UNIVERSIDAD DEL BÍO-BÍO
FACULTAD DE CIENCIAS EMPRESARIALES

Laboratorio 2: MiniShell

Nicolás Gómez Morgado

Laboratorio Sistemas Operativos

24 de septiembre de 2024

Ejercicios:

Parte A

1. & (5pts)

```
nicogomez@SODebian:~/Escritorio/Sistemas Operativos/Lab3$ chmod 744 Archivo.sh
nicogomez@SODebian:~/Escritorio/Sistemas Operativos/Lab3$ ./Archivo.sh
Sleeping for 10 seconds...
Creating the file test123
./Archivo.sh: línea 5: 1: orden no encontrada
Deleting the file test123
nicogomez@SODebian:~/Escritorio/Sistemas Operativos/Lab3$
```

Se creo un archivo llamado *Archivo.sh* en el cual se introdujo la secuencia de comandos asignada, como resultado se observa una correcta compilación a excepción de un error sintáctico en la línea 5, el cual se refiere a que hay un "1" de mas.

2. &&, (Sequence ; ;;) || (5pts)

```
nicogomez@SODebian:~/Escritorio/Sistemas Operativos/Lab3$ touch test && touch test123
nicogomez@SODebian:~/Escritorio/Sistemas Operativos/Lab3$ ls -l
total 4
-rwxr--r-- 1 nicogomez nicogomez 248 sep 24 09:48 Archivo.sh
-rw-r--r-- 1 nicogomez nicogomez   0 sep 24 09:58 test
-rw-r--r-- 1 nicogomez nicogomez   0 sep 24 09:58 test123
nicogomez@SODebian:~/Escritorio/Sistemas Operativos/Lab3$ (ls; ps)
Archivo.sh  test  test123
    PID TTY          TIME CMD
   2863 pts/0    00:00:00 bash
   4773 pts/0    00:00:00 ps
nicogomez@SODebian:~/Escritorio/Sistemas Operativos/Lab3$ ls; ps; whoami
Archivo.sh  test  test123
    PID TTY          TIME CMD
   2863 pts/0    00:00:00 bash
   4777 pts/0    00:00:00 ps
nicogomez
nicogomez@SODebian:~/Escritorio/Sistemas Operativos/Lab3$ lzl || echo "the lzl command failed"
bash: lzl: orden no encontrada
"the lzl command failed"
nicogomez@SODebian:~/Escritorio/Sistemas Operativos/Lab3$
```

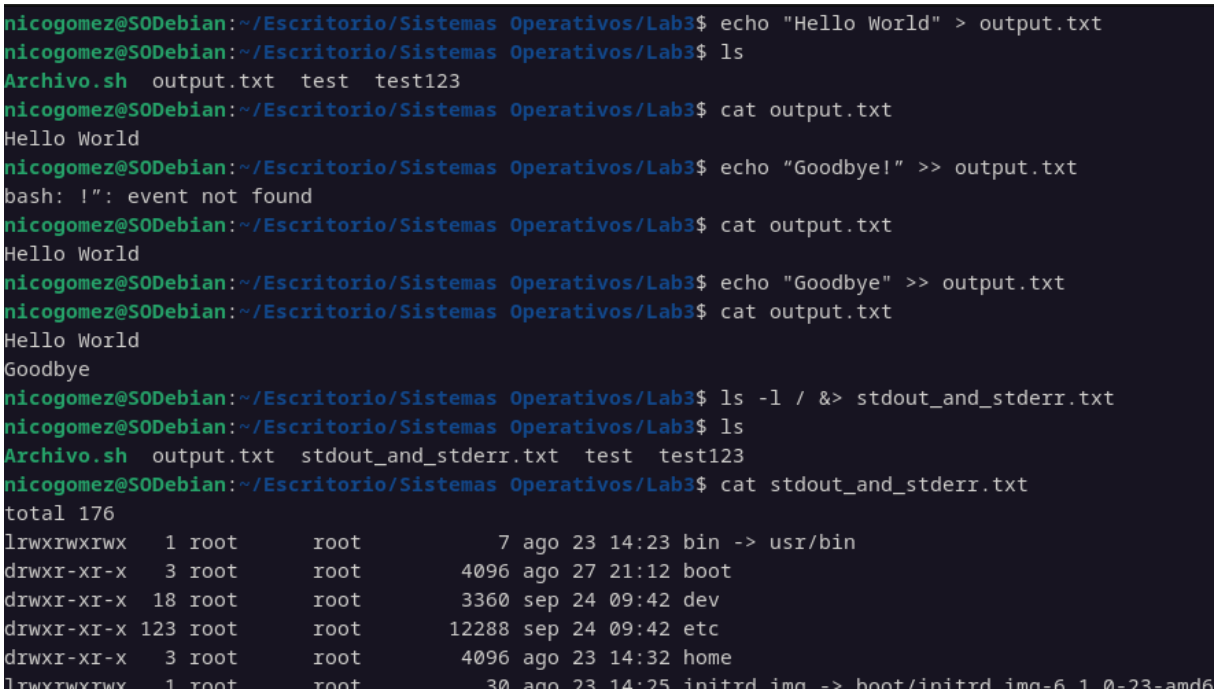
Como resultado de la ejecución del conjunto de comandos `touch test && touch test123` se observa que se crean los archivos *test* y *test123*, por otro lado, al ejecutar el conjunto de comandos `(ls; ps)` se observa que se listan los los procesos en ejecución, para el caso de `ls; ps; whoami` se observa que se listan los los procesos en ejecución y el usuario actual y finalmente al ejecutar `lzl || echo "the lzl command failed"` se observa un error del bash ya que el comando `lzl` no existe, ademas de la impresión del texto *"the lzl command failed"*.

3. Pipe (5pts)

```
nicogomez@SODebian:~/Escritorio/Sistemas Operativos/Lab3$ cat name.txt
Alice
Lamda
John
Mike
Bob
nicogomez@SODebian:~/Escritorio/Sistemas Operativos/Lab3$ cat name.txt | sort
Alice
Bob
John
Lamda
Mike
nicogomez@SODebian:~/Escritorio/Sistemas Operativos/Lab3$
```

Para este caso se creo un archivo a parte con de nombre *name.txt* que contiene una lista de 5 nombres, por consiguiente se ejecuta el comando `cat name.txt | sort` el cual imprime los nombres ordenados alfabéticamente.

4. Redirection (5pts)



Para este ejercicio se hizo lo siguiente:

- Se ejecuto `echo "Hello World» output.txt`: Este comando creo un archivo llamado *output.txt* con el contenido *Hello World*.
- Se ejecuto `echo "Goodbye»> output.txt`: Este comando agrego el contenido *Goodbye* al final del archivo *output.txt*.
- Se ejecuto `cat output.txt`: Este comando imprime el contenido del archivo *output.txt* para ver los cambios realizados.
- Se ejecutó `ls -l / &> stdout_and_stderr.txt`: Este comando imprime el contenido de la lista de archivos del directorio raíz y lo guarda en el archivo *stdout_ and_ stderr.txt*.

Parte B

```
nicogomez@SODebian:~/Escritorio/Sistemas Operativos/Lab3$ nano minishell.c
nicogomez@SODebian:~/Escritorio/Sistemas Operativos/Lab3$ cat minishell.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

#define MAX_LINE 80 /* The maximum length command */

void parseCommand(char *input, char **args) {
    int i = 0;
    args[i] = strtok(input, " \n");
    while (args[i] != NULL) {
        i++;
        args[i] = strtok(NULL, " \n");
    }
}

int main(void) {
    char *args[MAX_LINE / 2 + 1]; /* command line arguments */
    char input[MAX_LINE];
    int should_run = 1; /* flag to determine when to exit program */
```

Se creo el archivo *minishell.c* en el cual se introdujo la secuencia de comandos asignada.

```
nicogomez@SODebian:~/Escritorio/Sistemas Operativos/Lab3$ nano minishell.c
nicogomez@SODebian:~/Escritorio/Sistemas Operativos/Lab3$ ls
Archivo.sh  minishell.c  output.txt  stdout_and_stderr.txt  test  test123
nicogomez@SODebian:~/Escritorio/Sistemas Operativos/Lab3$ g++ -o minishell minishell.c
nicogomez@SODebian:~/Escritorio/Sistemas Operativos/Lab3$ ./minishell
minishell>
```

Se ejecuto el archivo *minishell.c*.

1. Ejecuta comandos básicos como 'ls', 'pwd', 'date'. Luego prueba la redirección de entrada y salida 'ls > output.txt', 'sort < file.txt > sorted.txt' y ejecuta comandos en segundo plano 'sleep 10 &'. Explica lo sucedido. (15pts)

```
minishell> ls
Archivo.sh  minishell  minishell.c  output.txt  stdout_and_stderr.txt  test  test123
minishell> pwd
/home/nicogomez/Escritorio/Sistemas Operativos/Lab3
minishell> date
mar 24 sep 2024 10:33:38 -03
minishell> ls > output.txt
ls: no se puede acceder a '>': No existe el fichero o el directorio
 output.txt
minishell> sort < file.txt > sorted.txt
sort: no se puede leer: '<': No existe el fichero o el directorio
minishell> sleep 10 &
sleep: intervalo de tiempo inválido «&»
Pruebe 'sleep --help' para más información.
minishell> sleep 10
minishell>
```

Se ejecutó el comando `ls` el cual me listó los archivos y carpetas existentes, se ejecutó el comando `pwd` el cual me muestra la ruta actual, se ejecutó el comando `date` el cual me muestra la fecha y hora actual, se ejecutó el comando `ls > output.txt` el cual me lanzó el error **ls: no se puede acceder a '>': No existe el fichero o el directorio** siendo que el archivo *output.txt* sí existe por lo que el error se puede deber a que el comando general del minishell no está diseñado para esta funcionalidad, se ejecutó el comando `sort < file.txt > sorted.txt` el cual también me lanzó el error **sort: no se puede leer '<': No existe el fichero o el directorio** sin embargo para este caso nunca existió el archivo *file.txt* por lo que el error se debe a que el archivo no existe, por último se ejecutó el comando `sleep 10 &` el cual me lanzó el error **bash: error sintáctico cerca del elemento inesperado '&'** el cual se debe a que el comando no está bien estructurado pero al ejecutar el comando `sleep 10 &` sí se ejecuta correctamente.

2. Investiga y añade soporte para el comando 'cd'. Copia tu código completo aquí. (40pts)

```
nicogomez@SODebian:~/Escritorio/Sistemas Operativos/Lab3$ ./minishell
minishell> ls
Archivo.sh        minishell    output.txt            test
directorioPrueba  minishell.c  stdout_and_stderr.txt  test123
minishell> cd directorioPrueba
minishell> pwd
/home/nicogomez/Escritorio/Sistemas Operativos/Lab3/directorioPrueba
minishell> cd ..
/home/nicogomez/Escritorio/Sistemas Operativos/Lab3
minishell> ls
Archivo.sh        minishell    output.txt            test
directorioPrueba  minishell.c  stdout_and_stderr.txt  test123
minishell>
```

Se añadió soporte para el comando `cd` en el archivo *minishell.c*.

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <pwd.h> // Para obtener el directorio home
#include <errno.h>

#define MAX_LINE 80  /* The maximum length command */
#define MAX_PATHS 100 /* Maximum number of stored paths */
#define MAX_PATH_LEN 1024 /* Maximum path length */

// Estructura para manejar las rutas previas
typedef struct {
    char paths[MAX_PATHS][MAX_PATH_LEN];
    int size;
} PathStack;

void push_path(PathStack *stack, const char *path) {
    if (stack->size < MAX_PATHS) {
        strncpy(stack->paths[stack->size], path, MAX_PATH_LEN - 1);
        stack->paths[stack->size][MAX_PATH_LEN - 1] = '\0'; // Null-terminated
        stack->size++;
    }
}

void pop_path(PathStack *stack, char *prev_path) {
    if (stack->size > 0) {
        strncpy(prev_path, stack->paths[stack->size - 1], MAX_PATH_LEN);
        stack->size--;
    }
}

void parseCommand(char *input, char **args) {
    int i = 0;
    args[i] = strtok(input, " \n");
    while (args[i] != NULL) {
        i++;
        args[i] = strtok(NULL, " \n");
    }
}

int main(void) {
    char args[MAX_LINE / 2 + 1]; / command line arguments */
    char input[MAX_LINE];
    int should_run = 1; /* flag to determine when to exit program */
    PathStack path_stack;
    char current_dir[MAX_PATH_LEN];

    path_stack.size = 0;

    // Obtener el directorio inicial
    if (getcwd(current_dir, sizeof(current_dir)) == NULL) {
        perror("getcwd");
        exit(1);
    }
```

```
58
59  while (should_run) {
60      printf("minishell> ");
61      fflush(stdout);
62
63      if (fgets(input, MAX_LINE, stdin) == NULL) {
64          perror("fgets failed");
65          exit(1);
66      }
67
68      if (strncmp(input, "exit", 4) == 0) {
69          should_run = 0;
70          continue;
71      }
72
73      parseCommand(input, args);
74
75      // Comando cd
76      if (strcmp(args[0], "cd") == 0) {
77          if (args[1] == NULL || strcmp(args[1], "~") == 0) {
78              // Cambiar al directorio home si no hay argumentos
79              const char *home_dir = getenv("HOME");
80              if (home_dir == NULL) {
81                  home_dir = getpwuid(getuid())->pw_dir;
82              }
83              push_path(&path_stack, current_dir);
84              if (chdir(home_dir) == -1) {
85                  perror("cd");
86              } else {
87                  getcwd(current_dir, sizeof(current_dir));
88              }
89          } else if (strcmp(args[1], "..") == 0) {
90              // Volver al directorio anterior
91              char previous_dir[MAX_PATH_LEN];
92              if (path_stack.size > 0) {
93                  pop_path(&path_stack, previous_dir);
94                  printf("%s\n", previous_dir);
95                  if (chdir(previous_dir) == -1) {
96                      perror("cd");
97                  } else {
98                      getcwd(current_dir, sizeof(current_dir));
99                  }
100             } else {
101                 printf("No previous directory stored.\n");
102             }
103         } else {
```

```
104     // Cambiar al directorio especificado
105                 push_path(&path_stack, current_dir);
106                 if (chdir(args[1]) == -1) {
107                     perror("cd");
108                 } else {
109                     getcwd(current_dir, sizeof(current_dir));
110                 }
111             }
112         } else {
113             pid_t pid = fork();
114             if (pid < 0) {
115                 perror("Fork failed");
116                 exit(1);
117             } else if (pid == 0) {
118                 if (execvp(args[0], args) < 0) {
119                     perror("execvp failed");
120                 }
121                 exit(1);
122             } else {
123                 wait(NULL);
124             }
125         }
126     }
127     return 0;
128 }
```