unix propose des primitives C pour traiter les sémaphores, on se propose dans cet exercice de définir les fonctions P et V en utilisant un tube (pipe). En effet, comme vous l'avez vu dans les TPs précédents, la lecture dans un tube vide est bloquante pour un processus. On peut donc imaginer le scénario suivant : soit une ressource accessible par au plus n processus à la fois et soit P un processus désirant y accéder. Tous les processus en exécution ont en commun un même tube T. Avant d'accéder à la ressource, P essaie de lire un caractère dans le tube T :

- si le tube T est vide, cela signifie que la ressource n'est pas disponible et P est bloqué,
- sinon la ressource est disponible, P peut y accéder et retire un caractère du tube pour le signaler,
- tout processus qui libère la ressource écrit un caractère dans le tube,
- au début le tube est rempli avec n caractères.

A partir de la définition : `typedef int Semaphore[2]`, écrivez les fonctions suivantes :

- `initSem(Semaphore S, int N)` qui initialise le tube S avec N caractères quelconques,
- `P(Semaphore S)` qui lit un caractère dans S,
- `V(Semaphore S)` qui écrit un caractère quelconque dans S.

Exercice2 :

On désire simuler à l'aide des sémaphores l'utilisation d'une unique piste d'aéroport. A chaque instant il ne peut y avoir qu'un avion qui atterrit ou qui décolle. Votre programme acceptera deux paramètres : le nombre d'avions devant décoller et le nombre d'avions souhaitant atterrir. Votre processus devra créer simultanément autant de processus fils que d'avions à gérer. Pour effectuer la simulation chaque processus avion commencera par attendre un temps aléatoire (entre 1 et 10 secondes) et affichera un message signalant qu'il souhaite atterrir ou décoller ainsi que son PID. On considèrera que le temps de décollage est de 5 secondes et celui d'atterrissage de 3 secondes. Un message devra signaler qui libère la piste à un instant donné.

Exemple de sortie :

```
Vol numero 3360 souhaite atterrir
     Vol numero 3360 en phase d'atterrissage
Vol numero 3359 souhaite atterrir
```

```
Vol numero 3361 souhaite atterrir
          Piste liberee par vol numero 3360
     Vol numero 3361 en phase d'atterrissage
Vol numero 3362 souhaite decoller
          Piste liberee par vol numero 3361
     Vol numero 3362 en phase de decollage
Vol numero 3363 souhaite decoller
          Piste liberee par vol numero 3362
     Vol numero 3363 en phase de decollage
          Piste liberee par vol numero 3363
     Vol numero 3359 en phase d'atterrissage
          Piste liberee par vol numero 3359
```

Exercice 3 :

L'objectif de cet exercice est de simuler le fonctionnement d'une mine d'or. Cette mine est composée de mineurs, de pioches, de pelles et d'une seule caisse pour ranger l'or. Chaque mineur agit de la façon suivante :

(1) Il prend une pelle et une pioche.
(2) Il travaille entre une et cinq heures (des secondes dans la simulation) et extrait entre 1 et 1000 grammes d'or.
(3) Il rend sa pelle et sa pioche.
(4) Il ajoute l'or qu'il a extrait à la caisse.

Plusieurs mineurs peuvent travailler en même temps. Dans cet exercice nous ne simulerons pas le remplissage de la caisse d'or.

Ecrire un programme C qui simule le fonctionnement de la mine sans la gestion de la caisse d'or. Votre programme devra créer autant de processus fils qu'il y a de mineurs.

Rappel : Pour que le père attende la fin de ses fils, vous utiliserez le fait que la fonction `waitpid(-1,NULL,0)` retourne -1 quand un processus n'a plus de fils.

Voici un exemple d'exécution pour 4 mineurs, 1 pelle et 1 pioche :

```
#./mineOr 2 1 1
cr\'eation du mineur 1 (1656)
Le mineur 1 attend 2 heures avant de travailler.
cr\'eation du mineur 2 (1657)
Le mineur 2 attend 1 heures avant de travailler.
Le mineur 2 prend une pelle et une pioche
Le mineur 2 commence \'a travailler pour 4 heures.
Le mineur 2 a extrait 139 g d'or
Le mineur 2 rend une pelle et une pioche
Le mineur 1 prend une pelle et une pioche
```

```
Le mineur 1 commence \‘a travailler pour 2 heures.
Le mineur de pid 1657 a fini.
Le mineur 1 a extrait 248 g d’or
Le mineur 1 rend une pelle et une pioche
Le mineur de pid 1656 a fini.
Fin du travail des mineurs.
```

Exercice 4 :

Dans l’exercice précédent nous avons modélisé une mine d’or avec des processus parallèles, nous allons y ajouter la gestion de la caisse d’or. La gestion de la caisse d’or (représentée par un entier) est faite en utilisant la mémoire partagée.

Modifier le programme précédent pour que le processus père crée un segment de mémoire partagé de 4 octets. Chaque mineur pourra alors ajouter la quantité d’or extrait à l’or déjà présent dans la caisse. (Attention, il faudra vous assurer qu’un seul mineur ne met à jour la caisse en même temps).

A la fin de l’exécution, le processus père devra afficher le poids total de l’or contenu dans la caisse.

Voici un exemple d’exécution pour 2 mineurs, 1 pelle et 1 pioche :

```
#./mineOr 2 1 1
cr\’eation du mineur 1 (1656)
Le mineur 1 attend 2 heures avant de travailler.
cr\’eation du mineur 2 (1657)
Le mineur 2 attend 1 heures avant de travailler.
Le mineur 2 prend une pelle et une pioche
Le mineur 2 commence \‘a travailler pour 4 heures.
Le mineur 2 a extrait 139 g d’or
Le mineur 2 rend une pelle et une pioche
Le mineur 2 met 139 g d’or dans la caisse
Le mineur 1 prend une pelle et une pioche
Le mineur 1 commence \‘a travailler pour 2 heures.
Le mineur de pid 1657 a fini.
Le mineur 1 a extrait 248 g d’or
Le mineur 1 rend une pelle et une pioche
Le mineur 1 met 248 g d’or dans la caisse
Le mineur de pid 1656 a fini.
Fin du travail des mineurs, la caisse contient : 387 g d’or.
```