INF2610
TP 1 : Appels système, processus et threads

Polytechnique Montréal
Autrice : Ann-Sophie St-Amand
Hiver 2026

Pondération : 10%
Dates de remise :

Groupe 01L	17 février à 23:59
Groupe 02L	10 février à 23:59
Groupe 03L	19 février à 23:59
Groupe 04L	12 février à 23:59
Groupe 05L	8 février à 23:59

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, vous trouverez le dossier du travail pratique directement sur le bureau. Pour vous connecter à la machine, utilisez 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 quatre 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. Appliquer vos nouvelles connaissances dans un contexte d'algorithme de tri

1 Appels système (2 points)

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'implémenter la fonction `count_errors()` qui doit compter le nombre d'occurences d'un type d'erreur.

— La fonction `count_errors()` prend en paramètre un numéro de ligne de début et de fin. La fonction doit donc compter le nombre d'erreurs présentes dans ces lignes. Par exemple, si on donne `start_line = 0` et `end_line = 5`, la fonction doit compter les erreurs de la première ligne jusqu'à la sixième ligne inclusivement.
— Vous pouvez considérer que votre fonction va lire un maximum de 1024 octets par appel.
— Le type d'erreur à compter est spécifié avec le paramètre `error_type`.
— La fonction doit lire un fichier donné en argument et écrire le résultat dans un nouveau fichier de texte au nom de votre choix.

Réfléchissez bien aux différents cas limites lors du comptage des erreurs.

Pour tester votre code, il suffit de faire la commande suivante :

```
make run
```

2 Arbre de processus (6 points)

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, qui sont à donner en argument au programme). 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 aussi se faire en parallèle.

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. L'idée est d'éviter que chaque processus lise le fichier en entier. Votre programme doit fonctionner avec n'importe quelle valeur de `n` où $1 <$ `n` $<$ nombre de lignes du fichier. Vous pouvez réutiliser votre fonction `count_errors()` implémentée à la section 1.

Lorsqu'un processus enfant termine de compter les erreurs dans son bloc attribué, il doit écrire son résulat dans un fichier de texte temporaire nommé par son PID. Le processus parent est alors en mesure de lire cette valeur, puis de supprimer le fichier de texte temporaire.

Vous devez donc bâtir un arbre de processus pour exécuter ces tâches. Lorsque la lecture des deux fichiers est terminée, le processus racine du programme doit retourner la somme des résultats dans une structure `Results` (la structure est donnée dans le fichier d'en-tête). Chaque processus parent doit attendre la terminaison de ses enfants avant de lui-même se terminer.

Pour tester votre code, il suffit de faire la commande suivante :

```
make run
```

3 Arbre de threads (4 points)

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`.

Remarque : La mémoire des threads ne fonctionne pas exactement comme la mémoire des processus. Trouvez une manière optimale de partager les données entre différents threads.

Pour tester votre code, il suffit de faire la commande suivante :

```
make run
```

4 Tri fusion parallèle pour l'analyse d'incidents (6 points)

Après l'analyse des fichiers de logs, le système de supervision doit classer les incidents détectés par ordre de priorité afin d'aider les analystes à traiter en premier les événements les plus critiques. Chaque incident est caractérisé par :

- un identifiant unique ;
- un niveau de sévérité (`CRITICAL`, `ERROR` et `FAILED_LOGIN`) ;
- un timestamp indiquant le moment d'apparition de l'incident.

Vous devez implémenter un algorithme de tri fusion parallèle (Merge Sort) utilisant des threads POSIX afin de trier un tableau d'incidents selon les règles suivantes :

1. Les incidents sont triés par niveau de sévérité décroissant.
2. En cas d'égalité, ils sont triés par timestamp croissant (l'incident le plus ancien doit être traité en priorité).
3. La profondeur maximale de votre arbre de tri est donnée par la relation suivante (Indice : utilisez `sysconf(_SC_NPROCESSORS_ONLN)` de la librairie `unistd.h`) :

$$\text{profondeur}_{\max} = \lfloor \log_2 (N_{\text{cœurs}}) \rfloor$$

4. Utiliser la structure `SortTask` pour le traitement.

Le tri fusion est naturellement divisible de manière récursive. Chaque appel récursif peut être exécuté en parallèle à l'aide de threads POSIX, jusqu'à la profondeur maximale définie ci-dessus, afin d'éviter la création excessive de threads et la surcharge du système.

Pour tester votre code, il suffit de faire la commande suivante :

```
make run
```

Remise

La remise se fait obligatoirement en équipe de deux sur la plateforme **GitHub Classroom**.

Une remise en retard ou sur un dépôt Git personnel entraîne automatiquement la note de 0 pour le tp.

Toute modification de la structure initiale du dépôt Git peut entraîner des pénalités.

Les fichiers temporaires doivent être supprimés à la fin de l'exécution, avec la commande suivante :

```
make clean
```

Critères d'évaluation et barème

Important : tout programme qui ne compile pas entraînera automatiquement la perte des points attribués à la section concernée.

Pour ce premier laboratoire, il est demandé d'utiliser seulement les concepts vus dans les trois premières séances de cours (Généralités, Processus, Fils d'exécution).

Section	**Points**
Section 1	/2
Section 2	/6
Section 3	/4
Section 4	/6
Qualité du code	/2
Total	**/20**