Architecture et Système Projet 2018-2019

Département Informatique, ENS Paris Saclay

Devoir maison

Introduction: Ce DM se décompose en deux parties :

— Une partie programmation qui va vous demander de recoder un *shell* en C

— Une partie contenant des questions théoriques/pratiques liées au cours

Les modalités de rendus seront précisées à la fin du sujet

1 Recoder un Shell

Télecharger le bootstrap à l'adresse suivante : `www.lsv.fr/~fthire/teaching/2018-2019/archisys/projet/shell-bootstrap.tar.gz`. Vous trouverez le squelette de base du code C que l'on va utiliser pour recoder un shell. Pour compiler le projet, je vous invite à utiliser la commande `make`. Par défault, le shell ne peut pas faire grand chose. On va essayer de le compléter pas à pas. On va d'abord s'intéresser à la fonction `execute` :

1.1 Question 1

Le cas de base, correspond au cas `C_PLAIN`. Donner un exemple de commande qui une fois parsée retourne un objet `cmd` tel que `cmd->type == C_PLAIN`. En utilisant `ps`, observer ce qui se passe dans un terminal lorsque vous lancez une commande.

Pour le moment, toute commande de base sera tout simplement exécutée. Pour exécuter une commande, la librairie `glibc` offre un panel de fonctions dont on peut avoir un aperçu en utilisant la commande

```
man exec
```

Selon vous, quelle fonction serait la plus appropriée dans notre cas (justifier) ? En utilisant toutes ces observations, remplir le trou `C_PLAIN`.

1.2 Question 2

Quel est le symbol pour l'opérateur de séquence en bash ? Donner un exemple de commande où la séquence se comporte différent vis-à-vis de l'opérateur *et* logique.

Implémenter le cas `C_SEQ`.

1.3 Question 3

Implémenter les cas `C_AND` et `C_OR`.

1.4 Question 4

Il est possible en bash d'écrire une commande de la forme

(cmd1 && cmd2 | cmd3 ...) 2>/dev/null

Quel est le rôle des parenthèses dans la commande ci-dessus ? Donner un exemple d'une commande qui utilise (de façon non triviale) ces parenthèses.

Implémenter le cas `C_VOID`.

1.5 Question 5

Que se passe-t-il si vous faites `CTRL+C` dans notre shell ? Proposer une solution pour récupérer la main après que l'utilisateur a entré `CTRL+C`.

1.6 Question 6

Que se passe-t-il lorsque que vous lancez la commande

```
ls > dump
```

Pour corriger ce problème, je vous invite à lire les pages de manuel `man stdin` et `man dup`. En utilisant toutes ces informations, implémenter la fonction `apply_redirections` puis modifier votre implémentation pour que la commande ci-dessus se comporte comme prévu.

1.7 Question 7

Il nous reste finalement à implémenter le cas `C_PIPE`, pour cela je vous invite à regarder le manuel de `man pipe`. Donner un exemple qui met en évidence pourquoi on ne peut pas simplement utiliser `dup2` pour réimplémenter le pipe ? En utilisant la fonction `pipe`, réimplémenter le cas `C_PIPE`.

1.8 Question 8

À ce stade, nous avons implémenté un *shell* très rudimentaire, cependant il est possible de l'étendre de bien des manières. Voici quelques possibilités d'extensions qui peuvent vous rapporter des points bonus :

- Réimplémenter les commandes `ls`, `cat` ou `cd`
- Implémenter l'extension des wildcards : `ls *.pdf`
- Implémenter les processus de fond via les commandes : `jobs`, `bg`, `fg`, ...

2 Questions diverses

2.1 Question 9

Le programme ci-dessous est inefficace, pourquoi ?

```c
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char **argv)
{
  int fd1, fd2, rc;
  char buf;
  if(argc != 3) {
    fprintf(stderr, "Syntaxe: %s f1 f2\n", argv[0]);
    exit(1);
  }

  fd1 = open(argv[1], O_RDONLY);
  if(fd1 < 0) {
    perror("open(fd1)");
    exit(1);
```

```
  }

  fd2 = open(argv[2], O_WRONLY | O_CREAT | O_TRUNC, 0666);
  if(fd2 < 0) {
    perror("open(fd2)");
    exit(1);
  }
  while(1) {
    rc = read(fd1, &buf, 1);
    if(rc < 0) {
      perror("read");
      exit(1);
    }
    if(rc == 0)
    break;
    rc = write(fd2, &buf, 1);
    if(rc < 0) {
      perror("write");
      exit(1);
    }
    if(rc != 1) {
      fprintf(stderr, "Ecriture interrompue");
      exit(1);
    }
  }
  close(fd1);
  close(fd2);
  return 0;
}
```

Expliquez pourquoi. En particulier, décrivez les endroits critiques du programme. Quelle(s) correction(s) pouvez-vous proposer pour corriger le programme ci-dessus ?

2.2 Question 10

Le principe du programme ci-dessous est le suivant : on génère des nombres aléatoires entre 0 et 512 et on fait la somme de ceux qui sont plus grand que 256 une certain nombre de fois (10000 fois présentement). On remarque cependant que si on trie le tableau, ce calcul est plus rapide en moyenne (en tout cas sur ma machine). Donner une explication à ce phénomène ainsi que les différents mécanismes qui rentrent en jeu.

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#define SIZE 100000

int comp (const void * elem1, const void * elem2)
{
    int f = *((int*)elem1);
    int s = *((int*)elem2);
    if (f > s) return  1;
    if (f < s) return -1;
    return 0;
```

```
}

int main(int argc, char* argv[]) {
  int data[SIZE];
  int data2[SIZE];

  srand(time(NULL));
  for (unsigned int c = 0 ; c < SIZE ; c++) {
    data[c]=rand()%512;
    data2[c]=rand()%512;
  }

  qsort (data2, sizeof(data2)/sizeof(*data2), sizeof(*data2), comp);

  int sum = 0;
  clock_t start_1, end_1;
  start_1=clock();
  for(unsigned int i = 0 ; i < 10000 ; i++) {
    for (unsigned int c = 0 ; c < SIZE ; c++) {
      if(data[c] < 256) {
        sum += data[c];
      }
    }
  }
  end_1=clock();

  sum=0;
  clock_t start_2, end_2;
  start_2=clock();
  for(unsigned int i = 0 ; i < 10000 ; i++) {
    for (unsigned int c = 0 ; c < SIZE ; c++) {
      if(data2[c] < 256) {
        sum += data2[c];
      }
    }
  }
  end_2=clock();

  printf("not sorted: %f\n", (double) (end_1-start_1)/CLOCKS_PER_SEC);
  printf("sorted: %f\n", (double) (end_2-start_2)/CLOCKS_PER_SEC);

  return 0;
}
```

2.3 Question 11

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

int main(int argc, char* argv[]) {
  float step1 = 1/2.;
  float step2 = 1/10.;
  float sum1, sum2 = 0;
```

```
  printf("step1: %f\n", step1);
  printf("step2: %f\n", step2);
  for(int i = 0 ; i < 50 ; i++) {
    sum1 += step1;
    sum2 += step2;
  }
  printf("sum1: %f\n", sum1);
  printf("sum2: %f\n", sum2);

  return 0;
}
```

Le programme ci-dessus produit l'affichage suivant.

```
step1: 0.500000
step2: 0.100000
sum1: 25.000000
sum2: 4.999998
```

— Expliquer l'affichage.
— Que se passe-t-il si on remplace les `float` par des `double` ?

2.4 Question 12

Expliquer le comportement du programme suivant :

```
#include <stdlib.h>
#include <stdio.h>
#include <float.h>
int main(int argc, char* argv[]) {
  double y = FLT_MAX;
  if (y == y+1.) {
    printf("true\n");
  }
  else {
    printf("false\n");
  }

  return 0;
}
```

2.5 Question 13

Sur le programme suivant :

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

#define SIZE 10000

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

  int **array = (int **) malloc(SIZE * sizeof(int *));
```

```
  for (unsigned int i=0; i < SIZE; i++)
    array[i] = (int *) malloc(SIZE * sizeof(int));
  int sum;
  srand(time(NULL));

  for (unsigned int x = 0 ; x < SIZE ; x++) {
    for (unsigned int y = 0 ; y < SIZE ; y++) {
      array[x][y]=rand()%2;
    }
  }

  sum = 0;
  clock_t temps1;
  for (unsigned int x = 0 ; x < SIZE ; x++) {
    for (unsigned int y = 0 ; y < SIZE ; y++) {
      sum+=array[x][y];
    }
  }

  temps1=clock();

  sum = 0;
  clock_t temps2;
  for (unsigned int y = 0 ; y < SIZE ; y++) {
    for (unsigned int x = 0 ; x < SIZE ; x++) {
      sum+=array[x][y];
    }
  }

  temps2=clock();
  printf("temps1: %f\n", (double) temps1/CLOCKS_PER_SEC);
  printf("temps2: %f\n", (double) temps2/CLOCKS_PER_SEC);
}
```

— Expliquer pourquoi on utilise un malloc au lieu de `int array[SIZE][SIZE]` ?
— Expliquer pourquoi on observe un temps d'exécution plus long la deuxième fois ?

2.6 Question 14

Le programme suivant n'a pas le comportement attendu :

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

int main (int argc, char *argv[]) {
  int p[2];
  pipe(p);
  close(p[0]);

  if (fork()) {
    while (1) {
      sleep(1);
      write(p[1],"a",1);
```

```
    }
  } else {
    char c = 'a';
    while (1) {
      read(p[0],&c,1);
      write(1,&c,1);
    }
  }
  return 0;
}
```

— Expliquer pourquoi le programme se comporte ainsi.
— Que se passe-t-il si vous le lancez en background (avec `&`) ? Pourquoi ?

2.7 Question 15

1. Écrire un programme qui affiche :
   — L'adresse courante de la pile
   — L'adresse courante du tas
   — L'adresse de la fonction `main`
2. En déduire la taille maximale qu'un tableau peut allouer sur la pile
3. À votre avis, pourquoi l'adresse de la fonction `main` n'est pas égale à `0x0` ?
4. Écrire un programme qui ne fait rien et retourne simplement la valeur `42`. Compiler, et observer la taille de l'exécutable. Selon vous, est-ce cohérent avec l'assembleur généré [1] ? Qu'est-ce qui peut expliquer ce phénomène ?

Pour ces deux dernières questions, on attend seulement une réponse de quelques lignes.

3 Modalités de rendu

Votre rendu devra se présenter sous une archive de la forme `NOM_PRENOM.tar.gz` (crée via la commande `tar czf`) qui se dézippera en un dossier `NOM_PRENOM`. Ce dossier contiendra un fichier `readme.pdf` (au format pdf donc) ainsi qu'un dossier `shell` qui contiendra votre code. **Attention : tout programme qui ne compile pas ne sera pas évalué**. Votre fichier `readme.pdf` devra contenir les informations suivantes :

— Les réponses à certaines questions concernant l'implémentation du `shell`
— Des exemples de commandes qui mettent en évidence le fonctionnement de votre implémentation
— Une description des extensions optionnelles implémentées
— Les réponses aux questions de la seconde partie

La qualité des réponses sera prise en compte dans la notation.

Quelques conseils :

— Il vous est conseillé d'écrire le `readme.pdf` avec `LaTeX`. Pour inclure du code, vous pouvez utiliser le package `minted` (utilisé pour ce document mais que vous devrez installer à la main avec le paquet python `pygmentize`) ou bien le paquet `lstlistings`.
— Dans vos réponses, demandez-vous si chaque phrase est nécessaire. Si une phrase n'est pas nécessaire, alors c'est qu'elle est inutile et vous pouvez l'enlever. Éviter aussi des réponses trop vagues ou bien trop génériques, soyez précis !

1. utiliser l'option `-S` pour générer un fichier assembleur