Lab2 — UNIX Shell (Part II)

Due: 11:59pm Thursday, 2/20/2020

Group Size: 1, which means you should finish this lab assignment by yourself.

Goal: Getting familiar with signal handling.

Introduction: This lab assignment is a further step based on the UNIX Shell interface you built in lab 1. This time, you need to add history feature into your UNIX Shell. The history feature allows users to access up to ten most recently entered commands. These commands will be numbered starting at 1 and will continue to grow larger even past 10, e.g. if the user has entered 35 commands, the ten most recent commands should be numbered 26 to 35.

A user will be able to list ten most recently entered commands when he/she presses <Control><C>, which sends the *SIGINT* signal to the shell process via OS kernels. UNIX systems use **signals** to notify a process that a particular event has occurred. Signals may be either synchronous or asynchronous, depending on the source and the reason for the event being signaled. Once a signal has been generated by the occurrence of a certain event (e.g., division by zero, illegal memory access, user entering <Control><C>, etc.), the signal is delivered to a process where it must be **handled**. After receiving a signal, a process may handle it by one of the following techniques:

- Ignoring the signal
- Using the default signal handler, or
- Providing a separate signal-handling function.

Signals may be handled by first setting certain fields in the C structure *struct sigaction* and then passing this structure to the *sigaction()* function. Signals are defined in the include file /usr/include/sys/signal.h. For example, *SIGINT* represents the signal for terminating a program with the control sequence <Control><C>. The default signal handler for *SIGINT* is to terminate the program.

Alternatively, a program may choose to set up its own signal-handling function by setting the *sa_handler* field in *struct sigaction* to the name of the function which will handle the signal and then invoking the *sigaction()* function, passing it (1) the signal we are setting up a handler for, and (2) a pointer to *struct sigaction*.

In Figure 3 we show a C program that uses the function *handle_SIGINT()* for handling the *SIGINT* signal. This function prints out the message "Caught Control C" and then invokes the *exit()* function to terminate the program.

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

#define BUFFER_SIZE 50

static char buffer[BUFFER_SIZE];

/* the signal handler function */
void handle_SIGINT() {
        write(STDOUT_FILENO,buffer,strlen(buffer));

        exit(0);    // This is only for illustration purposes. You may want to
                    // delete the exit(0) statement for this lab assignment.
}

int main(int argc, char *argv[])
{
        /* set up the signal handler */
        struct sigaction handler;
        handler.sa_handler = handle_SIGINT;
        sigaction(SIGINT, &handler, NULL);

        strcpy(buffer,"Caught <ctrl><c>\n");

        /* wait for <control> <C> */
        while (1)
                ;

        return 0;
}
```

Figure 3. example.c

This program will run in the *while (1)* loop until the user enters the sequence <Control><C>. When this occurs, the signal-handling function *handle_SIGINT()* is invoked.

The signal-handling function should be declared above *main()* and because control can be transferred to this function at any point, no parameters may be passed to it. Therefore, any data that it must access in your program must be declared globally, i.e. at the top of the source file before your function declarations.

If the user enters <Control><C>, the signal handler will output a list of the most recent 10 commands, followed by the command prompt. With this list, the user can run any of the previous 10 commands by entering r x where 'x' is the first letter of that command. If more than one command starts with 'x', execute the most recent one. If no command starts with 'x', print out an error message and continue accepting user inputs. Also, the user should be able to run the most recent command again by just entering 'r'. You can assume that only one space will separate the 'r' and the first letter and that the

letter will be followed by '\n'. Again, 'r' alone will be immediately followed by the '\n' character if it is wished to execute the most recent command.

Any command that is executed in this fashion should be echoed on the user's screen and the command is also placed in the history buffer as the next command. (r x does not go into the history list; the actual command that it specifies, though, does.)

Submission: Put all your modification in the file "shell.c" and only submit this file on Canvas. At the beginning of the file "shell.c". You need to create a readme file to show how to compile your file, run your compiled program and make sure your instructions work.