Système d'Exploitation - L3, S6
TP 2 - Gestion des Processus

L'objectif de ce TP est d'implémenter un système de gestion des processus, une gestion simplifiée empruntée aux simulateurs nachos et moss. Dans ce TP on va s'intéresser aussi bien à l'ordonnancement qu'à la communication entre processus. Pour simplifier, un programme sera vu comme un tableau de mots (non fixé a priori), chaque mot étant composé de deux champs : le premier indique si l'instruction est valide (ou autorisée) et le deuxième si elle est bloquante ou pas. Lorsqu'une instruction est bloquante, le processus devrait être mis dans la liste des processus en attente. Lorsqu'un processus est en attente, il faudra faire la différence entre l'attente bloquante d'une ressource (on n'a pas pour le moment à gérer les ressources) ou l'attente de la terminaison d'un processus. On gérera le réveil des processus en attente dans les questions suivantes.

1 Définition d'un Processus

Un processus sera identifié par son programme, un numéro dans la table des processus, le numéro de son père, la liste des numéros de ses fils. On incluera également un entier `pt_pile` indiquant la première instruction à exécuter lorsque c'est le processus courant. Pour l'instant, nous ne gérons pas la mémoire et on peut alors supposer que le programme est stocké dans l'objet processus. Ainsi, le `pt_pile` servira pour stocker le contexte d'un processus (réduit pour le moment à l'index où commencer l'exécution lorsqu'un processus est réveillé). Il faudra aussi gérer l'état d'un processus.

Exercice 2.1 *Module processus*
Dans le module **processus**, nous gérons : la création d'un processus, la sauvegarde du contexte d'un processus, le placement d'un processus dans la liste des processus en attente, des processus zombies, des processus prêts à s'exécuter ou en exécution. Vous devrez également gérer dans ce module la fin d'un processus (suite à une instruction autorisée ou pas), ainsi que les fonctions d'attente de processus. On supposera que l'on dispose toujours d'un processus 0 qui est l'ancêtre de tous les processus (et qui hérite en particulier des processus orphelins). Pour faire simple, lors de la création d'un processus, on spécifie en paramètre le numéro de son père.

1. Implémentez le module **processus**.
2. Testez votre module dans un fichier séparé (nommé par exemple **test-processus.c**).

2 Ordonnancement

On va dans cette section s'intéresser à l'exécution des processus et à l'ordonnancement. Commençons d'abord par gérer les interruptions. On va supposer qu'il y a deux types d'interruption : une interruption signalant la fin d'un processus et une interruption signalant la fin d'un appel bloquant à une ressource. A chaque instant on dispose d'une liste d'interruptions et à chaque cycle d'horloge (dans notre cas à chaque tour de boucle dans l'exécution des

processus) il faudra regarder cette liste d'interruptions et décider quoi faire. Pour la gestion des interruptions, on va disposer d'une fonction **gestion-interrupt** qui décide de la stratégie. Pour commencer, on choisit la stratégie suivante :

1. Si l'interruption concerne le processus en cours, on la traite.
2. Si l'interruption concerne un autre processus :
   (a) Si c'est l'attente de la fin d'un processus fils, on réveille le processus (s'il n'attendait que ce dernier) et on place le processus dans la liste des prêts à s'exécuter. On n'oublie pas d'informer qu'il faudra libérer les ressources allouées au fils concerné lors de la prochaine exécution du processus.
   (b) Si c'est l'attente bloquante d'une ressource (dans notre modèle il n'y en a qu'une seule par processus), on place le processus concerné dans la liste des prêts à s'exécuter et on informe sur le fait que l'instruction courante n'est plus bloquante. Pour éviter lors de la mise en exécution de ce processus que l'opération pour laquelle il était en attente soit à nouveau considéré bloquant, on peut supposer que l'on dispose d'un `registre` dans la machine qui informe de ce fait (il faudra alors modifier la structure processus pour enregistrer ce registre lors du changement de contexte).

N'oubliez pas qu'il peut y avoir des remontées de fin de processus sans qu'on n'ait pour le moment rencontré une instruction d'attente de fin de processus du père. Il faudra alors les stocker car non encore traitées.

Exercice 2.2 *Interruptions*
Ecrire le module **interrupt** gérant les interruptions. Pour la gestion des interruptions, on appliquera la règle FIFO.

On s'intéresse maintenant à l'exécution et à l'ordonnancement. Ces deux fonctions seront intégrées aux modules **machine** et **ordonnanceur** respectivement. Pour le moment, une machine est réduite à un compteur ordinal, au registre expliqué plus haut et à un pointeur sur un processus (le processus courant). Pour une première implémentation de l'exécution d'un processus, à chaque cycle d'horloge, on exécute les instructions suivantes :

1. Si le temps CPU imparti au processus courant est terminé, on met le processus courant dans l'état prêt à s'exécuter et on exécute l'ordonnanceur. Ce changement de contexte ne doit se faire que lorsqu'il existe un autre processus (autre que le 0).
2. On lance 1 dé (Pile ou face) et si Pile on traite une interruption ressource parmi celles présentes choisies aléatoirement, et si Face, on regarde parmi les interruptions de fin de processus non encore traitées si on ne peut pas en traiter une.
3. On exécute l'instruction courante du processus courant.
   — Si c'est une instruction autorisée, on fait un affichage « instruction autorisée. » Lorsque c'est l'instruction FIN, on termine le processus (avec tous les événements qui en découlent).
   — Si c'est une instruction non autorisée, on termine le processus (avec tous les évènements qui en découlent) et on affiche un message décrivant la fin du processus.
   — Si c'est une instruction bloquante, alors on lance 1 dé (Pile ou face) et si Pile on enregistre l'interruption attendue et on bloque le processus (avec tous les évènements qui en découlent), et sinon on fait un affichage « instruction bloquante rapide. »

Pour l'ordonnancement, on applique les règles suivantes :

— Lorsqu'un processus est choisi pour s'exécuter, on tire aléatoirement un nombre entre a et b (entiers à fixer par exemple lorsque l'on lance le simulateur) qui détermine le nombre de cycles maximum donné au processus avant d'être potentiellement remplacé par un autre.
— L'implémentation d'un algorithme d'ordonnancement lors de la demande de changement de contexte.
— La libération par le processus 0 de ces processus fils zombies.

Pour l'ordonnancement, implémentez d'abord l'algorithme FIFO pour pouvoir tester votre système.

Exercice 2.3 *Ordonnancement*

1. Implémentez les modules **machine** et **ordonnanceur**.
2. Ecrire un programme **simulateur** qui permet de tester notre simulateur. On peut par exemple supposer qu'on lui donne en entrée un fichier avec la liste des processus à créer et les programmes associés.
3. Modifiez l'algorithme d'ordonnancement en implémentant celui qui favorise le processus semblant nécessiter le moins de CPUs. Pour calculer le temps CPU, on peut faire un calcul basé sur le nombre d'instructions restant, le nombre d'instructions bloquantes du passé (ou leur fréquence) et le nombre d'interruptions traitées.