BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE PILANI, K. K. BIRLA GOA CAMPUS

I SEMESTER 2019-2020

Operating Systems (CS F372)

Assignment 3 [Non-evaluative Assignment]

QUESTION 1: *Spiderman to the rescue!!!*

There is an NxN dimensional maze with each entry either 0 or 1.

Spiderman is at (0, 0). Mary Jane (MJ) is held up by the Green Goblin at ((N-1), (N-1)). Spiderman wants to rescue MJ as fast as possible. i.e.

Source: Top left corner of the matrix whose entry is 0. (Spiderman)

Destination: Bottom right corner of the matrix whose entry is 2. (MJ)

To get a quick way out, Spiderman asks his spies to help him for finding MJ. Each spy represents a process and Spiderman is the main process. Initially Spiderman who is at (0, 0), calls two spies which will check (0, 1) and (1, 0) entries. Depending on those entries, each will decide whether to call one or two other spies. If the spy is on the last row or column of the maze, it will call only one spy otherwise it will call two spies. This continues till a spy reaches MJ or no spy can find a way further.

Which means: Each process at (i, j) will have either one or two child processes which can find the possible path in the maze. These child processes read the entries that are to the right- (i, j+1) of and below- (i+1, j) the current entry- (i, j).

The right child in the tree (shown in Figure 2) corresponds to the child reading the entry to the right of the current process whereas the left child in the tree (shown in Figure 2) corresponds to the child reading the entry below the current entry.

In the maze, the path is created by 0's whereas 1's are the obstructions and 2 is the final destination where MJ is. So, if you have a 0 as the entry for current process, you will have a path further from that node by creating either one or two child processes. If the entry for current process is 1, you cannot have a path from that process and it dies.

The input will be such that there is only one possible path from the source to the destination.

You have to print the path in reverse order where each process that participated in the actual path must print the corresponding entry as an (i, j) pair.

The following is the input matrix.

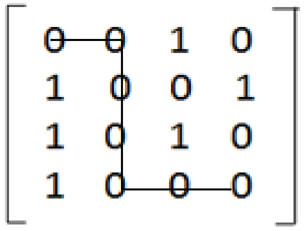

Figure 1 : Input Matrix

The corresponding process tree for the above given input matrix will be:

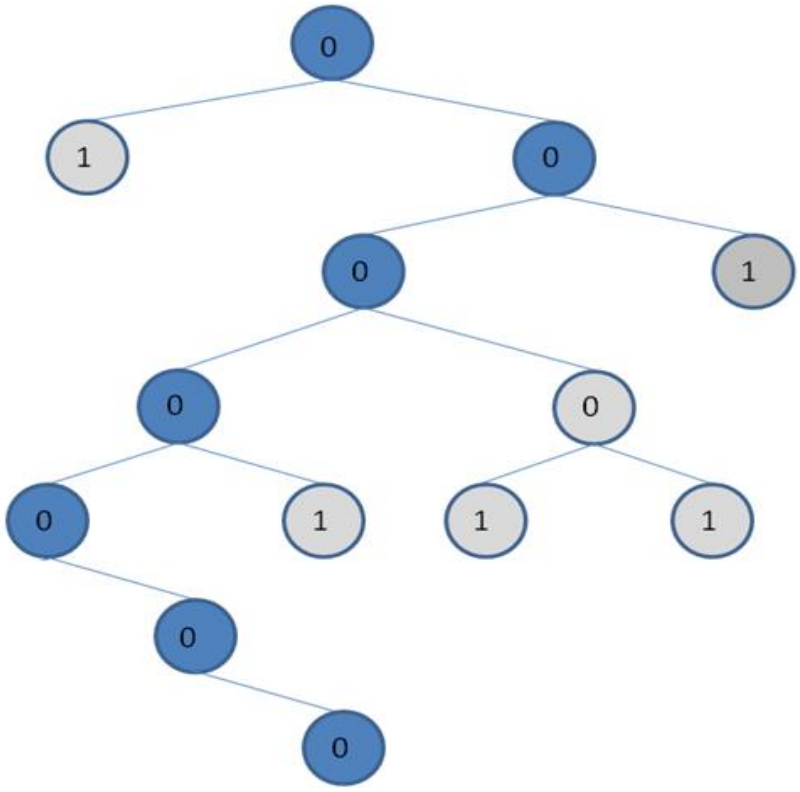

Figure 2 : The process tree

Sample Input 1:

$$\begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

Corresponding Output 1: (3,3) -> (3,2) -> (3,1) -> (2,1) -> (1,1) -> (0,1) -> (0,0)

MJ successfully rescued.

Sample Input 2:

$$\begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \end{bmatrix}$$

Corresponding Output 2:

Spiderman cannot rescue MJ.

Question 1: Shell Command line Interpreter

This program is to test your understanding in the following:

fork, signal, wait/waitpid, exit, exec, dup/dup2

In this question your task is to write a replacement for the shell command line interpreter. Typical command line shells under UNIX, which you may have used, are bash, sh, csh, tcsh. The goal of this assignment is to increase your understanding of processes and the role of system software.

Your shell provides a sample interface, prompting the user for commands, and then interpreting the supplied commands. It should have the following features, which constitute an informal specification for your program.

Input

The shell reads an input line at a time from the standard input until the end of line is reached. Each input line is treated as a sequence of commands separated by a pipe character (|). For example: `ls –l | grep alpha | more`

(There can be one or more white spaces separating the words – command name, options and pipe)

In this example there are three commands – the command ls -l, the command grep alpha, and the command more. Each command consists of words separated by one or more spaces. No special extra command editing facilities are necessary beyond what is already done by regular terminal input (example see man tty for the default editing features namely backspace, erase line, erase word and end of file).

Internal commands

Your program does not need to interpret any internal shell commands except the following ones

entry

The command line interpreter will start functioning with this command. Any external or internal command issued before the entry command should display “Command line interpreter not started” message.

exit

The command line interpreter exits. Any external or internal command issued after the exit command should display “Command line interpreter exited” message.

log

The command line interpreter starts the command and output logging into “command.log” and “output.log” files respectively with the log command.

unlog

The command line interpreter stops the command and output logging with the unlog command.

viewcmdlog

The viewcmdlog command displays the content of command.log file.

viewoutlog

The viewoutlog command displays the content of output.log file.

changedir <argument>

The changedir command takes the new present working directory path as argument. If the path is valid then the current working directory will be changed to the new directory specified in the argument. Example: changedir /home/usr/bin

Directory changed. The present working directory is /home/usr/bin

External commands

All commands that are not internal commands are external commands. An external command is a sequence of words separated by white space(s) of which the first is the executable name of the command and the remainder is the arguments to the command. If no executable file was found at any of the environment paths, the shell must check the same in the current working directory. If the executable still not found then the shell command line interpreter should print the error message: “The command <command name> not found”. If the executable files are found, the shell should create a new process to run the programs. The child process must use the exec() family of system calls to run each program together with the supplied arguments and shell environment variables.

Processing of commands

The tricky thing about this assignment is that you have to arrange for the piping between processes for each command. For example in the example given before, the stdout of ls must be piped to the stdin of grep alpha. The stdout of grep alpha must be fed to the stdin of more.

Your program must work for arbitrary numbers of processes on the command line, with arbitrary number of parameters. You would normally use dup2 to achieve this.

Your shell should record all input commands by appending to a log file called “command.log”. The format of “command.log” may be as shown below

```
<Command name>  <Date>  <Time of execution>  <Action taken (success/failure)>
```

Your program should record the output of all commands, including the intermediate commands, by appending to a log file called “output.log”. The format of the “output.log” file is such that each line of command output is associated with the command that generated it. For example the command ls | grep alpha | more should add the following entries in output.log file. (I would strongly recommend you to follow this format).

```
[ls]
<output of ls command>

[ls] | [grep alpha]
<output of the ls | grep alpha commands>

[ls] | [grep alpha] | [more]
<output of the ls | grep alpha | more commands>
```

Make sure no orphan processes are getting created while executing the commands.

Implement the shell with the help of dup2 and intermediate files.