IMA 3 - T.D. - Programmation Avancée
Piles, Files

1 Ordonnancement par file de priorités

Tout processus a besoin d'accéder au CPU pour pouvoir s'exécuter. *L'ordonnanceur* est l'élément du Système d'Exploitation responsable du choix de l'ordre d'exécution des processus sur le (ou les) CPU. Dans cette section, on s'intéresse à une structure de données permettant d'implémenter une politique d'ordonnancement basée sur les priorités sous Unix.

La politique que nous considérons combine deux mécanismes :

1. **Priorité** : l'ordonnanceur choisit toujours d'exécuter le (un des) processus de priorité maximale qui est en attente. Les processus de même priorité sont exécutés dans leur ordre d'arrivée ;
2. **Tourniquet (round robin)** : lorsqu'un processus s'exécute, il s'exécute au maximum pour un quantum de temps fixé, disons 5ms. A l'issue du quantum alloué, si le processus n'est pas terminé il est préempté, remis en attente, et l'ordonnanceur passe au processus suivant.

Un processus est caractérisé par :
— Son identifiant entier dit *process id* (pid) ;
— Son nom ;
— L'identifiant de l'utilisateur propriétaire ;
— Une priorité d'exécution allant de 0 (plus prioritaire) à 40 (moins prioritaire).

1. Écrivez la structure processus qui représente un processus Unix ;
2. Proposez une (ou plusieurs) structure de données `file_prio` pour gérer la file de priorité de l'ordonnanceur. Comparez différentes solutions possibles ;
3. Écrivez une fonction `ajouter` paramétrée par une `file_prio` et un `processus`, qui ajoute le processus dans la file de priorité ;
4. Écrivez une fonction `suivant`, paramétrée par une `file_prio`, qui renvoie (par valeur de retour ou paramètre) le processus suivant à exécuter.

2 Expressions arithmétiques post-fixées

Vous avez l'habitude de manipuler des expressions arithmétiques sous leur forme *infixe*, c'est-à-dire que les opérateurs apparaissent entre leurs opérandes. Il existe aussi une notation *préfixe* (opérateurs avant leurs opérandes) et *postfixe*

(opérateurs après leurs opérandes). Par exemple, les 3 expressions suivantes produisent le même résultat :

$$\begin{aligned} 2 + 3 * 4 \quad & \text{(infixe)} \\ + \ 2 \ * \ 3 \ 4 \quad & \text{(préfixe)} \\ 2 \ 3 \ 4 \ * \ + \quad & \text{(postfixe)} \end{aligned}$$

Dans la suite, nous allons nous concentrer sur la notation post-fixée.

1. Utilisez une pile pour calculer l'expression suivante : $3\ 4\ 2\ *\ +\ 1\ -$
2. Définissez un type `lexeme` pour représenter les différents composants d'une expression arithmétique. On incluera un lexeme `End` désignant la fin d'une expression ;
3. On suppose l'existence de la fonction suivante qui renvoie le prochain lexeme de l'expression en cours d'analyse : `lexeme lexSuivant()`. Ecrire une fonction permettant de calculer le résultat d'une expressions arithmétique postfixée ;
4. Convertissez l'expression suivante en son équivalent postfixé : $1*5+4-2$
5. Ecrire une fonction permettant d'afficher l'expression postfixée correspondant à une expression infixée analysée à l'aide de la fonction `lexSuivant()`.