INF2610
TP 1 : Appels système, processus et threads

Polytechnique Montréal
Auteurs : Antoine Bourassa et Ann-Sophie St-Amand
Automne 2025

Pondération : 10%
Dates de remise : Voir calendrier du cours

Présentation

Ce travail pratique a pour but de vous familiariser avec l'environnement de programmation des laboratoires, les appels système de la norme POSIX liés à la gestion de fichiers et à la gestion des processus et threads (création, attente de fin d'exécution et terminaison).

Les processus et les threads sont souvent utilisés en informatique pour faire des traitements en parallèle. Par exemple, lors de la compilation d'un programme, les systèmes modernes exploitent largement les processus et les threads pour accélérer le travail. Un projet logiciel est généralement composé de nombreux fichiers source indépendants (par exemple en C ou en C++). Chaque fichier peut être compilé séparément, ce qui permet de lancer plusieurs processus ou threads en parallèle afin de traiter plusieurs unités de compilation simultanément. Ensuite, le système d'exploitation répartit cette charge de travail entre les cœurs du processeur, réduisant considérablement le temps nécessaire par rapport à une compilation séquentielle. C'est pourquoi les outils de compilation comme `make` ou `gcc` offrent des options (par exemple `make -j`) qui exploitent le parallélisme pour améliorer la performance. Ce n'est qu'un exemple d'utilisation parmi tant d'autres. Dans ce laboratoire, vous aurez à appliquer ces concepts dans le contexte d'analyse de fichiers de logs serveur.

Prise en charge du laboratoire

Il y a deux options pour préparer l'environnement de travail pour le laboratoire :

1. Utiliser les ordinateurs de laboratoires de Polytechnique.
2. Télécharger et installer VMware.

VMware est un logiciel qui fait la gestion de machines virtuelles (création, copie, suppression, etc.). Il fonctionne sur Windows et sur Mac. Il sera utile tout au long du baccalauréat et pour la suite. Le point négatif est qu'il prend beaucoup de ressources sur un ordinateur. Ceci étant dit, voici les instructions pour télécharger VMware :
`https://www.polymtl.ca/gigl/guides-informatiques#vmware`

La prochaine étape consiste à créer la machine virtuelle en utilisant le fichier fourni sur Moodle dans la section TP1. Une fois la machine virtuelle créée, tu trouveras le dossier du travail pratique directement sur le bureau. Pour te connecter à la machine, utilise le compte suivant :
Nom d'utilisateur : user
Mot de passe : user

Prenez quelques instants pour vous familiariser avec la structure du répertoire sur lequel vous travaillerez durant ce TP. Vous trouverez dans le répertoire courant 4 dossiers représentant chacun une section du TP. Chaque section vous sera présentée plus bas. Il y a aussi un fichier Makefile qui peut être utilisé pour compiler et exécuter le code, mais ne doit pas être changé. Il est conseillé de lire l'énoncé en entier avant de commencer le TP. Il n'est pas demandé de traiter les erreurs éventuelles liées aux appels système. Par contre, si besoin est, à chaque fois que votre programme effectue un appel système (directement ou via une fonction de librairie), vous avez la possibilité d'afficher un message d'erreur explicite en cas d'échec de cet appel système. Pour ce faire, il vous suffit d'utiliser la fonction `perror` après l'appel système (ou l'appel de fonction de librairie). Consultez sa documentation !

Mise en situation

Votre équipe fait partie d'un service de supervision et sécurité informatique responsable de surveiller en continu l'état des serveurs d'une grande organisation. Chaque jour, ceux-ci génèrent plusieurs gigaoctets de journaux système (logs) contenant des informations sur les connexions, les opérations réussies ou échouées, ainsi que les erreurs rencontrées.
Lorsqu'un incident critique survient (panne de service, intrusion potentielle, défaillance matérielle), les analystes doivent être capables de parcourir rapidement ces fichiers volumineux afin de détecter :

- Le nombre d'erreurs critiques (`CRITICAL`, `ERROR`),
- La présence de motifs suspects (`FAILED LOGIN`, `segmentation fault`, etc.),
- Des anomalies dans la structure ou le format des entrées.

Dans ce contexte, il est souvent nécessaire de compter efficacement les occurrences de certains types d'erreurs dans un fichier de logs donné, ou encore de rechercher en parallèle différents motifs. Pour traiter ces volumes de données rapidement, on fait appel à des arbres de processus et à des threads, qui permettent de diviser la charge de travail et de l'exécuter en parallèle.

Dans ce premier TP, vous aurez à :

1. Manipuler des fichiers de logs et en extraire des informations en utilisant des appels système ;
2. Mettre en place un programme qui crée et organise des processus et des threads afin d'effectuer des traitements parallèles ;
3. Comparer les temps d'exécution de vos différentes approches (processus vs threads) ;
4. Analyser la performance de votre programme selon plusieurs métriques définies dans les sections à venir.

1 Appels système

La première section a pour but de vous familiariser avec la machine virtuelle et son OS. Les appels systèmes est le sujet principale du premier exercice. Comme vu en classe, il existe plusieurs appels systèmes dans un language de programmation (par exemple C). Il en existe aussi à même le système d'exploitation (en utilisant des commandes sur un terminal).

Nous vous demandons d'exécuter ces commandes dans un terminal et de sauvegarder les sorties dans un fichier.

1. `ps` est une commande qui donne la liste des processus en cours sur le système d'exploitation. Essayez de rediriger sa sortie vers un fichier `.txt` avec `»`.
2. En créant un fichier avec la commande `touch`, `vi` ou `nano`, c'est aussi un appel système. Essayez de créer un fichier qui s'appelle `section1_2.txt`.

L'exercice est de créer une fonction en C qui fait la lecture d'un fichier (le fichier s'appelle mots.txt et est dans le même répertoire que le code), qui calcule le nombre de mots dans le fichier et qui retourne la somme dans un fichier en écriture qui s'appelera `section2_2.txt`.

2 Arbre de processus

On vous demande d'extraire le nombre d'erreurs `CRITICAL`, `ERROR` et `FAILED LOGIN` de deux fichiers de logs générés par le serveur (logs.txt et logs_2.txt). Vous devez respecter certaines contraintes :

- L'analyse des deux fichiers doit se faire en parallèle ;
- Dans un même fichier, la détection des trois types d'erreurs doit se faire en une seule lecture.

Pour accélérer le traitement, on vous demande de diviser la lecture d'un même fichier en blocs. À partir de l'entrée standard, vous devez entrer un nombre `n` de blocs. Le fichier sera alors divisé en `n` sections, et chaque processus aura à analyser un bloc (Indice : utiliser `lseek` de POSIX). L'idée est d'éviter que chaque processus lise le fichier en entier. Votre programme doit fonctionner avec n'importe quelle valeur de `n` plus grande ou égale à 1.

Vous devez donc bâtir un arbre de processus pour exécuter ces tâches. Lorsque la lecture des deux fichiers est terminée, le processus parent du programme doit écrire dans un fichier de texte `RESULT_PROCESS.txt` les résultats (basez-vous sur votre code de la section 1 au besoin). Chaque processus parent doit attendre la terminaison de ses enfants avant de lui-même se terminer. Aussi, lorsqu'un processus enfant se crée, il doit afficher son PID sur la sortie standard. L'affichage de la sortie standard doit uniquement se faire à la fin du programme, et non au fur et à mesure que les processus sont créés.

```
inf2610@inf2610-virtual-machine:~/Desktop/INF2610-TPs/tp1/solutionaire$ cat bin/RESULT_PROCESS.txt
utils/logs.txt : CRITICAL : 765
utils/logs.txt : ERROR : 765
utils/logs.txt : FAILED LOGIN : 765
utils/logs_2.txt : CRITICAL : 765
utils/logs_2.txt : ERROR : 765
utils/logs_2.txt : FAILED LOGIN : 765
Total CRITICAL : 1530
Total ERROR : 1530
Total FAILED LOGIN : 1530
```

FIGURE 1 – Exemple de sortie du terminal

Question 2.1

Ajoutez à votre rapport un schéma de l'arbre de processus créé. Indiquez directement sur le schéma le rôle de chaque niveau de processus. Intégrez aussi la valeur n de blocs de lecture (0.5 point).

Question 2.2

Exécutez quelques fois de suite votre programme avec une même valeur de `n`. Il est possible que certains PID soient affichés dans un ordre différent d'une exécution à l'autre, et que certains soient affichés dans le même ordre. Expliquez brièvement pourquoi (1 point).

Question 2.3

Il vous était demandé d'afficher les PID de tous les processus seulement à la fin du programme. Expliquez la différence de fonctionnement de l'appel système `write` et de la fonction `printf` de la librairie standard, et justifiez pourquoi vous avez utilisé l'un ou l'autre dans votre code (0.5 point).

Question 2.4

Expliquez comment vous avez fait pour communiquer les valeurs d'un processus enfant vers un processus parent. Votre explication devrait aborder le concept de COW (Copy On Write) (1 point).

3 Arbre de threads

Vous devez refaire le même traitement qu'à la section 2, mais cette fois-ci en utilisant des threads POSIX. La lecture des blocs doit se faire par des threads concurrents (Indice : utiliser `pread` de POSIX).
Cette fois-ci, le processus à la racine du programme doit écrire ses résultats dans un fichier nommé `RESULT_THREADS.txt`.

Question 3.1

Ajoutez à votre rapport un schéma de l'arbre de threads créé. Indiquez directement sur le schéma le rôle de chaque niveau de threads. Intégrez aussi la valeur `n` de blocs de lecture (0.5 point).

Question 3.2

Expliquez pourquoi l'utilisation de la fonction `read` par les threads aurait été problématique avec un seul descripteur de fichier. Expliquez aussi comment `pread` règle ce problème (1 point).

Question 3.3

Expliquez comment vous avez fait pour communiquer les valeurs d'un thread vers son processus parent. Expliquez pourquoi c'est différent de votre solution de la section 2 (1 point).

Question 3.4

Dans quelques semaines, vous allez voir en classe le concept de synchronisation. À l'aide d'une petite recherche en ligne, expliquez brièvement dans vos mots ce qu'est la synchronisation, à quoi ça sert, et pourquoi c'est surtout utilisé avec les threads. N'oubliez pas d'indiquer vos sources (0.5 point).

4 Analyse des performances

Deux ordinateurs peuvent présenter des performances très différentes en matière de parallélisme en raison de leur système d'exploitation et de leurs caractéristiques matérielles. Par exemple, un processeur doté de plusieurs cœurs et d'un grand nombre de threads matériels (hyper-threading) permet d'exécuter plusieurs tâches simultanément, alors qu'un processeur plus ancien à un seul cœur sera limité. De plus, la gestion du parallélisme par l'OS joue un rôle important : certains systèmes optimisent mieux la planification des processus et des threads, la répartition sur les cœurs et l'utilisation de la mémoire. Ainsi, deux programmes identiques peuvent s'exécuter beaucoup plus rapidement sur une machine moderne multicœur avec un OS optimisé que sur un ordinateur moins puissant ou mal configuré.

Question 4.1

Exécutez votre programme de la section 2, puis celui de la section 3, le tout sur la même machine. Chronométrez les deux exécutions (voir la fonction `measure_time()` du fichier timer.c). Commentez la différence de temps d'exécution en utilisant des processus et des threads (0.5 point).

Question 4.2

Exécutez votre programme de la section 2 sur deux machines différentes. Chronométrez les deux exécutions. Commentez la différence de temps d'exécution en abordant les différences matérielles des deux machines (0.5 point).

Question 4.3

Exécutez votre programme de la section 3 sur deux machines différentes. Chronométrez les deux exécutions. Commentez la différence de temps d'exécution en abordant les différences matérielles des deux machines (0.5 point).

Question 4.4

Selon vous, dans quelles situations est-il préférable d'utiliser des processus et dans quelles situations est-il mieux d'utiliser des threads ? Utilisez des exemples pour expliquer votre raisonnement (0.5 point).

Barème

Important : Un programme qui ne compile pas entraine automatiquement une perte de la moitié des points de code attribués à cette section.

Votre rapport est aussi à remettre sur Git.

Section 1	Code	2
	Total	/2
Section 2	Code	4
	Questions	3
	Total	/7
Section 3	Code	4
	Questions	3
	Total	/7
Section 4	Questions	2
	Total	/2
Qualité du code	Total	/2