Université de Caen
U.F.R. de Sciences

Année 2017 - 2018
Devoir maison
A. Lechervy

L3 Informatique

Devoir maison

Systèmes d'exploitation avancés

à rendre le 4 décembre 2017 à 23h59 au plus tard

Ordonnanceur de travaux

1 Présentation

Le but de ce devoir est de réaliser un ordonnanceur de travaux (job scheduler) permettant d'organiser l'exécution de programme en fonction des ressources d'une machine. Vous aurez à réaliser un programme C permettant cette opération et devant être fonctionnel sur les machines en libre-services du département informatique.

Ce projet est à rédiger **en C**. Vous n'avez **pas le droit d'utiliser de bibliothèques extérieures** mis à part les bibliothèques vu en cours.

2 Présentation générale

L'objectif de ce projet est de réaliser un ordonnanceur de travaux (Job scheduler). Afin d'éviter la surcharge d'une machine, l'ordonnanceur exécutera les tâches demandées par l'utilisateur uniquement quand les ressources sont disponible. Il permettra ainsi d'éviter le lancement de trop de processus en parallèle qui surchargerait la machine et pénaliserait l'ensemble des processus.

Le programme que vous réaliserez ressemblera à un terminal classique mais au lieu d'exécuter directement les commandes tapées, il les placera dans une liste d'attente et les exécutera lorsque le nombre de processus actif est inférieur à un seuil fixé. Par exemple pour un seuil de 4, si 4 processus sont en cours d'exécution, une nouvelle commande sera mise en attente jusqu'à qu'un des processus actifs se finisse. Si plusieurs commandes sont en attente, la première arrivée sera le première à être exécutée.

Nous allons également ajouter un timer sur les processus en cours d'exécution, pour que leurs temps d'exécution ne dépasse pas un certain temps. Au delà le processus, sera tué par le système.

3 Travail à faire

3.1 Un point de départ : un mini-shell

1. Commencez par reprendre le code que vous avez réalisé pour le TP 3 (mini shell). Votre mini-shell doit être pleinement fonctionnel. Nous utiliserons ce code comme point de

départ pour écrire l'ordonnanceur.

2. Vous vous assurez que rien ne se passe lorsque la commande est vide ou ne contient que des espaces (aucun processus fils ne doit être créé dans ce cas).

3. Faites en sorte que toutes les commandes s'exécutent en arrière plan (comme si `&` était écrit à chaque fin de ligne).

4. Déplacer le code permettant de lancer la commande tapée dans le mini shell dans un processus fils, dans une fonction `lancerCmd`.

3.2 Une liste des processus actif

1. Nous allons devoir gérer une liste permettant de connaitre les processus actuellement en exécution sur le système. Pour cela créez une variable globale qui contiendra les pid des processus actif sur le système. La dimension de cette liste correspondra au nombre maximal de processus possible actif (par exemple 4). Vous aurez également besoin d'une variable globale indiquant le nombre d'élément remplit dans ce tableau. Elle permettra notamment d'indiquer l'indice de la prochaine case à remplir.

2. Écrivez une fonction `echanger` qui prend en argument l'adresse de deux entier et qui échange leurs valeurs. On utilisera cette fonction pour enlever un élément de la liste de la question précédent. En effet, il suffit d'échanger l'élément à supprimer avec le dernier élément de la liste et de décrémenter le nombre d'élément dans la liste. Cela fonctionne même s'il ne reste qu' un seul élément dans la liste. En effet, vous échangez l'élément avec lui même puis vous décrémenter le nombre d'élément de la liste en le mettant à 0.

3. Faites une fonction de recherche qui permet de trouver l'indice dans la liste précédente d'un PID donné. Si la valeur n'est pas présente, la fonction retournera -1.

4. A chaque création de processus fils, lors de l'exécution d'une commande, ajoutez le pid du fils dans la liste des processus actif. Cette opération doit être faite par le père.

5. En réceptionnant le signal SIGCHLD correspondant à la mort d'un fils, supprimez le fils de la liste des processus actif. Pour connaitre le pid du fils qui vient de mourir, vous pouvez utiliser la valeur de retour de la fonction `wait` (qui sera non bloquante dans ce cas). Faites attention à ne faire que cette opération dans le processus principale et non dans ses fils.

3.3 Une file FIFO des commandes en attentes

1. Nous allons maintenant gérer une file des processus en attente avec des variables globales. Commencez par créer un tableau de chaine de caractère qui contiendra la liste des commandes en attente d'exécution. Nous utiliserons deux indices sur ce tableau : l'indice de début de la file et l'indice de fin. Vous utiliserez aussi une autre variable indiquant le nombre d'élément actuellement dans la file. Nous considérons que si la file est rempli, l'ajout de nouvelles commandes écrasera les commandes les plus ancienne. A chaque ajout d'un élément dans la liste, l'indice de fin s'incrémente. A chaque suppression d'élément l'indice de début s'incrémente. Si l'un des deux indices atteint la limite du tableau, il repars de 0. Pensez à incrémenter et décrémenter le nombre d'élément dans la liste en conséquence.

2. Changez le fonctionnement de votre programme pour qu'au lieu de lancer une commande en tache de fond à chaque commande tapez, votre application l'ajoute à la place dans la liste des commandes en attentes.

3. À chaque itération de la boucle de votre mini shell, lancez la commande en attente la plus ancienne si le nombre de processus actif est inférieur au maximum autorisé. Vous enlèverez la commande de la liste des commandes en attente.
4. À la mort de chaque processus fils associé à l'exécution d'une commande, lancez la commande en attente la plus ancienne si le nombre de processus actif est inférieur au maximum autorisé. Vous enlèverez la commande de la liste des commandes en attente.
5. Affichez après chaque commander écrite par l'utilisateur le nombre de processus en cours d'exécution et en attente. Le simple fait d'appuyer sur entrée dans le mini shell devra afficher ces informations.
6. Vérifiez que le nombre de processus actif ne dépasse pas le maximum autorisé. Vous pouvez utiliser la commande shell `sleep` avec un argument approprié pour simuler un processus long.

3.4 Un timer sur l'exécution des commandes

Nous allons maintenant ajouter un timer sur les commandes pour empêcher une commande de s'exécuter trop longtemps sur le système. Vous fixerez par exemple le temps maximale des commandes à 10s.

Pour réalisez cela vous aller devoir modifier le processus fils lançant la commande à exécuter. Ce processus fils va devoir se dupliquer, d'un coté il fera le lancement de la commande et de l'autre il lancera une alarme de 10s suivi d'un wait et d'un exit. Dans la fonction déclenché par l'alarme, il enverra un signal tuant le processus exécutant la commande. On passera par des variables globale pour passer le PID du processus à tuer.

3.5 Question bonus

1. Assurez vous que la mémoire est correctement libérée dans votre programme et qu'il n'y a pas de fuite mémoire.
2. Ajoutez des commandes de gestion de l'ordonnanceur permettant par exemple de connaitre les commandes en attente, les processus actif avec les commandes quelles exécutent, le temps restante pour ces commandes...
3. Vous pouvez également faire en sorte que la commande `kill` puisse s'exécuter directement sans être mise en attente.
4. Vous pouvez ajouter un gestion des priorités en offrant la possibilité de lancer des commandes plus ou moins prioritaire en indiquant, par exemple, en début de ligne la priorité de la commande tapé.

4 Rendu

Le travail est à réaliser en binôme (ou seul). Vous devrez rendre votre travail en utilisant le système *Devoir* de ecampus. Pour cela, vous devez déposer une archive targz sur ecampus.

Rappel de la date limite de la remise : **4 décembre 2017 à 23h59**.

Dans le cas présent cette archive doit contenir au moins un fichier **noms.txt** contenant les noms des étudiants ou étudiantes composant le binôme. Chaque étudiant(e) doit déposer au

minimum ce fichier, qu'il ou elle soit seul(e), ou en binôme. Les autres fichiers (liste ci-dessous) ne seront déposées qu'une seule fois par binôme.

- les sources de vos programmes et les scripts éventuelles ;
- un fichier **README.txt** indiquant clairement les instructions pour lancer votre programme ;
- un rapport expliquant la manière dont vous avez résolu les problèmes, et donnant le mode d'emploi de votre réalisation ;
- tout autre fichier dont la présence est nécessaire pour faire fonctionner ou comprendre le fonctionnement de l'ensemble de votre réalisation.

5 Évaluation

Le non-fonctionnement du programme sera fortement pénalisé, vous devrez donc respecter scrupuleusement les contraintes du sujet et vous assurer du bon fonctionnement de votre code. Il est donc inutile d'essayer d'introduire des fonctionnalités si ce qui a déjà été fait ne marche pas correctement.

La qualité et le contenu du rapport ainsi que le respect des consignes seront également pris en compte dans la notation.