INF2610

TP #1: Process Lab

Ecole Polytechnique de Montréal

Hiver 2021 — Durée: 2 heures et 15 minutes

Présentation

Le *Process Lab* a pour but de vous familiariser avec les appels système de la norme POSIX liés à la gestion de processus (création, attente de fin d'exécution, transformation et terminaison). Vous aurez aussi l'occasion d'expérimenter les outils de traçage *strace* et *ltrace* de Linux. **Il est conseillé de lire l'énoncé en entier avant de commencer le TP.**

Attention: Ce cours a une politique bien définie en termes de plagiat. L'archive que vous avez téléchargée sur Moodle a été générée pour votre groupe seulement. Il vous est interdit de partager votre code ou le code qui vous a été fourni, que ce soit en ligne (via un dépôt Git public par exemple) ou avec d'autres étudiants. Votre code sera systématiquement vérifié et le plagiat sera sanctionné. Afin d'éliminer tout doute quant à ce qui peut constituer du plagiat et pour connaître les sanctions, veuillez vous référez àau site de l'école (https://etudiant.polymtl.ca/plagiat).

Prendre en main le lab

Prenez quelques instants pour vous familiariser avec la structure du répertoire sur lequel vous travaillerez durant ce TP. Pour tous les TPs qui suivront, la structure sera similaire. Vous trouverez dans le répertoire courant:

- `processlab.pdf` ➔ L'énoncé du TP;
- `Makefile` ➔ Ce fichier contient les commandes qui vous permettront de compiler votre TP;
- `grade.sh` ➔ Ce script vous permet d'évaluer votre travail. Vous trouverez plus d'informations dans le paragraphe *Evaluer votre travail* de ce document;
- `q1.c`, `part2.c` ➔ Ce sont les fichiers de code que vous modifierez au fur et à mesure des TPs;
- `grader/`, `processlab.c`, `libprocesslab.o`, `libprocesslab.h` ➔ Ce sont les fichiers et répertoires qui permettront de compiler et d'évaluer votre travail. Ils contiennent tout le code qui permet secrètement de faire fonctionner le TP...!

Voici quelques points très importants à retenir:

Tous les TPs du cours se dérouleront sur une plateforme de type Linux, avec des programmes écrits en C. Vos chargés de TP vous aideront en vous présentant cet environnement de travail au tout début de cette séance de TP. N'hésitez pas à faire appel à eux tout au long de la session!

Contrairement à ce que vous avez pu faire dans vos cours précédents, vous utiliserez de simples éditeurs de texte pour modifier les fichiers de code. La compilation se fera séparément, dans une fenêtre de terminal. N'oubliez pas que vous devez recompiler le TP à chaque fois que vous modifiez les fichiers de code, sinon vos modifications n'auront aucun effet! Vous trouverez plus de détails dans le paragraphe *Compiler et exécuter votre TP* de ce document.

☞ Le format de ces TPs est nouveau pour vous autant qu'il l'est pour nous! Ces TPs sont encore en développement et votre avis sera précieux pour le faire évoluer. N'hésitez pas à faire part de vos remarques ou à faire remonter les dysfonctionnements à vos chargés de TP.

Énoncé

Vous trouverez par la suite les instructions précises pour chacune des questions à résoudre. Pour vous aider à gérer votre temps, nous indiquons pour chaque question avec le symbole ◔ le temps qu'un élève finissant le TP dans le temps imparti devrait y consacrer. **Ces indications ne sont que des estimations pour que vous puissiez situer votre progression, alors pas de panique si vous dépassez la durée conseillée!**

Question	Contenu	◔	Barème
1	Appels système et fonctions de librairie	25mn	3.0 pt

Conseil: N'oubliez pas de compiler le TP à chaque fois que vous modifiez les fichiers de code pour pouvoir tester votre programme. Vous pouvez vous servir de l'exécutable `processlab`, qui va notamment exécuter et tester votre solution, en exécutant `./processlab` dans un terminal. Enfin, vous pouvez à tout moment évaluer votre travail grâce aux scripts d'évaluation, comme indiqué dans le paragraphe *Evaluer votre travail* de ce document.

Question 1.1 *(Plutôt `printf` ou plutôt `write`?)* *environ 25mn*

Vous avez vu dans la question précédente qu'il est souvent utile de faire appel à des fonctions de librairie de haut niveau comme `printf`. Dans cette question cependant, vous allez devoir afficher un message en utilisant directement l'appel système `write`, grâce à la fonction de librairie `write` – consultez sa documentation en exécutant `man 2 write` dans un terminal!

Une fonctionnalité importante de `printf` est que cette fonction possède un tampon dans lequel sont stockés temporairement les messages à afficher; ce n'est que quand l'utilisateur demande à imprimer un message contenant un caractère de fin de ligne `\n`, ou quand le tampon est plein, ou quand l'utilisateur fait appel à la fonction `fflush`, que l'appel système `write` est exécuté pour afficher le contenu du tampon au complet. Cela permet de faire l'économie d'appels systèmes trop fréquents quand ce n'est pas nécessaire. Nous allons exploiter ce comportement pour afficher le message suivant:

```
44b014bef71345075b78cb1d6402e4f0 (printed using write)
44b014bef71345075b78cb1d6402e4f0 (printed using printf)
```

Les consignes à respecter sont les suivantes :

- La première ligne du message doit être affichée en utilisant l'appel système `write`, et la deuxième ligne en utilisant `printf`, **mais**...
- ...vous devez faire un premier appel à `printf` **avant** de faire un appel système `write`. Ce premier appel doit contenir la totalité de la ligne à afficher avec `printf`, sauf éventuellement le caractère de nouvelle ligne `\n`.
- N'utilisez pas la fonction `fflush`. Vous devez obtenir le comportement voulu en utilisant le caractère de fin de ligne `\n` seulement.

☞Complétez la fonction `question2` du fichier `q2.c` pour obtenir le comportement voulu. Une fois que vous aurez recompilé le TP, vous pourrez tester votre solution en exécutant `./processlab` dans un terminal.

Astuce: A chaque fois que votre programme effectue un appel système (directement ou via une fonction de librairie), vous avez la possibilité d'imprimer un message d'erreur explicite en cas d'échec de cet appel système. Pour ce faire, il vous suffit d'utiliser la fonction `perror` – consultez sa documentation! – après l'appel système ou l'appel de fonction de librairie en question. Prenez cette habitude pour déboguer plus efficacement votre code pour les prochains TPs!

Instructions pour la partie 2

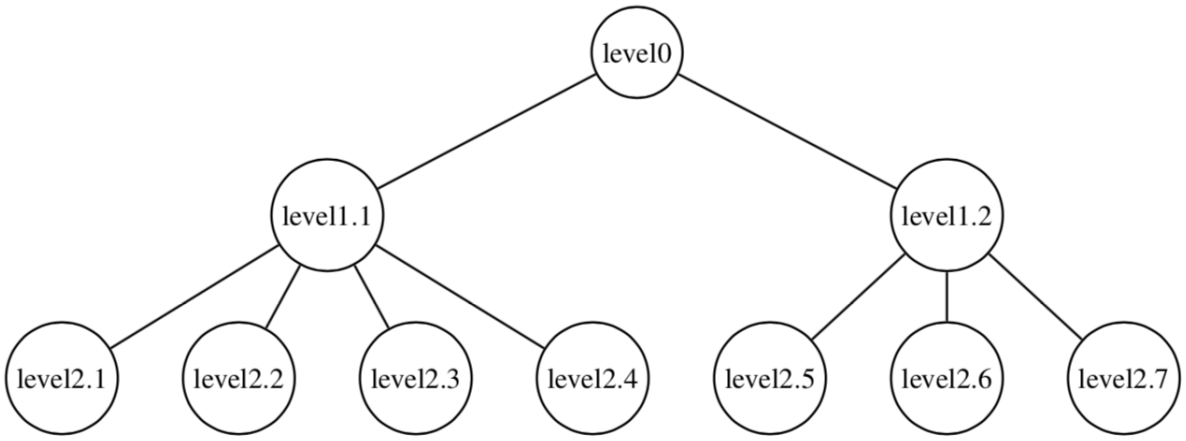

Figure 1: La hiérarchie des processus à recréer pour la partie 2.

Question 2.1 *(Création de l'arbre des processus)* *environ 30mn*

Dans un premier temps, vous allez recréer l'arbre de processus décrit par la figure 1. **Le processus racine *level0* correspond au processus à partir duquel est exécutée la fonction `part2` du fichier `part2.c`.**

☞ Complétez la fonction `part2` du fichier `part2.c` afin de créer les processus selon la hiérarchie définie par la figure 1. A ce niveau, le traitement de chaque processus se limite à créer ses fils (s'il en a) et à attendre leurs terminaisons.

Information: Il n'est pas du tout exigé que votre code comporte des boucles `for`. Faites au plus simple! De même, il n'est pas demandé que votre solution traite les erreurs éventuelles liées aux appels système.

☞ Complétez le code précédent afin que chaque processus (hormis *level0*) fasse appel une fois et une seule fois à la fonction `registerProc` qui vous est fournie dans le fichier `libprocesslab.h`. La fonction `registerProc` a besoin de quatre arguments :

1. le niveau du processus appelant (2 dans le cas de *level2.3* par exemple),
2. le numéro du processus appelant à ce niveau-là (3 dans le cas de *level2.3* par exemple),
3. le PID du processus appelant, et enfin
4. le PID du parent du processus appelant.

Attention: L'ordre des processus décrit par la figure 1 importe! Par exemple, le processus *level1.1* doit être créé avant le processus *level1.2*, car il possède le même processus parent que *level1.2* mais est situé plus à gauche dans la hiérarchie.

Question 2.2 *(Transformation des processus)* *environ 30mn*

Nous vous fournissons des exécutables déjà compilés dans le dossier `part2/`. Vous y trouverez un exécutable par processus décrit par la hiérarchie de la figure 1 : `level0`, `level1.1`, `level1.2`, `level2.1`...

☞Modifiez votre solution pour que chaque processus, **hormis le processus *level0***, se transforme en l'exécutable correspondant (par exemple le processus *level1.2* doit se transformer en l'exécutable `level1.2`). Vous pouvez utiliser n'importe laquelle des fonctions de la famille *exec*, mais soyez attentif à bien respecter la syntaxe de la fonction et la sémantique des arguments. En cas de doute, référez-vous aux *manpages* des fonctions concernées.

☞Vous devez passer comme argument à tous les exécutables `level1.x` le PID du processus *level0*. Pour le reste, laissez-vous guider par les indications données par les exécutables.

☞Modifiez maintenant votre solution pour que le processus *level0* se transforme en l'exécutable `level0`. Ce dernier s'attend également à recevoir un argument, qui est la chaîne de caractères que vous obtiendrez en concaténant les jetons fournis par les processus de niveau 1 dans l'ordre décrit par la figure 1.

Attention: Chaque processus doit se transformer **après** l'appel à la fonction `registerProc` et la création de ses fils, afin de ne pas briser votre solution pour la question 1. D'une manière générale, faites toujours attention à ce que la solution d'une question ne vienne pas briser la solution d'une question précédente!

Question 2.3 *(Comportement de `fprintf` avec `fork`)* *environ 25mn*

Vous avez vu en cours qu'un seul appel à la fonction `printf`, placé juste avant un `fork`, peut imprimer deux messages au lieu d'un seul sur la sortie standard. Vous allez exploiter ce comportement pour imprimer dans plusieurs processus le message suivant:

```
Root process has pid LEVEL0_PID (message from process PROC_NAME)
```

où vous remplacerez `LEVEL0_PID` par le PID du processus *level0* et `PROC_NAME` par le nom du processus qui émet le message (par exemple `level0`). Notez que le message doit être terminé par un caractère de fin de ligne `\n`. Les messages doivent être imprimés **non pas sur la sortie standard, mais dans le fichier** `part2Output.txt`, que nous avons ouvert pour vous dans le fichier `part2.c`. Vous utiliserez ainsi la fonction `fprintf` au lieu de la fonction `printf` que vous avez vue dans le cours. N'hésitez pas à consulter la documentation des fonctions `fprintf` et `fclose`.

Attention: Vous devez faire **exactement un seul appel à la fonction** `fprintf` **qui contienne la chaîne de caractères :** `"Root process has pid LEVEL0_PID "`. Le non-respect de cette consigne invalidera votre note pour cette question.

☞Modifiez votre solution pour que le message précédent soit imprimé dans le fichier fourni par les processus **level0, level1.1, level1.2, level2.1, level2.5 level2.7**, et seulement ces processus.

Question 2.4 *(La question mystère)* *environ 25mn*

Utilisez l'utilitaire `strace` avec l'option `-f` pour analyser votre programme `./processlab`. Testez aussi les options `-fe trace=clone` de `strace` qui permettent de limiter le traçage aux appels système liés à la création des processus. L'appel système `fork` (de la norme `POSIX`) se traduit sous Linux en un appel système `clone`. Ce dernier ne fait pas partie de la norme `POSIX`. Pensez également à visualisez le contenu du fichier `part2Output.txt` produit durant le traçage.
Est-ce que vous ne remarquez pas quelque chose d'incohérent par rapport à l'arbre des processus décrit par la figure 1 ?

☞ En utilisant `strace`, trouvez lequel des programmes de niveau 2 crée un fils supplémentaire non décrit par la figure 1.

☞ Il est possible d'empêcher ce processus de créer un fils en lui communiquant, via une variable d'environement, le jeton suivant: `5b66f7e634e655e2c3848e89`. Utilisez `ltrace` avec les options `-fe getenv` pour déterminer le nom de cette variable d'environnement, qui apparaîtra comme paramètre de la fonction `getenv`.

☞ En utilisant les fonctions `execle` ou `execvpe`, modifiez votre solution afin que le programme de niveau 2 déterminé ci-dessus ne crée pas de fils.

Avertissement: Faites attention à ne pas perturber le comportement demandé par les questions précédentes en modifiant votre solution!

Information: Lorsque vous exécutez un programme sur votre machine, il peut être très utile de savoir:

- quels appels sytème il fait, avec quels paramètres et à quel moment;
- quels appels à des fonctions de librairie il fait, avec quels paramètres et à quel moment;
- combien de processus sont créés et quand.

Ces informations sont utiles pour comprendre le comportement ou la performance d'un programme que vous avez écrit – et de découvrir pourquoi il ne s'exécute pas comme on le voudrait, par exemple! –, voire même pour analyser le comportement d'un programme donc vous n'avez pas le code source. Certains outils, comme `strace` ou `ltrace`, permettent de faire cela. Ils vont vous être utiles tout au long de la session pour vos séances de TP – et peut-être même pour la suite de vos études et ce qui suivra –, alors n'hésitez pas à développer le réflexe de les utiliser quand cela paraît nécessaire.

Instructions

Travailler sur le TP

Toutes vos solutions pour ce TP doivent être écrites dans les fichiers `q1.c` (pour la question 1), `part2.c` (pour les questions 2.1, 2.2, 2.3 et 2.4). **Seuls ces deux fichiers seront pris en compte pour évaluer votre travail; il est donc inutile, voire contre-productif, de modifier les autres fichiers que nous vous fournissons!**

Compiler et exécuter le TP

Nous vous fournissons tous les scripts et librairies pour vous permettre de compiler les sources du TP. Pour compiler le TP initialement et après chacune de vos modifications sur le code source:

Console
```
$ make
```

lorsque vous vous situez dans le répertoire de base du laboratoire. Si la compilation se déroule sans problème, vous pouvez ensuite exécuter le programme:

Console
```
$ ./processlab
```

qui va lancer successivement vos solutions pour chacune des questions du TP.

Evaluer votre travail

Nous vous fournissons les scripts qui vous permettront d'évaluer votre travail autant de fois que vous le souhaitez. Il vous suffit d'exécuter:

Console
```
$ ./grade.sh
```

pour avoir un rapport détaillé sur votre travail.

Information: Notez cependant que cette évalutation s'appuie sur des analyses purement syntaxiques de votre code source et de ses traces d'exécution. Par conséquent, c'est une méthode d'évaluation incomplète qui ne couvre pas tous les cas d'erreur. Les scripts fournis vous donnent une indication mais pas une garantie sur la note finale que vous obtiendrez. Seule l?évaluation par vos chargés de laboratoire sera prise en compte (mais si vous respectez bien les consignes ci-dessus, les deux notes devraient être identiques!).

Rendre votre travail

Votre travail doit être rendu sur le site moodle **avant la fin de cette séance de TP**. Aucune autre forme de remise de votre travail ne sera acceptée. Les retards ou oublis seront sanctionnés comme indiqué sur la page Moodle du cours.

Lorsque vous souhaitez soumettre votre travail – vous pouvez le faire autant de fois que vous le souhaitez pendant la séance –, créez l'archive de remise en effectuant:

Console
```
$ make handin
```

Cela a pour effet de créer le fichier `handin.tar.gz` que vous devrez soumettre sur le site moodle. Seul le dernier fichier remis sera pris en compte pour l'évaluation finale.

Evaluation

Ce TP est noté sur 20 points, répartis comme suit:

- /3.0 pts: Question 1
- /4.5 pts: Question 2.1
- /4.0 pts: Question 2.2
- /3.0 pts: Question 2.3
- /2.0 pts: Question 2.4
- /3.5 pts: Clarté du code

Une première note sur 16.5 points vous est donnée par le script d'auto-évaluation (voir ci-dessus) à titre indicatif. N'hésitez pas à exécuter le script d'auto-évaluation pour connaître le barème détaillé. Les 3.5 points restants seront évalués par la suite par vos chargés de laboratoire, qui vous feront des commentaires via le site moodle.

Ressources

Ce TP vous laisse beaucoup d'autonomie pour programmer votre solution. Le cours constitue une première ressource pour résoudre ce TP. Si vous avez besoin d'informations sur la syntaxe d'une fonction ou d'un programme en particulier, les *manpages* sont une ressource précieuse.

Attention: Copier puis coller du code tout prêt à partir d'Internet (par exemple depuis Stack Overflow) n'est pas considéré comme du travail original et peut être considéré comme du plagiat. Les *manpages* et les documentations officielles, en revanche, vous aident à apprendre à construire un code par vous-même. Privilégiez cette solution pour améliorer votre apprentissage, et n'hésitez pas à solliciter votre chargé de laboratoire si vous avez une question.