INF2610

TP #1: Init Lab

Ecole Polytechnique de Montréal

Hiver 2019 — Durée: 2 heures et 10 minutes

Présentation

Le *Init Lab* a pour but de vous familiariser avec les techniques et outils que vous utiliserez pour tous les TP qui suivront. C'est aussi une introduction ou un rappel à la programmation en C, qui est le langage de programmation qui sera utilisé pour tout le cours. **Il est conseillé de lire l'énoncé en entier avant de commencer le TP.**

Tous les TPs qui suivront seront donnés selon le même format; profitez donc de cette occasion pour vous familiariser avec votre environnement de travail et l'aspect général des TPs.

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:

- Travailler sur les TPs du cours INF2610;
- Rendre un travail sur Autolab;
- Ecrire des programmes simples en C;
- Faire des appels systèmes en C;
- Manipuler des pointeurs en C;
- Comprendre le comportement de programmes avec des outils de traçage.

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:

- `initlab.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`, `q2.c`, `q3.c`, `q4.c` ➔ Ce sont les fichiers de code que vous modifierez au fur et à mesure des TPs;
- `answers-q5.h` ➔ C'est dans ce fichier que vous reporterez les réponses que vous trouverez à la question 5;

- `q2.h`, `q3.h`, `q4.h` ➔ Ces fichiers contiennent la définition des fonctions publiques implémentées dans les fichiers `.c` associés. Vous n'avez pas à vous soucier beaucoup de ces fichiers, mais retenez qu'ils peuvent être utiles pour voir d'un coup d'oeil quelles sont les fonctions que vous pouvez importer d'un fichier à l'autre;
- `q2/`, `q3/`, `q4/`, `q4/` ➔ Ces répertoires contiennent le code précompilé et les programmes qui vous serviront pour chacune des questions;
- `grader/`, `initlab.c`, `libinitlab.o`, `libinitlab.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	Traitement des arguments passés à un exécutable	20mn	2.0 pt
2	Appels systèmes et fonctions de librairie	25mn	3.0 pt
3	Ecriture dans un fichier, obtention de l'identifiant du processus courant	20mn	2.0 pt
4	Pointeurs et allocation dynamique	40mn	5.5 pt
5	Utilisation d'outils de traçage (`strace` et `ltrace`)	25mn	3.5 pt

Question 1 *(Un premier programme C avec `main`)* *environ 20mn*

Nous avons créé pour vous une fonction principale vide dans le fichier `q1.c`. Votre but est de créer un programme qui, quand on l'exécute, affiche le nombre d'arguments qu'il a reçus et la liste de ces arguments, selon le format suivant:

Console

```
$ ./q1 monargument1 2 monargument3
Program received 4 arguments:
./q1
monargument1
2
monargument3
```

Remarquez que, par convention, le premier argument passé à un programme est le nom du programme lui-même, ou plus précisément le contenu de la commande qui a été utilisée pour lancer le programme. **Il est important de s'en souvenir, et cela vous sera notamment utile pour le TP1**.

Attention: Nous imposons une petite subtilité dans le format demandé. Si le programme ne reçoit qu'un seul argument, la première ligne affichée par le programme doit être `Program received 1 argument:`, sans "s" à la fin de "argument"!

Vous utiliserez la fonction `printf`: n'hésitez pas à consulter sa documentation! N'oubliez pas d'imprimer un caractère de fin de ligne `\n` à la fin de chaque ligne.

☞ Complétez la fonction `main` du fichier `q1.c` pour obtenir le comportement voulu.

Conseil: N'oubliez pas de compiler le TP à chaque fois que vous modifiez les fichiers de code pour pouvoir tester votre programme. Dans le cas de la question précédente, vous pouvez tester votre programme en exécutant `./q1 arg1 arg2` dans un terminal. Vous pouvez également vous servir de l'exécutable `initlab`, qui va notamment exécuter et tester votre solution, en exécutant `./initlab` 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 2 *(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 write.2` 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:

```
99c9405f7ec6 (printed using write)
99c9405f7ec6 (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 `./initlab` 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!

Question 3 *(Fichiers et PID)* *environ 20mn*

Ecrire dans des fichiers depuis un programme C vous sera utile tout au long des séances de TP. Le programme `initlab` crée (ou vide s'il existe déjà) pour vous un fichier nommé `q3Output-7f86dc2e4a77.txt`. Dans cette question, vous allez devoir ouvrir ce fichier en écriture, puis y écrire le message suivant:

```
This file has been opened by process ID CURRENT_PID.
```

où vous remplacerez `CURRENT_PID` par le PID du processus qui exécutera votre solution, ce que vous pouvez obtenir grâce à la fonction `getpid`. **N'oubliez pas de terminer la ligne de votre message par un caractère de fin de ligne `\n`**. Vous pourrez utiliser les fonctions `fprintf`, `fopen` et `fclose`.

Pour aller plus loin: Il est également possible de résoudre cet exercice en utilisant respectivement les fonctions `write`, `open` et `close`. Les fonctions `fprintf`, `fopen` et `fclose` ont l'avantage d'être plus simples à utiliser, notamment pour l'ouverture d'un fichier. Si vous voulez aller plus loin et vous préparer aux TP suivants, dans lesquels vous utiliserez la fonction `open`, essayez de résoudre cet exercice sans utiliser `fopen`. Vous remarquerez quelques différences dans votre code: d'une manière générale, c'est toujours bon de savoir arriver à un même résultat en utilisant des techniques différentes!

☞ Complétez la fonction `question3` du fichier `q3.c` pour obtenir le comportement voulu. Une fois que vous aurez recompilé le TP, vous pourrez tester votre solution en exécutant `./initlab` dans un terminal. Vous pouvez vérifier que le message est écrit correctement soit en lisant le contenu du fichier `q3Output-7f86dc2e4a77.txt` après avoir exécuté `initlab`, soit en exécutant le script d'évaluation.

Question 4 *(Pointeurs et fonctions)* *environ 40mn*

Cette question vous permettra de manipuler des pointeurs en C, et de les utiliser pour passer des arguments à une fonction ou pour récupérer le résultat d'une fonction. Vous allez écrire cinq fonctions qui vont toutes prendre un entier x en argument et qui vont calculer le résultat de l'expression:

$$67 * x^2 + 56 * x + 682$$

Voici la signature de chacune des fonctions que vous devrez compléter dans `q4.c`, et les particularités que nous imposons:

- `uint64_t question4A(uint64_t x);`

  Cette fonction est la plus simple des trois. Elle prend la variable x de type `uint64_t` en argument et renvoie le résultat du calcul de l'expression ci-dessus sous forme d'entier.

- `void question4B(uint64_t x, uint64_t *result);`

  Cette fonction prend en argument la variable x, ainsi qu'un pointeur vers un emplacement mémoire (déjà alloué) où stocker le résultat final. Vous devrez modifier la valeur contenue dans cet emplacement mémoire pour y mettre le résultat du calcul de l'expression ci-dessus. La fonction elle-même ne renvoie rien.

- `uint64_t *question4C(uint64_t x);`

  Cette fonction prend en argument la variable x et renvoie un pointeur vers un emplacement mémoire que votre fonction aura alloué – consultez la documentation de la fonction `malloc`! – et qui contiendra le résultat du calcul de l'expression ci-dessus. Vous n'avez pas à vous soucier de libérer l'emplacement mémoire que votre fonction aura créé, c'est le code du TP qui s'en chargera.

- `void question4D(uint64_t x, uint64_t **resultPtr);`

  Cette fonction prend en argument la variable x, ainsi qu'un deuxième argument qui est pointeur vers un emplacement mémoire pouvant contenir un autre pointeur. Vous devez modifier le pointeur contenu dans cet emplacement mémoire par un pointeur vers un emplacement mémoire que votre fonction aura alloué – vous utiliserez encore `malloc`! – et qui contiendra le résultat du calcul de l'expression ci-dessus. Vous n'avez pas à vous soucier de libérer l'emplacement mémoire que votre fonction aura créé, c'est le code du TP qui s'en chargera.

- `uint64_t question4E(uint64_t x);`

  Cette fonction a la même signature et le même comportement que la fonction `questionA`. Nous ajoutons cependant une contrainte: vous devez impérativement obtenir le résultat du calcul de l'expression ci-dessus par un appel à la fonction `_question4B` (**et non `question4B`!**) définie dans le fichier `q4/libq4.h`, que nous avons programmée et qui a la même signature que votre fonction `question4B`. Cela vous permettra d'apprendre à passer correctement un argument de type pointeur à une fonction.

☞ Complétez le code du fichier `q4.c` pour obtenir le comportement voulu. Une fois que vous aurez recompilé le TP, vous pourrez tester votre solution en exécutant `./initlab` dans un terminal.

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.

Question 5 *(Suivons nos programmes à la trace)* *environ 25mn*

Le but de cette question est de vous présenter des situations dans lesquelles l'utilisation des outils mentionnés précédemment peut se révéler précieuse. Nous avons mis dans le dossier `q5/` un programme `filewriter` dont on aimerait comprendre mieux le comportement. Avant de vous lancer dans la résolution des exercices, prenez le temps de consulter la documentation des outils mentionnés précédemment – par exemple en exécutant `man strace` dans un terminal pour obtenir la documentation de `strace`. **Pour chacune des questions suivantes, vous devez inscrire la réponse dans le fichier `answers-q5.h`**, dans la variable dont le nom est indiqué entre crochets à la fin de la question.

Attention: Pour les questions suivantes, on vous demande d'utiliser `ltrace` et `strace` pour tracer un processus fils, ou encore pour récupérer le contenu de longues chaînes de caractères. **Vous aurez besoin des options `-f` et `-s` de ces outils: consultez la documentation!**

☞ **a)** Le programme `filewriter` tente d'ouvrir un fichier en lecture seule. Quel est le chemin du fichier? Vous pouvez utiliser `ltrace` ou `strace` pour trouver la réponse à cette question. [reportez la réponse dans la variable `Q5_ANS_A`]

☞ **b)** Le programme `filewriter` crée un processus fils qui ouvre un fichier en écriture. Quel est le chemin du fichier ouvert en écriture? [reportez la réponse dans la variable `Q5_ANS_B`]

☞ **c)** Quelle est la chaîne de caractères écrite dans le fichier ouvert en écriture? [reportez la réponse dans la variable `Q5_ANS_C`]

☞ **d)** Quel est le nom de la fonction de librairie utilisée pour écrire dans le fichier? [reportez la réponse dans la variable `Q5_ANS_D`]

☞ **e)** Le processus fils du programme `filewriter` a la capacité de se mettre en pause pendant son exécution. Pour cela, il faut lui préciser la durée d'attente voulue (en secondes) via une variable d'environnement. Trouvez le nom de la variable d'environnement en question en utilisant l'outil `ltrace`. [reportez la réponse dans la variable `Q5_ANS_E`]

Instructions

Travailler sur le TP

Toutes vos solutions pour ce TP doivent être écrites dans les fichiers `q1.c` (pour la question 1), `q2.c` (pour la question 2), `q3.c` (pour la question 3), `q4.c` (pour la question 4) et `answers-q5.h` (pour la question 5). **Seuls ces cinq 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

```
$ ./initlab
```

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: 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 (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:

- /2.0 pts: Question 1
- /3.0 pts: Question 2
- /2.0 pts: Question 3
- /5.5 pts: Question 4
- /3.5 pts: Question 5
- /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.

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.