INF2610

TP #2: Clone Lab

Ecole Polytechnique de Montréal

Hiver 2019 — Durée: 2 heures et 10 minutes

Présentation

Le *Clone Lab* a pour but de vous familiariser avec la manipulation des processus (partie 1 du TP) et des fils d'exécution (partie 2 du TP). **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 Autolab 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 à la page Moodle du cours.

Objectifs

A l'issue du TP, vous serez capables de:

- Créer des processus selon une hiérarchie donnée;
- Exécuter des programmes à l'intérieur de processus existants;
- Passer des paramètres à un programme;
- Examiner l'exécution d'un programme à l'aide d'outils de traçage;
- Comprendre le comportement des tampons de sortie lors d'un `fork`.

Enoncé

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
1.1	Transformation de processus; lancement de programmes à paramètres	20mn	2.5 pt
2.1	Création de processus; hiérarchie de processus	30mn	4.5 pt
2.2	Transformation de processus	30mn	4.0 pt
2.3	Copie des tampons de sortie pour un processus fils	25mn	3.0 pt
2.4	Utilisation d'outils de traçage; variables d'environnement	25mn	2.0 pt

Instructions pour la partie 1

Question 1.1 *(Les exécutables perdus)* *environ 20mn*

Dans le dossier `part1/` se trouvent deux exécutables `roslyn` et `carlton`, qui se sont "perdus de vue". Chacun prend comme argument le PID d'un autre processus et tente de contacter cet autre processus. Si vous passez à `roslyn` le PID de `carlton`, et inversement, les deux programmes seront capables de se contacter de nouveau!

☞ Complétez la fonction `part1` du fichier `part1.c` pour exécuter les deux programmes du dossier `part1/` en utilisant une des fonctions de la famille `exec`. Vous aurez besoin de créer un processus fils; le père exécutera le premier programme juste après avoir créé un processus fils, et le fils exécutera le second programme. Vous devez passer à chaque exécutable le PID du processus dans lequel roule l'autre exécutable.

Conseils:

- Commencez par exécuter les programmes `roslyn` et `carlton` dans un terminal pour connaître leur syntaxe d'appel.
- Vous pourriez vouloir utiliser la fonction `sprintf` pour construire les chaînes de caractères à passer en argument des exécutables.

Instructions pour la partie 2

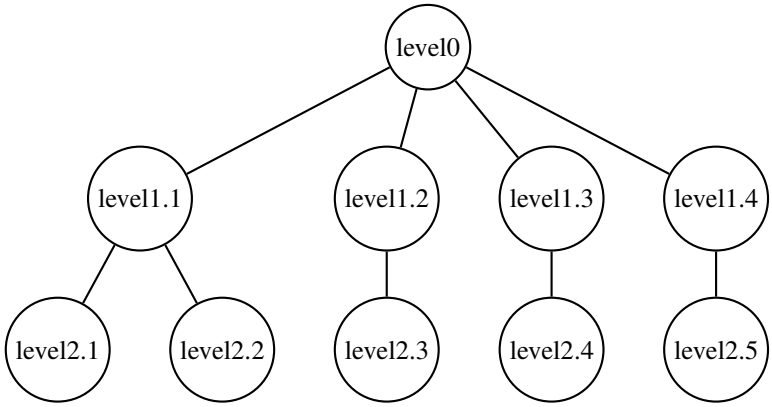

Figure 1: La hiérarchie des processus à recréer pour la partie 2.

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 la création de ses fils (s'il en a).

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 `libclonelab.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 des 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`**. 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, level1.3, level2.3, level2.4, level2.5**, et seulement ces processus.

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

Utilisez l'utilitaire `strace` avec l'option `-f` pour analyser votre programme `./clonelab`. 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: `5f45b266aae3da1be2ada091`. Utilisez `ltrace` 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!

Instructions

Travailler sur le TP

Toutes vos solutions pour ce TP doivent être écrites dans les fichiers `part1.c` (pour la partie 1) et `part2.c` (pour la partie 2). **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
```
$ ./clonelab
```

qui va lancer successivement vos solutions pour les deux parties 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: Les scripts que nous vous fournissons vous donnent une indication mais pas une garantie sur la note finale que vous obtiendrez. Seule l'évaluation par les serveurs d'Autolab sera prise en compte, après vérification par vos chargés de laboratoire (mais si vous respectez bien les consignes ci-dessus, les deux notes devraient être identiques!).

Rendre votre travail

Votre travail doit être rendu sur Autolab **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 remettre sur Autolab. Seul le dernier fichier remis sera pris en compte pour l'évaluation finale.

Evaluation

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

- /2.5 pts: Partie 1
- /13.5 pts: Partie 2
- /4.0 pts: Clarté du code

Une première note sur 16 points vous est donnée par le script d'auto-évaluation (voir ci-dessus) ainsi que par Autolab. N'hésitez pas à exécuter le script d'auto-évaluation pour connaître le barème détaillé. Les 4 points restants seront évalués par la suite par vos chargés de laboratoire, qui vous feront des commentaires via la plateforme Autolab.

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.

Pour ce TP, vous pourriez avoir envie de vous documenter sur les fonctions `fork` et `exec`. Le programme `strace` vous sera certainement très utile également.

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.