Travail pratique #1

IFT-2245

January 30, 2019

Dû le 8 février à 23h59

1 Survol

Ce TP vise à vous familiariser avec la programmation système dans un système d'exploitation de style POSIX. Les objectifs de ce travail sont les suivantes:

1. Parfaire sa connaissance de programmation en C.
2. Parfaire sa connaissance des "forks" et processus.

Les étapes sont les suivantes:

1. Lire et comprendre cette donnée.
2. Lire, trouver, et comprendre les parties importantes du code fourni.
3. Compléter le code fourni.
4. Écrire un rapport. Il doit décrire **votre** expérience pendant les points précédents: problèmes rencontrés, surprises, choix que vous avez dû faire, options que vous avez sciemment rejetées, etc... Le rapport ne doit pas excéder 5 pages.

Ce travail est à faire en groupes de 2 étudiants. Le rapport, au format LaTeXexclusivement et le code sont à remettre par remise électronique avant la date indiquée. Chaque jour de retard est -25%. Indiquez clairement votre nom au début de chaque fichier.

Si un étudiant préfère travailler seul, libre à lui, mais l'évaluation de son travail n'en tiendra pas compte. Si un étudiant ne trouve pas de partenaire, il doit me contacter au plus vite. Des groupes de 3 ou plus sont **exclus**.

2 CH: un shell

Vous allez devoir implanter une ligne de commande, similaire à `/bin/sh`:

1. **Le Shell** - être capable d'exécuter des commandes

   ```
   $ ./ch

   votre shell> echo bonjour

   Output: bonjour
   ```

   Autre commandes que nous allons tester: (`cat, ls, man, tail`). NB: Tu ne doit pas faire les mises en oeuvre des fonctions mais juste les executer.

   vous devez supposer que la commande a la structure:

   ```
   [COMMAND_NAME] [arg1] [arg2] ... [-option_1] [-option_2] ...
   ```

   où le nombre d'arguments et / ou le nombre d'options peuvent être 0. Exemple:

   ```
   votre shell> ls -h (pas d'argument mais 1 option)
   votre shell> ls .. (1 argument mais pas d'option)
   ```

2. **Comprendre l'utilité du status de terminaison d'un processus avec** && **et** ||:

   ```
   votre shell> cat nofile && echo Le fichier existe.

   Output:

   cat: nofile: No such file or directory

   votre shell> cat nofile || echo Le fichier n'existe pas

   Output:

   cat: nofile: No such file or directory
   Le fichier n'existe pas
   ```

3. **une déclaration "IF"** - votre shell doit être capable de faire:

```
votre shell> if true ; do echo bonjour ; done

Output:

bonjour
```

De plus - il faut que votre "IF" fonctionne correctement avec le && et ||:

```
votre shell> if cat nofile && true ; do echo Le fichier existe ; done

Output:

cat: nofile: No such file or directory

votre shell> if cat nofile || true ; do echo Le fichier n'existe pas ; done

Output:

cat: nofile: No such file or directory
Le fichier n'existe pas
```

4. **Redirection de sortie et gestion de tache arriere-plan avec** & **et** >

Mettre un & à la fin de n'importe quelle instruction devrait renvoyer immédiatement l'utilisateur au shell et lui permettre d'entrer une autre commande. En d'autres termes, cette tâche devrait être mise en arrière-plan. Mais la sortie est toujours affichée. Exemple:

```
votre shell> sleep 10 &
[commence sleep 10, rien imprimé à l'écran]
votre shell> echo blah
blah
votre shell> ...
[après 10 seconde ... fini sleep 10]
```

Mettre un > à la fin de n'importe quelle instruction redirige la sortie vers un fichier. Exemple:

```
votre shell> echo blah > blah.txt
```

Aucun sortie et "blah" est ajouté à la fin de blah.txt (blah.txt est créé s'il n'existe pas)

Ces deux choses devraient également fonctionner ensemble. Exemple:

```
votre shell> grep test / -r > output.txt &
```

À noter: cela ne devrait pas avoir besoin de fonctionner pour les exécutables qui lisent depuis l'entrée standard. Exemple:

```
votre shell>cat &
```

Bonus 1 (1 pt): Bonus 1: Étendre l'opérateur & pour qu'il gère de la bonne façon les processus qui utilisent l'entré standard (STDIN).

Bonus 2 (1 pt): Met un processus en arrière-plan pendant son exécution (et renvoie le shell à l'utilisateur) si l'utilisateur appuie sur le caractère "CTRL-z"

À noter: Les démonstrateur ne répondront pas aux questions sur les bonus

Le programme doit être exécutable sur `arcade.iro` ou sur une nouvelle image de docker d'ubuntu.

Cela ne vous empêche pas bien sûr de le développer sur un système différent, e.g. sous Windows avec Cygwin, mais assurez-vous que le résultat fonctionne *aussi* sur `arcade.iro`.

3 Cadeaux

Vous recevez en cadeau de bienvenue les fichiers `Makefile`, `rapport.tex`, et `ch.c` qui contiennent le squelette (vide) du code et du rapport que vous devez rendre.

3.1 Remise

Pour la remise, vous devez remettre deux fichiers (`ch.c` et `rapport.tex`) par la page StudiUM du cours. Assurez-vous que tout fonctionne correctement sur `arcade.iro`.

4 Détails

- La note sera divisée comme suit: 20% pour chaque tache (1-4), et 20% pour le rapport.
- Tout usage de matériel (code ou texte) emprunté à quelqu'un d'autre (trouvé sur le web, ...) doit être dûment mentionné, sans quoi cela sera considéré comme du plagiat.
- Le code ne doit en aucun cas dépasser 80 colonnes. indications supplémentaires.
- La note sera basée d'une part sur des tests automatiques, d'autre part sur la lecture du code, ainsi que sur le rapport. Le critère le plus important, et que votre code doit se comporter de manière correcte. Ensuite, vient

la qualité du code: plus c'est simple, mieux c'est. S'il y a beaucoup de commentaires, c'est généralement un symptôme que le code n'est pas clair; mais bien sûr, sans commentaires le code (même simple) est souvent incompréhensible. L'efficacité de votre code est sans importance, sauf s'il utilise un algorithme vraiment particulièrement ridiculement inefficace.

Appendix

Examples avec output:

```
# Output: AB
if true; do echo -n A; true; done && echo B

# Output: AB
if true; do echo -n A; true; done && echo B &

# Output: A
if true; do echo -n A; true; done || echo B

# Output: A
if true; do echo -n A; true; done || echo B &

# Output: A
if true; do echo -n A; false; done && echo B

# Output: A
if true; do echo -n A; false; done && echo B &

# Output: AB
if true; do echo -n A; false; done || echo B

# Output: AB
if true; do echo -n A; false; done || echo B &

# Output: B
if false; do echo -n A; true; done && echo B

# Output: B
if false; do echo -n A; true; done && echo B &

# Output:
if false; do echo -n A; true; done || echo B

# Output:
```

```
if false; do echo -n A; true; done || echo B &

# Output: B
if false; do echo -n A; false; done && echo B

# Output: B
if false; do echo -n A; false; done && echo B &

# Output:
if false; do echo -n A; false; done || echo B

# Output:
if false; do echo -n A; false; done || echo B &
```