FACULTY OF ENGINEERING

COMPUTER ENGINEERING DEPARTMENT

CSE3033 - OPERATING SYSTEMS

PROJECT-2 REPORT

Ekin Nohutçu - 150116067

Zeynep Naz Akyokuş - 150119073

Merve Ayer - 150119828

MARMARA
UNIVERSITY

PART A

- We have defined “fileExistenceCheck” function and this function checks for the file exist in the given path or not. It returns 1 if file exist in the given path.

- We have created “findPath” function. When we enter a program name to execute it we need to find this programs’ path so this function defined for finds the path of given program. “findPath” function searches the environment variable with getenv() function and gets the path environment variable then copies the path environment variable into a string and splits that string using “ : “ as delimiter. “ **paths ” we defined this variable as an array of the paths. Then we have created the while loop for fill the “paths” and this loop executes until the whole string is split. Also we have created the for loop in order to concatenate with path and the given program to execute. And also in this part we have checked if file exist for this checking process we calling the “fileExistenceCheck” function that we have defined previously.

- We have defined “addBackgroundProcess” function in order to add a newly created background process which takes & in command line arguments to the end of the queue.

- We have defined the “removeFromBackgroundQueue” function removes the element from the background process queue. It checks the given pid is equal to the background queue header’s pid. If they are equal, it deletes the header. If they are not equal, it finds the process with given pid and delete it.

- “SignalHandler” function calling when the “ ^Z “ entered. It stop the currently running foreground process. If there is no currently running process, then the signal does not affect. If SIG is 0, then no signal is sent but existence and permission checks are still performed. We have also two if statement inside the “SignalHandler” function first if statement for if there is a foreground process and also second if statement for error checking before stopping the foreground process. If there is an error we printed that error. In else statement “kill” function send a “SIGSTOP” signal in order to stop the currently running foreground process.

- In ChildProcessHandler function, we check for the killed process has a child or not. If there is terminated child process, we remove them from background queue by using removeFromBackgroundQueue function.

- We have 2 structs for background processes and commands. We have defined node struct for commands and we have defined background_process struct in order to create a queue for the background processes.

- Execute function takes 3 parameters. One is for the path of the command, second one for the command itself and third one is a background value for that command. Execute function forks a child and this child executes the command by using **execv()**. For the parent part, if the current process is a foreground process, it calls waitpid with child pid and assigns currentForegroundProcess and makes ForegroundProcessCheck as true. If the current process is a background process, it adds this process into the background process queue.

PART B

ps_all:

- We have defined “write_ps_all” function. When the “ps_all” command is received, “write_ps_all” function runs and prints the currently running programs and programs that have terminated but not printed to screen before. These information are stored in “psAllArray” . While we add the new child process to queue with “addBackgroundQueue” function, we also put the process to our psAllArray in form of ‘firefox (Pid=10000)’ to the first empty index in our psAllArray. When a process terminates, we remove it from the queue with “removeFromBackgroundQueue” which takes the terminated process Pid as parameter. We also change the value of the process in “psAllArray” in this function as ‘firefox’.

search:

- “search” function takes two parameters. One is the args array to determine the word to search and the type of the search. The other parameter is the path. Path is “.” as default which helps search in the current directory. First program reads the files and inside the current directory. If it is not a directory and header is .c or .C or .h or .H, It opens the file and reads line by line. If it finds the searched word in line; it will print the line number, file name, and the line. If there is a -r option in the args; search will be recursive, which means we will also search for the word in subdirectories. To do this, it adds the filename to the end of the path every time it gets a directory file.

exit:

- For exit part, it checks for background processes. If there is, we printed an error message. If there is no background process, exit.

PART C

- If the second argument of the command is >, it will create if file does not exist, otherwise it re-write the file. Results of the executable to the given are printed to the terminal.
- If the second argument of the command is >>, it will create if file does not exist, otherwise it appends the file. Results of the executable to the given are printed to the terminal.
- If the second argument of the command is <, it reads the file and uses them to execute the command.
- If the second argument of the command is < and the fourth argument of the command is >, it reads the file and uses them to execute the command and write the output to another file.
- If the second argument of the command is 2>, writes the standard error of executable to the given file in arguments.