TP 1 Shell et couteau suisse

{cecile1.goncalves,abderraouf.kabouche,pascal.vanier}@u-pec.fr

1 Shell élémentaire

Le langage C est un langage de programmation très permissif, qui permet d'écrire facilement des programmes illisibles et incompréhensibles tant au niveau syntaxique qu'au niveau sémantique. Cependant, il est moins aisé d'écrire des programmes qui soient très illisibles, imbriquant plusieurs niveaux de difficulté. Ainsi, le programme C suivant met en oeuvre un mini-shell incluant la plupart des fonctionnalités que nous allons développer dans cette fiche de TP (à l'exception de la gestion des variables d'environnement et des scripts) :

```
#define D ,close(
char*c,q[512],m[256],*v[99],**u,*i[3];int f[2],p;main(){for(m[m[60]=m[62]=
32]=m[*m=124[m]=9]=6;e(-8),gets(1+(c=q))||exit(0);r(0,0))for(;*++c;);}
r(t,o){*i=i[2]=0;for(u=v+98;m[*--c]^9;m[*c]&32?i[*c&2]=
*u,u-v^98&&++u:3)if(!m[*c]){for(*++c=0;!m[*--c];);*--u=
++c;}u-v^98?strcmp(*u,"cd")?*c?pipe(f),o=f[1]:1,(p=fork())?e(p),o? r(o,0)D
o)D*f):4,wait(0):(o?dup2(*f,0)D*f)D o):*i?1 D 0),e(open(*i,0)):5,t?
dup2(t,1)D t):i[2]?9 D 1),e(creat(i[2],438)):2,e(execvp(*u,u))):
e(chdir(u[1])*2):6;} e(x){x<0?write(2,"?\n$ "-x/4,2),x+1||exit(1):5;}
```

— B. Rakitzis et S. Dorward, IOCCC 1990

Voici quelques conseils, en vrac, afin de faciliter votre travail :

- lisez attentivement les consignes et respectez les ;
- utilisez un éditeur de texte raisonnable (par exemple, **vim**, **emacs**...) ;
- indentez votre code source ;
- commentez vos en-têtes de fonctions ;
- la documentation est dans le manuel en ligne : *man(1)* (c'est-à-dire **man 1 man**) ;
- utilisez les options de détection d'erreurs du compilateur C ;
- utilisez un Makefile pour compiler avec **make**, en voici un minimal :

```
CC=gcc
CFLAGS=-W -Wall -std=gnu99 -g
```

- lisez les messages d'erreur et les avertissements du compilateur ;
- utilisez les outils de débuggage comme **gdb** ;
- testez les valeurs de retour des fonctions.

Exercice 1 (Initialisation de votre compte GIT)**.** Si vous n'êtes pas redoublant :

- Vous avez reçu un lien d'activation pour GIT dans votre boite mail UPEC, cliquez dessus.
- Réinitialisez votre mot de passe pour http://git-etudiants.lacl.fr en cliquant sur `Forgot your password ?`

Si vous êtes redoublant, vous avez fait cela l'année dernière, si non, retrouvez le mail d'activation dans votre boite mail et cliquez dessus.

Exercice 2 (Création du TP sur git)**.** Suivez les instructions sur http://git-etudiants.lacl.fr afin de créer le répertoire dans lequel vous effectuerez les TP. Des slides sont également disponibles sur eprel pour vous aider à comprendre GIT. Vous devrez versionner tous les fichier **source** de vos TP, mais pas les exécutables. **A la fin de chaque TP vous devrez faire un push de toutes les modifications.**

Exercice 3 (Analyse syntaxique)**.** Écrire une fonction `int parse_line(char *s);` qui, à partir de la chaîne de caractères `s`, terminée par `'\0'`, analyse cette chaîne et appelle la fonction `int simple_cmd(char *argv[]);` chargée de l'interpréter : la dernière case du tableau `argv` doit être mise à `NULL` afin de connaître la fin du tableau. La fonction `simple_cmd` se contente d'afficher ce qu'on lui passe en argument et retourne `0`. Dans le cas où le nom de la commande passée en paramètre est `exit`, le programme est arrêté. Une ligne de commandes est une suite de mots-clefs `commande argument1 ... argumentn` qu'il faut passer à `simple_cmd`. Elle peut contenir des commentaires, le caractère `#` indiquant que tout ce qui suit est un commentaire et doit donc être ignoré. La fonction `main` affiche une invite de commande `$ ` et attend qu'une ligne soit tapée avant de l'envoyer à `parse_line`. **Attention :** n'oubliez pas de versionner vos modifications

Références : *fgets(3)*, *strpbrk(3)*, *strcmp(3)*.

Exercice 4 (Exécution de commandes simples)**.** Modifier la fonction `simple_cmd` afin d'exécuter la commande passée en paramètre, d'attendre sa terminaison, puis de retourner dans la boucle principale du shell. Dans le cas où la commande invoquée est `cd`, changer de répertoire sans faire d'appel à `fork()`. Pourquoi faut-il exécuter cette commande à l'intérieur du shell ? **Attention :** n'oubliez pas de versionner vos modifications

Références : *fork(2)*, *execvp(3)*, *wait(2)*, *chdir(2)*.

Exercice 5 (Variables d'environnement)**.** Modifier la fonction `parse_line` afin d'ajouter un nouveau type de ligne de commandes : les lignes de la forme `chaine=valeur` qui servent à mettre la valeur `valeur` dans la variable d'environnement `chaine`. Faites en sorte que les occurrences de mots de la forme `$chaine` soient remplacées par la valeur de la variable d'environnement `chaine`. **Attention :** n'oubliez pas de versionner vos modifications

Références : *getenv(3)*, *setenv(3)*, *putenv(3)*.

Exercice 6 (Fichiers scripts)**.** Modifier la fonction `main` afin d'ajouter l'exécution de scripts de commandes. Lorsque le programme est invoqué avec un nom de fichier en argument, ce fichier est ouvert et les commandes qu'il contient sont exécutées, le programme s'arrête à la fin du fichier ou lorsqu'il rencontre la commande `exit`. Dans le cas d'un script, le shell n'affiche pas d'invite de commande. **Attention :** n'oubliez pas de versionner vos modifications

Références : *fopen(3)*, *fclose(3)*.

Exercice 7 (Redirections d'entrées-sorties)**.** Modifier la fonction `parse_line` afin d'ajouter la gestion des redirections vers des fichiers en entrée `<` et en sortie `>`. Pour exécuter la commande, on utilisera la fonction `int redir_cmd(char *argv[], char *in, char *out);` (à la place de `simple_cmd`) qui reçoit en paramètre les noms des fichiers à ouvrir en entrée et en sortie (`NULL` si on utilise le fichier standard). **Attention :** n'oubliez pas de versionner vos modifications

Références : *open(2)*, *dup2(2)*, *close(2)*.

Exercice 8 (Gestion des filtres)**.** Modifier la fonction `parse_line` afin d'ajouter la gestion des filtres. Une ligne de commande est alors une suite de commandes simples séparées par des pipes `|`. La première commande peut comporter une redirection en entrée et la dernière une redirection en sortie. Pour exécuter l'ensemble des commandes on utilisera trois fonctions (et éventuellement des variables globales) :

- `start_cmd(char *argv[], char *in );` pour la première commande ;
- `next_cmd(char *argv[]);` pour les commandes intermédiaires ;
- `last_cmd(char *argv[], char *out);` pour la dernière commande.

Attention : n'oubliez pas de versionner vos modifications

Références : *dup2(2)*, *pipe(2)*.

Exercice 9 (Gestion des signaux)**.** Modifier votre programme afin d'ajouter la gestion des signaux : dans un premier temps, vous devez rattraper, si vous le pouvez, les signaux pouvant faire terminer votre shell pour que celui-ci ne termine pas, mais que les programmes exécutés gèrent les signaux normalement. **Attention :** n'oubliez pas de versionner vos modifications

Références : *sigaction(2)*.

Exercice 10 (Gestion des signaux)**.** Dans un second temps, **pour ceux qui sont en avance**, vous implémenterez les commandes `fg` et `bg` qui permettent de réveiller un processus endormi grâce à CTRL+Z et de l'exécuter soit en arrière plan, soit en avant plan.

Lorsque l'on appuie sur CTRL+Z, le terminal envoie un signal SIGSTP, par défaut ce signal endort le processus en cours. Il faut que votre shell garde en mémoire les processus qui ont été endormis et permette de les réveiller avec soit `fg n` soit `bg n` où `n` est le numéro du processus endormi.

On peut réveiller un processus en lui envoyant le signal `SIGCONT`. **Attention :** n'oubliez pas de versionner vos modifications

Références : *sigaction(2)*, *kill(2)*, *wait(2)*.

Exercice 11 (Nettoyage du code)**.** Il faut maintenant préparer votre code pour le rendu :

- Nettoyez votre code, et documentez le. Il faut un commentaire au début de chaque fonction expliquant ce qu'elle fait, sa valeur de retour ainsi que ce qu'elle prend en argument.
- Rendez votre projet facile à compiler en ajoutant un *Makefile* au dépôt. L'exécutable obtenu devra s'appeler `myshell`, celui-ci ainsi que le Makefile devront être dans la racine de votre dépôt.
- Votre dépôt git ne doit versionner aucun exécutable.

Si vous ne respectez pas ces consignes le code ne sera pas noté, entraînant une note de 0.
Attention : n'oubliez pas de versionner vos modifications

Références : *make(1)*, *git(1)*.