TP3 : Shell

Gilles Menez - UNSA - UFR Sciences - Dépt. Informatique

28 septembre 2018

Objectifs pédagogiques

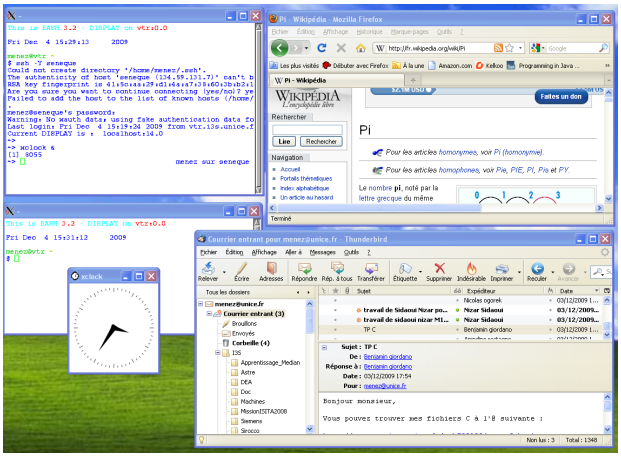
Il s'agit de vous familiariser avec :

- ➢ La notion de processus
- ➢ La notion de shell
- ➢ Et j'espère quelques points vus en cours !

Clairement, ca pulse un peu :-) On s'accroche (= on bosse et on demande).

1 C'est quoi un Shell "basique" ?

Le Shell est le programme qui s'exécute dans les terminaux alphanumériques permettant de formuler des commandes.



Un `Shell` est un nom générique désignant (dans le monde Unix) un **interpréteur de commandes** :

- ➢ C'est un logiciel et donc un processus durant son exécution.
- ➢ Son rôle est de provoquer/organiser l'exécution de la commande spécifiée par une ligne de texte.

  Donc il permet de lancer des processus !

Plusieurs logiciels existent et permettent de remplir cette tâche :

- ➢ sh, ksh, zsh, csh, tcsh, bash, ...

Ils ont des spécificités mais partagent beaucoup de concepts !

Un shell moderne offre de nombreuses fonctionnalités **rendant l'approche graphique lente et imprécise** en comparaison :

① Définition de variables : "`export TOTO=1234`"

② Mécanisme d'**édition de la ligne de commande** :

➢ `[Ctrl] a` ,`[Ctrl] e` , `[Ctrl] b` ,`[Ctrl] f` , `[Ctrl] d` , ...

③ Gestion d'un **historique des commandes** :

➢ On recule dans le temps par `[Ctrl] p`

➢ et on avance dans le temps par `[Ctrl] n` .

④ Mécanisme de **complétion** :

➢ `[Tab]`

⑤ Programmation de scripts : un **langage** de commandes !

Les commandes auxquelles répond un shell sont :

① les **commandes internes** au shell :
Certaines commandes à la disposition de l'utilisateur, sont programmées **dans** le shell et celui-ci peut donc les exécuter directement.

➢ Elles sont peu nombreuses.

➢ On trouve par exemple les commandes `cd` ou `pwd` .

② les **commandes externes** au shell
Pour les exécuter, le shell lance un programme c'est à dire **transforme un fichier exécutable situé dans l'arborescence des fichiers en un processus en mémoire**.

Pour faciliter le travail de l'utilisateur, les commandes qui sont situées dans la liste des répertoires enregistrée dans la variable ***« PATH »*** (attention, le nom est en majuscules) peuvent être appelées par leur nom terminal.

➢ On peut par exemple taper `man` au lieu de `/bin/man` parce que le répertoire `/bin` fait partie de la liste de répertoires de la variable PATH :

echo $PATH
PATH = . :/bin/ :/usr/bin :/sbin/

Voila, c'était un petit rappel !

1.1 Activités d'un Shell

Entre son lancement et sa terminaison, un Shell réalise trois choses principales :

① Initialisation : le processus "shell" lit et utilise ses fichiers de configuration. On peut ainsi configurer la chaîne du prompt, les raccourcis clavier possibles ...

② Interprétation : le shell lit les commandes de son stdin et les exécute.

③ Terminaison : le shell termine "proprement" les commandes qu'il a permis de lancer, libère la mémoire, ...

Cela donne un corps de programme :

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char * argv[]){

  // Init : Load config files, if any

  // Interp : Run Command loop
  sh_loop();

  // Termin : Perform any shutdown / cleanup

  return EXIT_SUCCESS;
}
```

1.2 La boucle de base

La boucle de base d'un Shell consiste donc à lire la commande puis à exécuter cette commande :

```
#include <stdio.h>
#include <stdlib.h>

void sh_loop(void){
  char *prompt = ">␣";
  char *line;
  char **args;
  int status;

  do {
    printf("%s",prompt);
    fflush(stdout);

    line = sh_read_line();       /* Lecture de la ligne de commande */
    args = sh_split_line(line); /* dont on extrait les args */

    status = sh_execute(args);

    sh_free(line);
    sh_free(args);
  } while(status);
}

int main(int argc, char * argv[]){

  // Init : Load config files, if any

  // Interp : Run Command loop
  sh_loop();

  // Termin : Perform any shutdown / cleanup

  return EXIT_SUCCESS;
}
```

1.3 Lire et analyser

La lecture s'appuiera sur la fonction de bibliothèque :

ssize_t getline(char **lineptr, size_t *n, FILE *stream) ;

On vous laisse faire le lien entre le manuel et son utilisation :

```c
#include <stdio.h>
#include <stdlib.h>

char *sh_read_line(void){
  char *line = NULL;
  ssize_t bufsize = 0; // donc getline realise l'allocation
  getline(&line, &bufsize, stdin);
  return line;
}

void sh_loop(void){
  char *prompt = ">␣";
  char *line;
  char **args;
  int status;

  do {
    printf("%s",prompt);
    fflush(stdout);

    line = sh_read_line();
    args = sh_split_line(line);

    status = sh_execute(args);

    sh_free(line);
    sh_free(args);
  } while(status);
}

int main(int argc, char * argv[]){

  // Init : Load config files, if any

  // Interp : Run Command loop
  sh_loop();

  // Termin : Perform any shutdown / cleanup

  return EXIT_SUCCESS;
}
```

L'analyse de la commande nécessite de procéder à sa "tokenisation". Dit autrement il s'agit d'identifier les différents éléments de la ligne :

- Le nom de la commande,
- Ses options,

➢ Ses arguments.

On pourra ensuite lui associer un sens (i.e. sémantique).

On prend la même convention qu'Unix où la syntaxe générale des commandes est :

```
commande options ... arguments ...
```

Il y un (au moins) espace entre les éléments (c'est à dire les tokens) qui composent une commande.

➢ Les **options** indiquent des variantes dans l'exécution de la commande.

✔ Les options sont le plus souvent précédées d'un tiret « - ».

✔ L'ordre des options est le plus souvent, mais pas toujours, indifférent.

✔ Et plusieurs options peuvent être regroupées derrière un seul tiret.

➢ Les **arguments** indiquent les objets sur lesquels la commande va agir

✔ Les arguments peuvent être absents, ils prennent alors des valeurs par défaut.

1.4 Exemples de commandes

Voila des exemples de commandes que votre Shell pourrait avoir à exécuter :

```
>date
>ls -la
>ls -l /users/students
>ls /users/students
>find . -name "*.c" -print
>tar zcf toto.tgz Mycours
```

1.5 Tokenisation et fonction strtok()

Là encore, il y a la fonction qu'il faut en bibliothèque pour nous permettre de décomposer ces lignes de commandes :

```
/*
   Ce programme illustre l'utilisation de la fonction
   strtok pour décomposer une chaine (qui pourrait
   etre une commande) et obtenir une liste des différents
   tokens qui la compose (séparés par un separateur).
*/
#include<string.h>
#include<stdio.h>
#include<stdlib.h>

#define MNB 50    /* Nombre max de tokens dans une ligne */

/*=====================================*/

int analyse_cmd(char *l[]){
  int i = 0;
  /* Affichage des tokens */
  while(1){
    if(l[i]==NULL)
      break;
    printf("Item #%d is %s.\n",i,l[i]);
```

```
    i++;
  }
  return 0;
}

/*=====================================*/

char **find_tokens_list(char input_string[], char sep[]){
  int i;
  char **tl; /* Liste de tokens */
  tl = calloc(sizeof(char *),MNB);

  /* Y a t'il un debut d'une sequence de tokens ? */
  tl[0]=strtok(input_string, sep);
  if(tl[0]==NULL){
    printf("string is empty or contains only delimiters ?\n");
    exit(0);
   }

  /* Stockage des tokens cf NULL param*/
  for(i=1;i<MNB;i++){
    tl[i]=strtok(NULL, sep); /* get next */

    if(tl[i]==NULL) /* Fin de liste */
      break; /* rien de plus a extraire */
   }

  return tl;
}

/*=====================================*/

int main(int argc, char *argv[]){
  char s[]="pierre || jean paul || anna";
  //char s[]="Woody";

  char sep[] = "||"; /* Separateur de tokens */
  char **l;
  l = find_tokens_list(s,sep);
  analyse_cmd(l);
}
```

L'exemple qui est fourni

- ➢ utilise un séparateur : "||"
- ➢ qui permet de d'obtenir la liste des tokens.

L'analyse de la commande est réalisée par la fonction dédiée . . . qui se contente d'effectuer l'affichage des tokens.

- ➢ Prenez note de la gestion des espaces qui persistent ici puisque ce ne sont pas eux les séparateurs.

Je trouve le descriptif suivant de `strtok` plutôt pas mal : http://icarus.cs.weber.edu/~dab/cs1410/textbook/8.Strings/strtok.html

1.6 Processus fils

Le shell en cours de réalisation doit, une fois la commande identifiée (ainsi que ses paramètres et arguments), lancer cette commande.

➢ Il doit donc "forker" puis changer le code de ce processus fils.

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

/*=========================================================*/

char *sh_read_line(void){
  char *line = NULL;
  ssize_t bufsize = 0; // donc getline realise l'allocation
  getline(&line, &bufsize, stdin);
  return line;
}

/*=========================================================*/

#define LSH_TOK_BUFSIZE 64
#define LSH_TOK_DELIM " \t\n"
char ** sh_split_line( char *line){
  int bufsize = LSH_TOK_BUFSIZE, position = 0;
  char **tokens = malloc(bufsize * sizeof(char*));
  char *token;

  if (!tokens) {
    fprintf(stderr, "lsh: allocation error\n");
    exit(EXIT_FAILURE);
  }

  token = strtok(line, LSH_TOK_DELIM);
  while (token != NULL) {
    tokens[position] = token;
    position++;

    if (position >= bufsize) {
      bufsize += LSH_TOK_BUFSIZE;
      tokens = realloc(tokens, bufsize * sizeof(char*));
      if (!tokens) {
        fprintf(stderr, "lsh: allocation error\n");
        exit(EXIT_FAILURE);
      }
    }

    token = strtok(NULL, LSH_TOK_DELIM);
  }
  tokens[position] = NULL;
  return tokens;
}

/*=========================================================*/

int sh_execute(char **args){
  /* To complete !! */
```

```
}

/*=========================================================*/

void sh_loop(void){
  char *prompt = "l3miage shell > ";
  char *line;
  char **args;
  int status;

  do {
    printf("%s",prompt);
    fflush(stdout);

    line = sh_read_line();
    args = sh_split_line(line);
    status = sh_execute(args);

    /* sh_free(line); */
    /* sh_free(args); */
  } while(status);
}


int main(int argc, char * argv[]){

  // Init : Load config files, if any

  // Interp : Run Command loop
  sh_loop();

  // Termin : Perform any shutdown / cleanup

  return EXIT_SUCCESS;
}
```

Votre contribution :

Ecrire la fonction `int sh_execute(char **args)` qui permettra de lancer vos commandes.

2 Evolutions

On propose quelques évolutions pour ce shell : **TODO** !

① On introduit un mécanisme de type "contrôle parental".
Toutes les commandes contenant les chaînes : "mp3", "mp4", "youtube", "avi" ne seront pas exécutées.
Un message sur la console en donnera la raison :

"Travaille au lieu de jouer !"

Au début, ces chaînes "interdites" seront définies dans le code.

② Mais il pourrait être intéressant qu'elles soient contenues dans une variable d'environnement :

FORBIDDEN

Cette variables auraient été définies dans le Shell qui a permis de lancer votre Shell :

export FORBIDDEN="mp3 :mp4 :youtube :avi"

③ Votre shell pourrait proposer des commandes internes.

La première, c'est `exit` qui permet de quitter votre Shell.

Puis deux autres commandes : `newf` et `rmf` .
Ces commandes permettent respectivement d'ajouter et d'enlever une chaîne dans FORBIDDEN.

④ Puisque vous venez de faire la commande `exit` , il serait bon **d'interdire** que l'on puisse quitter votre Shell avec un Ctrl-c.

⑤ Dans la version actuelle, le lancement d'une commande bloque peut être la console qui attend la fin du fils et il n'y a pas de mécanisme d'arrière plan.

Comment en introduire un ?

3 Démo

```
menez@vtr ~/EnseignementsCurrent/Cours_Sys_Prog/TheTps/TP3_Unix/Src
$ gcc shell_full.c -o shell_miage

menez@vtr ~/EnseignementsCurrent/Cours_Sys_Prog/TheTps/TP3_Unix/Src
$ echo $FORBIDDEN


menez@vtr ~/EnseignementsCurrent/Cours_Sys_Prog/TheTps/TP3_Unix/Src
$ export FORBIDDEN="mp3:mp4:avi"

menez@vtr ~/EnseignementsCurrent/Cours_Sys_Prog/TheTps/TP3_Unix/Src
$ echo $FORBIDDEN
mp3:mp4:avi

menez@vtr ~/EnseignementsCurrent/Cours_Sys_Prog/TheTps/TP3_Unix/Src
$ ./shell_miage
Get FORBIDDEN variable from env : mp3:mp4:avi
le shell de la l3miage > ls
Commande conforme !
a.out               shell_cp_executetocomplete.c  shell_cp_shloop.c  shell_miage
film.mp4            shell_cp_main.c               shell_cp_split.c   strtoka.c
shell_cp_execute.c  shell_cp_readline.c           shell_full.c       strtok.c
le shell de la l3miage > ls film.mp4
Va bosser au lieu d'essayer de glander!
le shell de la l3miage > ^C
Utilisez exit pour sortir !
on veut savoir si vous avez essayer de jouer !

le shell de la l3miage > exit
Commande conforme !
Vous avez essaye 1 fois d'executer une commande interdite !
Je m'en souviendrais !

menez@vtr ~/EnseignementsCurrent/Cours_Sys_Prog/TheTps/TP3_Unix/Src
$
```