Operating Systems Project 1

Necla Mutlu - Hazal Mengüaslan

14 Mart 2020

1 Implementation

1.1 Part 1: Replace Execvp with Execv

To complete this part, we declared a new function called "execute command". In this function, we took the path variable and tokenized it with ":". For each directory in the path variable, we checked whether we can access an executable "directory/command-¿name" in that directory. When we found it, we supplied this full path to the execv with the parameters of the command.

1.2 Part 2

1.2.1 I/O Redirection

For I/O redirection, we checked the command-¿redirects. For redirecting command-¿redirects[0], which is supposed to be stdin channel, we opened the file given in command-¿redirects[0] with necessary permissions and we assigned the file descriptor of stdin to it by using dup2 function.

1.2.2 Piping

To complete the piping, we checked whether the command-¿next is null or not. If it is not, we should connect the output of the command with its command-¿next and this should be done recursively. For that purpose, we created a pipe and we forked once. To the write end of this pipe, the parent process, which is currently running command, can write its stdout. We again used dup2 to assign stdout file descriptor to the pipe's write end. The child process is responsible from the execution of the next command. And we assigned stdin of the next command to pipe's read end by using dup2, so that it takes its input from the current command. The child process called process command function again for the next command in case this command has also a next.

1.3 Part 3

1.3.1 Auto-complete

We divided auto-complete into three parts:

- User executables in the current directory
- Newly added commands: Psvis, mybg, myfg,myjobs, pause
- Linux executables

In all of those parts, we used ls and grep to complete the task. In the user executables case, after some modifications on the name of the command, we executed "ls — grep command-¿name" command via popen and stored the results in a file. And by reading lines in this file, we print the results. For newly added commands, we executed "echo myjobs mybg myfg pause psvis — grep" command and we stored the results in another file. We did a similar thing for the linux executables. We tokenized the path and for each directory in the path we executed "ls directory — grep command-¿name" command and got the results into a file. While printing, we developed a mechanism to understand whether there is only one result or there are more results. We had a counter, which will count the results coming from new commands and we had a seperate counter for the results coming from user executables. If the total number of results is 1 and the command-¿name given by the user exactly matches with the result, we created an ls command and by using process command function, we executed it.

1.4 Job Management

1.4.1 Myjobs

To satisfy the given task, we executed "ps -u user –format pid,stat,ucmd" command by using shellgibi's process command function.

1.4.2 Mybg

Mybg sends a SIGCONT signal to the given pid by using kill function.

1.4.3 Pause

Pause sends a SIGSTOP signal to the given pid by using kill function.

1.4.4 Myfg

To satisfy the expected execution myfg, we created a child process by forking and we assigned some jobs to the parent and the child. The child process sends a SIGCONT signal to the process with given pid. As the process should execute in the foreground, the parent should wait until the process with the given pid terminates or is stopped by another process. We used waitid function for that purpose.

1.5 Psvis

Our kernel module only traverses the subprocess tree whose root is the given pid. It prints the pid's and start times of the children with printk. Unfortunately, we could not be able to visualize the results we got from the kernel module. For visualizing, we wrote another command psvis2 that does not use a kernel module.

1.6 Psvis2

We used graphiz to create human readable graph of the subprocess tree. Draw.c file in the folder contains the code for this command. We compiled it and created an executable called psvis2. We called this psvis2 executable inside the shellgibi and piped results into dot -Tx11 command. Psvis2 executable's output is a diagraph structure. A diagraph structure consists of nodes and edges and it is a kind of list that is formatted in a way that when compiled with dot -Tx11 command, it gives us a png of a human readable graph. Psvis2 uses a recursive function to traverse all the children of the process with the given id. It traverses all the subprocess tree and adds edges and nodes to the diagraph.

1.7 Alarm

For alarm command, we created a play alarm function that will handle the process. In this function, we created a pipe. To the write end of that pipe, the child process writes the cronjob string. By cronjob string, I mean "minutes hours * * * aplay musicGivenByUser". Stdin of the parent process is assigned to the read end of the pipe. Parent takes this cron string input and calls "crontab" command with it, which writes the string to the crontab file.

1.8 Custom Command

1.8.1 bbc

bbc command prints the headlines of the day from BBC. Parser.py is the code for this command. It uses python's beautiful soup library to extract specific data from the html templates. It extracts headlines from "https://www.bbc.com/news/world". You can also use piping with this command. In order to run this command, python3 and requests package must be installed.

1.9 gitUserCheck <username>

This command connects to GitHub and checks with given username whether user exists or not on GitHub. We connect to "https://github.com/" with username at the end of it. If url returns success, we know that we have a user, else we don't have a user with that username. In order to run this command, python3 and requests package must be installed.

2 Non-Working Parts

- Psvis kernel module does not visualize the process tree.
- To be able to run custom commands, python 3 should be installed in your computer.

3 References

We benefited from the following website to use graphiz for psvis2 command.
https://stackoverflow.com/questions/53024715/creating-process-tree-using-fork/53033406