UNIVERSITÉ DE PARIS-DAUPHINE DE MI2E 2e année

Systèmes d'exploitation (E. Lazard)
Projet avril-mai 2019

Le but du projet est de réaliser un simulateur de sémaphores pour pouvoir tester les solutions des exercices sémaphores vus en TD.

1 Cadre général

On considère trois sections critiques (corps de la lecture, de l'écriture, de l'exécution) pouvant être exécutées chacune par plusieurs processus.

Dans une première phase, l'utilisateur entre, à l'aide d'un fichier, sa solution à base de sémaphores, c'est-à-dire définit les prologues/épilogues pour les trois sections critiques et indique les initialisations des sémaphores. Dans une seconde phase, le programme va simuler le fonctionnement des processus en exécutant aléatoirement une ou plusieurs instructions d'un processus. Si un processus entre en section critique, le programme le signalera comme nouvelle situation autorisée.

2 Première phase : lecture du fichier

L'utilisateur va fournir au programme sa solution à base de sémaphores dans un fichier. Ce fichier texte est composé d'au plus huit parties qui apparaissent dans un ordre quelconque : Initialisations, Prologue_Lecture, Prologue_Ecriture, Prologue_Exécution, Epilogue_Lecture, Epilogue_Ecriture, Epilogue_Execution, Paramètres, précédées respectivement par les lignes `%IN`, `%PL`, `%PE`, `%PX`, `%EL`, `%EE`, `%EX`, `%PA`.

Les deux parties initialisation (`%IN`) et paramètres (`%PA`) sont obligatoires. Les autres parties peuvent être absentes (et donc vides) ou présentes (mais peuvent également dans ce cas être vides). Le fichier se terminera (ou au moins sa lecture) par une ligne `%FI` (obligatoirement présente). On pourra mettre des lignes vides dans le fichier et on ne tiendra pas compte des espaces (autrement dit, il peut y en avoir sur chaque ligne).

L'utilisateur aura la possibilité de mettre des commentaires dans le fichier en faisant directement commencer la ligne par `#`.

La partie « Initialisations »

Dans cette partie on trouve la déclaration des sémaphores utilisés. Il y a 26 sémaphores disponibles chacun identifié par une lettre de `'A'` à `'Z'`.

Pour chaque sémaphore qu'il souhaite déclarer, l'utilisateur ajoute dans cette partie du fichier une ligne de la forme :

```
S=valeur
```

où `S` est le nom du sémaphore (de `'A'` à `'Z'`) et `valeur` un nombre strictement positif.

La partie « Paramètres »

Dans cette partie se trouvent les paramètres de la simulation : nombre de processus en lecture (`L`), nombre de processus en écriture (`E`), nombre de processus en exécution (`X`) et nombre de simulations (`N`).

Pour chacune de ses déclaration, on fera suivre la lettre par '=' et le nombre souhaité (voir exemple plus loin).

Les parties « Prologues/Epilogues »

Dans ces six parties, l'utilisateur indique les instructions composant les prologues/épilogues, à raison de une instruction par ligne. Une instruction peut être soit la prise d'un sémaphore (P), soit sa libération (V). La syntaxe de ces instructions sera :

`P(S)`

ou

`V(S)`

où `S` sera le nom du sémaphore (de `'A'` à `'Z'`).

Exemples

Le corrigé de l'exercice 8 des TD pourrait s'écrire :

```
# Voici le corrigé de l'exercice 8
%IN
 L=2
 E=1
%PA
 L=10
 E=10
 N=500
%PL
 P(L)
%EL
 V(L)
%PE
 P(E)
 P(L)
%EE
 V(L)
 V(E)
%FI
```

Les prologues et épilogues sont donnés, l'initialisation indique 2 sémaphores, `L` et `E`.

Chaque simulation se fera avec 10 processus en lecture et 10 en écriture, et il y aura 500 simulations.

3 Seconde phase : simuler

Dans cette seconde phase, on vous demande de simuler l'exécution des processus avec les instructions précédemment entrées.

Dans le fichier, dans la partie paramètres, se trouvent le nombre de processus essayant d'effectuer les sections critiques de lecture/écriture/exécution. Une fois ces trois valeurs récupérées, le programme va simuler l'exécution de tous ces processus. Il ne s'agit donc pas de lancer de véritables processus sur le système mais bien de les simuler, ainsi que les sémaphores.

La simulation

Une simulation va consister à effectuer une boucle dans laquelle on exécute une ou plusieurs instructions d'un processus. Plus précisément, on choisit aléatoirement au début de la boucle un

processus parmi tous ceux existants, non terminés (il y en a au début autant que la somme des trois valeurs indiquées dans la section paramètres) et pas en attente sur un sémaphore, et on tire aléatoirement un nombre d'instructions entre 1 et 7 à exécuter. On simule ensuite leur exécution. Cette boucle se terminera lorsque tous les processus auront fini d'exécuter leur épilogue.

Les instructions

Puisque les processus exécutent les instructions chargées lors de la première phase, il y en a de trois types possibles :

- `P(S)` : le processus essaie de prendre le sémaphore `S` :
  - si `S` est strictement positif, on le décrémente et le processus peut poursuivre son exécution ;
  - si `S` est nul, le processus est mis en attente sur ce sémaphore et on peut tout de suite passer à un autre processus (c'est-à-dire recommencer une itération de la boucle).
- `V(S)` : le processus libère le sémaphore `S` :
  - si aucun processus n'est en attente sur ce sémaphore, on l'incrémente ;
  - s'il y a au moins un processus en attente, on le libère (choisi aléatoirement s'il y en a plusieurs en attente).

  Dans les deux cas le processus poursuit son exécution.
- Section critique : après avoir exécuté la dernière instruction d'un prologue, le processus entre dans sa section critique. Les instructions composant cette section critique n'ont bien sûr aucune importance et on considérera que chaque section critique dure 40 instructions, ceci afin d'éviter qu'un processus ne ressorte immédiatement de sa section critique et de permettre ainsi à d'autres processus d'entrer également en section critique.

Le programme devra simuler ces instructions, c'est-à-dire gérer lui-même les différents sémaphores ainsi que tenir à jour une liste des processus en attente.

Les situations autorisées

Le but du projet est d'obtenir la liste des « situations autorisées », c'est-à-dire des combinaisons de processus en section critique. Par exemple, pour l'exercice 8, les situations autorisées sont :

- un processus en lecture ;
- un processus en écriture ;
- deux processus en lecture ;
- un processus en lecture, un processus en écriture.

On espère obtenir les mêmes résultats avec la simulation.

Lors de la simulation, chaque fois qu'un processus entre en section critique ou sort de section critique, une nouvelle situation autorisée apparaît. Comme une même situation autorisée peut arriver plusieurs fois lors de la simulation, il ne faut pas les afficher directement mais plutôt les mémoriser pour afficher à la fin de la simulation les différentes possibilités.

Une simulation ou plusieurs ?

Une simulation se déroule de la façon indiquée plus haut mais elle ne donnera probablement pas toutes les situations autorisées. En effet, le résultat obtenu dépend des tirages aléatoires gérant l'exécution des processus.

Pour essayer d'obtenir le maximum de situations autorisées, on effectuera plusieurs simulations avant d'afficher l'ensemble des situations autorisées obtenues. Le nombre total de simulations nécessaires sera celui indiqué dans la section paramètres.

4 Votre projet

Règle habituelle : la discussion entre groupes est autorisée, le partage direct du code est interdit.

Le projet est à réaliser en binôme dans le langage de votre choix et devra m'être envoyé pour pouvoir être testé au Crio Unix. Il sera à rendre avant le 28 mai 01h00 sur l'espace MyCourse dédié. Une fois votre fichier envoyé, vous devez avoir un écran de confirmation avec votre fichier et la date de l'envoi. Le format de rendu est une archive au format ZIP contenant :

- Le code-source de votre projet (éventuellement organisé en sous-dossiers) ;
- un répertoire *docs* contenant :
  - une documentation pour l'utilisateur *user.pdf* décrivant à un utilisateur quelconque comment se servir de votre projet,
  - une documentation pour le développeur *dev.pdf*, devant expliquer les choix effectués, les avantages et inconvénients de vos choix, expliquer les algorithmes, indiquer quelles ont été les difficultés rencontrées au cours du projet ;
- un exécutable C ou un « exécutable » Java/Python dont le nom sera `semaphore`, `Semaphore` ou `semaphore.py`.

L'archive aura pour nom `Nom1Nom2.zip`, où `Nom1` et `Nom2` sont les noms des membres du binôme par ordre alphabétique. L'extraction de l'archive devra créer un dossier `Nom1Nom2` contenant les éléments précisés ci-dessus.

Notation

La note de projet tiendra compte de :

- la qualité du programme (correction des résultats, lisibilité du code, facilité de mise en œuvre, robustesse, ergonomie, présentation des résultats...) ;
- la qualité du rapport (description détaillée du programme, explication des choix algorithmiques, discussion des limites de la simulation, améliorations possibles...).

Tests

Les projets seront testés sur des fichiers identiques à celui donné ci-dessus (aussi mis sur MyCourse).

Afin de faciliter les tests, l'exécutable prendra le nom du fichier à tester en argument, effectuera les simulations et affichera les résultats. Une fois la commande lancée, l'utilisateur ne doit pas devoir entrer lui-même quoi que ce soit au clavier.

En fonction du langage, voici trois exemples de commande lançant l'exécution du programme sur le fichier de données `exo8.txt` :

```
$ ./semaphore exo8.txt
$ java Semaphore exo8.txt
$ python3 semaphore.py exo8.txt
```