HMIN105M - IPC : Mémoires partagées et sémaphores
Durée moyenne : 2 séances de 3h

1 Gestion d'un parking

Le but de cet exercice est de réaliser une simulation de gestion d'un parking. Le parking dispose d'un nombre de places limité, initialement toutes disponibles. Des voitures peuvent se présenter à différentes bornes pour demander un ticket d'accès au parking. Si au moins une place est libre, un ticket est délivré et le nombre de places libres est décrémenté. Si le parking est complet, un message d'erreur est délivré. Enfin, des voitures peuvent quitter le parking en libérant les places qu'elles ont occupées.

Concrètement, le nombre de places libres sera stocké dans un segment de mémoire partagée, accessible par un programme représentant une borne (simulant par la même occasion les voitures qui arrivent) et un programme simulant les sorties de voitures. Un programme supplémentaire sera responsable de la mise en place des objets IPC nécessaires au bon déroulement de la simulation. Tous les programmes sont indépendants (sans relation de parenté).

Le programme de simulation d'une borne de distribution des tickets et le programme simulant la sortie de voitures sont des programmes capables de lire et de mettre à jour le nombre de places disponibles. Les algorithmes suivants sont des exemples simplifiés de fonctionnement d'une borne. Il est possible de proposer d'autres solutions pour créer d'autres variantes.

```
1  tant_que vrai faire
2    si nombre_de_places > 0 alors
3      afficher "Demande␣acceptée"
4      décrémenter nombre_de_places
5      afficher "Impression␣ticket"
6      afficher nombre_de_places
7    sinon
8      afficher "Pas␣de␣place"
9    dormir 2 seconde
10 fin_tant_que
```

```
1  tant_que vrai faire
2    dormir 1 seconde
3    si nombre_de_places > 0 alors
4      afficher "Demande␣acceptée"
5      décrémenter nombre_de_places
6      afficher "Impression␣ticket"
7      afficher nombre_de_places
8    sinon
9      afficher "Pas␣de␣place"
10 fin_tant_que
```

1. Montrer qu'il est nécessaire de mettre en place une exclusion mutuelle entre l'ensemble des processus (bornes et sortie de voitures).
2. Proposer une solution à l'aide de sémaphores et écrire le schéma algorithmique détaillé de l'ensemble des programmes de la simulation.
3. Implémenter ces algorithmes et s'assurer du bon fonctionnement de la simulation. Testez votre réalisation en exécutant plusieurs processus bornes et plusieurs processus voitures.

2 Rendez-Vous

Il est souvent nécessaire de réaliser un rendez-vous entre processus. Pour lancer un jeu à plusieurs joueurs par exemple, pour synchroniser des calculs, etc.

Proposer une solution utilisant un sémaphore et permettant à n processus d'attendre jusqu'à ce que tous les processus soient présents et qu'ils soient arrivés à un point déterminé dit *point de rendez-vous* de leur code, ceci avant de poursuivre leur exécution.

Implémenter cette solution, en affichant la valeur du sémaphore à chaque arrivée d'un processus au point de rendez-vous (utiliser *semctl()*).

3 Traitement synchronisé

On envisage le traitement parallèle d'une image vu au TD précédent, mais cette fois, ce traitement est à effectuer par plusieurs programmes. Chaque programme a un rôle déterminé. Par exemple, un premier programme pourrait faire du lissage, un second, des transformations de couleurs ou de l'anti-crénelage, etc. Enfin, ces traitements doivent se faire dans un ordre bien déterminé.

Pour pouvoir réaliser les différents traitements en parallèle, l'idée est de diviser l'image en sous-ensembles ordonnés de points (pixels) et de permettre à chaque programme de travailler sur un sous-ensemble (appelé zone) différent. Le travail doit alors se faire de la manière suivante :

- chaque programme doit traiter, dans l'ordre, toutes les zones de l'image,
- avec garantie d'exclusivité : pendant qu'un programme travaille sur une zone, aucun autre programme ne doit pouvoir accéder à cette zone en même temps,
- sur toute zone, les différents traitements doivent se faire dans un ordre déterminé : le programme P_1 doit passer en premier pour traiter cette zone, puis P_2, puis P_3, etc.

1. On se limite d'abord à deux traitements (donc deux programmes). Proposer une solution permettant un fonctionnement correct. L'image sera stockée dans un segment de mémoire partagée.

2. Pour passer à trois traitements (et plus), quelle solution proposez vous ?

3. Implémenter progressivement vos algorithmes, en simulant un temps de travail aléatoire pour chaque traitement. Générer un temps de travail suffisamment long, de sorte à pouvoir corriger les éventuelles erreurs et à montrer que la protection mise en place fonctionne.

4 Exercice optionnel : Koh-Lanta : l'eau précieuse ...

Cet exercice propose de réaliser une simulation d'un jeu inspiré du jeu dans lequel les candidats de Koh-Lanta doivent remplir un seau d'eau, à l'aide d'une moitié de bambou percé.

Pour cette simulation, deux équipes, constitués de 4 joueurs, vont devoir remplir le seau à l'aide des bamboux qu'il faudra remplir dans des bassines pleines d'eau. Chaque bassine est située à 5 mètres du joueur l'utilisant. Le but est de remplir le seau en premier.

Pour chaque équipe, la règle est la suivante :

Au top, deux joueurs courrent, chacun ayant un bambou à la main, vers leur bassine respective. Ils récupérent l'eau en bouchant les trous pour éviter les fuites, reviennent, versent l'eau dans le seau et passent leur bambou aux deux autres joueurs en attente. Une fois les deux bamboux remis à ces deux derniers (pas un avant/après l'autre), ils peuvent jouer leur tour de la même manière. Le jeu se poursuit ainsi jusqu'à ce qu'il se termine (une équipe a rempli le seau).

Cet exercice s'interesse en particulier au problème de synchronisation au sein d'une équipe. Le but est donc de réaliser une simulation du déroulement du jeu de cette dernière. Concrètement, chaque joueur sera représenté par un processus, issu d'un processus père représentant une équipe.

1. Sur papier, proposer un shéma algorithmique pour les processus joueurs. L'accent est à mettre sur la synchronisation : l'attente et le départ de chaque binôme à chaque tour. A ce stade, vous vous contenterez d'une dixaines de tours pour pouvoir arrêter le jeu.
2. Implémenter la solution proposée après l'avoir discuté avec votre chargé(e) de TD/TP. Pour la simulation d'un aller-retour effectué par un joueur à chaque tour, vous pouvez simplement endormir le processus joueur d'un temps aléatoire entre 3 et 10 secondes (un temps suffisant pour observer et comprendre le déroulement de l'exécution). Enfin, assurez vous aussi du bon fonctionnement de l'execution sans cet endormissement.
3. Enrichir le shéma algorithmique pour prendre en compte l'évolution du niveau d'eau dans le seau de l'équipe courante, pour suivre l'évolution d'une autre équipe et enfin pour être capable de terminer le jeu quand une équipe gagne. Implémenter le tout et tester votre jeu avec deux équipes.