Ivan Chowdhury
Professor Hakner
ECE 357: Computer Operating Systems
Problem Set 3 - Simple Shell
10/29/2018

Interactive Operation [Running shell, Comments, Error Handling]

```
Ivan@DESKTOP-BBPO23V /cygdrive/c/users/acobl/google drive/source code/GitHub [Public]/school/os/3 - shell
$ ./myshell
# Ignore this
# Hello world
Hello world
Error: Failed to execute command Hello: No such file or directory
Command returned exit status: 0
consuming 0.037000 real seconds, 0.015000 user, 0.000000 system.
```

Built-in commands

```
pwd
/cygdrive/c/users/acobl/google drive/source code/GitHub [Public]/school/os/3 - shell
cd .
pwd
/cygdrive/c/users/acobl/google drive/source code/GitHub [Public]/school/os/3 - shell
cd ..
pwd
/cygdrive/c/users/acobl/google drive/source code/GitHub [Public]/school/os
cd
pwd
/home/Ivan
exit (56)

Ivan@DESKTOP-BBPO23V /cygdrive/c/users/acobl/google drive/source code/GitHub [Public]/school/os/3 - shell
```

Script Interpreter

```
Ivan@DESKTOP-BBPO23V /cygdrive/c/users/acobl/google drive/source code/GitHub [Public]/school/os/3 - shell
$ ./myshell testscript.sh
srfyhgsyhswy
sty
st
ywst
y
wyy25
757y65w2
gdb
nb
Command returned exit status: 0
consuming 18.362000 real seconds, 0.000000 user, 0.030000 system.
srfyhgsyhswy
sty
st
ywst
y
wyy25
757y65w2
gdb
nb
Command returned exit status: 0
consuming 0.053000 real seconds, 0.000000 user, 0.031000 system.
```

```
Ivan@DESKTOP-BBPO23V /cygdrive/c/users/acobl/google drive/source code/GitHub [Public]/school/os/3 - shell
$ ./myshell ./testscript.sh <input.txt
Command returned exit status: 0
consuming 0.055000 real seconds, 0.031000 user, 0.015000 system.
This is a test input file for testing I/O redirection.
Line 2
;
--gshhtweuj testCommand returned exit status: 0
consuming 0.061000 real seconds, 0.000000 user, 0.015000 system.

Ivan@DESKTOP-BBPO23V /cygdrive/c/users/acobl/google drive/source code/GitHub [Public]/school/os/3 - shell
$ echo $
$
```

I/O Redirection [Input, Output, Error]

```
Ivan@DESKTOP-BBPO23V /cygdrive/c/users/acobl/google drive/source code/GitHub [Public]/school/os/3 - shell
$ ./myshell testscript2.sh <input.txt
Command returned exit status: 0
consuming 0.063000 real seconds, 0.015000 user, 0.015000 system.
```

```
ls -l >c.out
Command returned exit status: 0
consuming 0.070000 real seconds, 0.000000 user, 0.062000 system.
ls >c2.out
Command returned exit status: 0
consuming 0.069000 real seconds, 0.000000 user, 0.030000 system.
cat c.out c2.out
total 208
-rw-r--r--+ 1 Ivan None      0 Oct 29 23:15 c.out
-rw-r--r--+ 1 Ivan None     49 Oct 29 23:13 cat.out
-rw-r--r--+ 1 Ivan None     20 Oct 29 23:11 cat2.out
-rwx------+ 1 Ivan None     83 Oct 27 13:12 input.txt
-rwx------+ 1 Ivan None    140 Oct 29 21:38 Makefile
-rwx------+ 2 Ivan None  10176 Oct 29 22:37 myShell.c
-rwxr-xr-x+ 1 Ivan None 165301 Oct 29 22:55 myshell.exe
-rwx------+ 1 Ivan None  11782 Oct 22 21:47 pset-w03.pdf
drwx------+ 1 Ivan None      0 Oct 29 23:14 Screenshots
-rwx------+ 1 Ivan None    894 Oct 29 23:10 testscript.sh
-rwx------+ 1 Ivan None    535 Oct 29 23:10 testscript2.sh
c.out
c2.out
cat.out
cat2.out
input.txt
Makefile
myShell.c
myshell.exe
pset-w03.pdf
Screenshots
testscript.sh
testscript2.sh
Command returned exit status: 0
consuming 0.050000 real seconds, 0.000000 user, 0.015000 system.
```

```
Ivan@DESKTOP-BBP023V /cygdrive/c/users/acobl/google drive/source code/GitHub [Public]/school/os/3 - shell
$ ./myshell 2>>Error_report.txt
pwd
/cygdrive/c/users/acobl/google drive/source code/GitHub [Public]/school/os/3 - shell
cd .
cd ..
pwd
/cygdrive/c/users/acobl/google drive/source code/GitHub [Public]/school/os
cd
pwd
/home/Ivan
ls
'Assignment 2.rar'   c.out
ls -l
total 160965
-rwxr-xr-x 1 Ivan None 164826038 May  3 11:23 'Assignment 2.rar'
-rw-r--r-- 1 Ivan None       128 Oct 29 22:36  c.out
exit 123

Ivan@DESKTOP-BBP023V /cygdrive/c/users/acobl/google drive/source code/GitHub [Public]/school/os/3 - shell
$ cat Error_report.txt
Command returned exit status: 0
consuming 0.054000 real seconds, 0.000000 user, 0.030000 system.
Command returned exit status: 0
consuming 0.056000 real seconds, 0.000000 user, 0.046000 system.
```

```
#!/absolute/path/to/your/shell
#This is an example of a shell script that your shell must execute correctly
#notice that lines starting with a # sign are ignored as comments!
#let’s say this here file is called testscript.sh.  you created it with say
#vi testscript.sh ; chmod +x testscript.sh
#you invoked it with
#./testscript.sh
cat >cat.out
#at this point, type some lines at the keyboard, then create an EOF (Ctrl-D)
#your shell invoked the system cat command with output redirected to cat.out
cat cat.out
#you better see the lines that you just typed!
exit 123
#after your shell script exits, type echo $? from the UNIX system shell
#the value should be 123.  Since your shell just exited, the following
#bogus command should never be seen
###############################################################################
```

```
#!/absolute/path/to/your/shell
#here is another example, say it is called testscript2.sh
#you invoked it with
#./testscript2.sh <input.txt
cat >cat2.out
#since you invoked the shell script (via the system shell such as bash)
#with stdin redirected, your shell runs cat which gets stdin from input.txt
exit
#the above exit had no specified return value, so your shell exited with 0
#because the last child spawned, cat, would have returned 0
#again, test this with echo $?
```

```c
// Ivan Chowdhury
// Cooper Union ECE357: Computer Operating Systems
// Professor Hakner
// Fall 2018
// Program 3 - Simple Shell

#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/times.h>
#include <sys/resource.h>
#include <sys/wait.h>

#define BUFSIZE 4096    // Default buffer size

int myShell(FILE *input);   // Shell function
int builtpwd();     // Built-in commands: pwd, cd, and exit
int builtcd(char *CDpath);
void builtexit(char *exit_status);

int main(int argc, char **argv) {
    if (argc > 2) {     // Program takes in 1 argument max
        fprintf(stderr, "Error: Only one argument is accepted: %s\n",
strerror(errno));
        return -1;
    }
    else if (argc == 2) {   // If input file argument is given, open for reading
        FILE *input;
        if ((input = fopen(argv[1], "r")) == NULL) {
            fprintf(stderr, "Error: Failed to open input file %s: %s\n", argv[1],
strerror(errno));
            return -1;
        }
        myShell(input);

        if (argc == 2 && fclose(input) != 0) {    // Close input file
            fprintf(stderr, "Error: Failed to close input file %s: %s\n", argv[1],
strerror(errno));
            return -1;
        }
    }
    else
        myShell(stdin);  // If no arguments given, read from standard input

    return 0;
}

int myShell(FILE *input) {

    char *line;   // Buffer for line string
    char *token, **tokenArgV;    // Contains characters and argument vector for
tokens
    char *delim = " \r\n";     // Delimiter for extracting tokens from string

    char *new_path;     // New path for I/O redirection
    int fdnew;      // New file descriptor for I/O redirection
```

```
57
58      int flags, i;  // Flags for open syscall
59      size_t n;   // getline parameters: buffer size and characters read
60      ssize_t bytesRead;
61
62      fdnew = -1;
63      i = 0;
64      n = 4096;
65
66
67      if (!(new_path = malloc(BUFSIZE * (sizeof(char))))) {   // Dynamically allocate
memory for new file path
68          fprintf(stderr, "Error: Failed to allocate memory for redirected file path:
%s\n", strerror(errno));
69      }
70
71      if (!(line = malloc(BUFSIZE * (sizeof(char))))) {   // Dynamically allocate
memory for line
72          fprintf(stderr, "Error: Failed to allocate memory for line buffer: %s\n",
strerror(errno));
73      }
74
75      tokenArgV = malloc(BUFSIZE * (sizeof(char *)));     //Dynamically allocate memory
for token argument vector
76      if (tokenArgV == NULL)
77          fprintf(stderr, "Error: Failed to allocate memory for argument vector: %s\n",
strerror(errno));
78
79      while ((bytesRead = getline(&line, &n, input)) != -1) {   // Read next line from
input
80
81          if (line[0] == '#' || bytesRead <= 1)   // If first character of line is #,
skip to next line
82              continue;
83          else
84          {
85              token = strtok(line, delim);     // Extract tokens from string, delimited
by carriage return + newline
86
87
88              // Perform I/O redirection
89              while (token != NULL)
90              {
91                  if (token[0] == '<') {      // If token starts with <, open filename
and redirect stdin
92                      fdnew = 0;
93                      flags = O_RDONLY;
94                      strcpy(new_path, (token + 1));
95                  }
96                  else if (token[0] == '>') {
97                      if (token[1] == '>') {      // If token starts with >>,
open/create/append filename and redirect stdout
98                          flags = O_WRONLY | O_APPEND | O_CREAT;
99                          strcpy(new_path, (token + 2));
100                     }
101                     else {   // If >, open/create/truncate filename and redirect
stdout
102                         flags = O_WRONLY | O_TRUNC | O_CREAT;
103                         strcpy(new_path, (token + 1));
104                     }
```

```
105                     fdnew = 1;
106                 }
107                 else if (token[0] == '2' && token[1] == '>') {
108                     if (token[2] == '>') {  // If 2>>, open/create/append and
    redirect stderr
109                         flags = O_WRONLY | O_APPEND | O_CREAT;
110                         strcpy(new_path, (token + 3));
111                     }
112                     else {    // If 2>, open/create/truncate and redirect stderr
113                         flags = O_WRONLY | O_TRUNC | O_CREAT;
114                         strcpy(new_path, (token + 2));
115                     }
116                     fdnew = 2;
117                 }
118                 else {
119                     tokenArgV[i++] = token;
120                 }
121                 token = strtok(NULL, delim);
122             }
123             tokenArgV[i] = NULL;
124 
125 
126             // Check input for built-in commands
127             if (strcmp(tokenArgV[0], "pwd") == 0) { // Use built-in pwd
128                 builtpwd();
129             }
130             else if (strcmp(tokenArgV[0], "cd") == 0) { // Use built-in cd
131                 builtcd(tokenArgV[1]);
132             }
133             else if (strcmp(tokenArgV[0], "exit") == 0) {  // Use built-in exit
134                 builtexit(tokenArgV[1]);
135             }
136 
137 
138             // Perform fork/exec and time command execution
139             else
140             {
141                 int fd;      // File descriptor
142                 int pid, status;    // Child process id and exit status
143 
144                 clock_t start, end;     // For storing time information
145                 struct tms time_start, time_end;
146 
147                 if ((start = times(&time_start)) == -1) {   // Begin timing command
    execution
148                     fprintf(stderr, "Error: Failed to start timing command: %s\n",
    strerror(errno));
149                     return -1;
150                 }
151 
152                 // Use fork to create duplicate child process, test return values
153                 if ((pid = fork()) == -1) {     // Fork process fails
154                     fprintf(stderr, "Error: Failed to fork process: %s\n",
    strerror(errno));
155                     exit(-1);
156                 }
157                 else if (pid == 0) {    // Fork succeeds
158                     if (fdnew > -1)
159                     {
```

```
160                         if ((fd = open(new_path, flags, 0666)) != -1) {     // I/O
Redirection: duplicate file and close old file descriptor
161                             if (dup2(fd, fdnew) == -1) {
162                                 fprintf(stderr, "Error: Failed to duplicate file for
I/O redirection: %s\n", strerror(errno));
163                                 return -1;
164                             }
165                             else if (close(fd) == -1) {
166                                 fprintf(stderr, "Error: Failed to close file for I/O
redirection: %s\n", strerror(errno));
167                                 return -1;
168                             }
169                         }
170                         else {
171                             fprintf(stderr, "Error: Failed to open file %s for I/O
redirection: %s\n", new_path, strerror(errno));
172                             return -1;
173                         }
174                     }
175                     if (execvp(tokenArgV[0], tokenArgV) == -1) {   // Execute program
via program's name and argument vector
176                         fprintf(stderr, "Error: Failed to execute command %s: %s\n",
tokenArgV[0], strerror(errno));
177                         return -1;
178                     }
179                 }
180                 else {
181                     if (wait(&status) == -1) {  // Wait for process to change state.
182                         fprintf(stderr, "Error: Failed to wait for the child process
%s to complete\n", tokenArgV[0], strerror(errno));
183                         return -1;
184                     }
185                     if ((end = times(&time_end)) == -1) {   // Time command execution
186                         fprintf(stderr, "Error: Failed to get end timing of
command:%s\n", strerror(errno));
187                         return -1;
188                     }
189 
190                     long clktck = clktck = sysconf(_SC_CLK_TCK);    // Number of
clock ticks per second for kernel
191 
192                     fprintf(stderr, "Command returned exit status: %d\n", status);
  // Print shell messages
193                     fprintf(stderr, "consuming %f real seconds, %f user, %f
system.\n", (end-start) / (double) clktck,  (time_end.tms_cutime-
time_start.tms_cutime) / (double) clktck, (time_end.tms_cstime-time_start.tms_cstime)
/ (double) clktck);
194                 }
195             }
196             fdnew = -1;     // Reset parameters for next input line
197             flags = 0;
198             i = 0;
199         }
200     }
201 
202     free(line);     // Free dynamically allocated memory
203     free(new_path);
204     free(tokenArgV);
205 
206     return 0;
```

```
207 }
208
209 // Built in functions
210 int builtpwd() {    // Built-in pwd command
211     char *WDpath;   // Buffer for holding current working directory path
212     if (!(WDpath = malloc(BUFSIZE * sizeof(char)))) {
213         fprintf(stderr, "Error: Failed to allocate memory for current working
    directory path: %s\n", strerror(errno));
214         return -1;
215     }
216     if (!(getcwd(WDpath, BUFSIZE))) {
217         fprintf(stderr, "Error: Failed to retrieve current working directory path:
    %s\n", strerror(errno));
218         return -1;
219     }
220     else {
221         printf("%s\n", WDpath);    // Print current working directory
222     }
223     free(WDpath);
224     return 0;
225 }
226
227 int builtcd(char *CDpath) {    // Built in cd command
228     if (CDpath == NULL && chdir (getenv ("HOME")) == -1 || CDpath != NULL && chdir
    (CDpath) == -1) {    // Receive HOME path from environment variable if no argument
    given, use given path otherwise.
229         fprintf(stderr, "Error: Failed to change to directory %s: %s\n", CDpath,
    strerror(errno));
230         return -1;
231     }
232
233     return 0;
234 }
235
236 void builtexit(char *exit_status) {     // Built-in exit command
237     if (exit_status == NULL)   // Default exit status = 0
238         exit(0);
239     else
240         exit(atoi(exit_status));   // Convert from string to integer and use given
    exit status.
241 }
```