Orçun Özdemir, 54020
Buğra Sipahioğlu, 53694

COMP304 Shelldon: Project 1 Report

Introduction

We defined a child process before creating shell commands, since child uses what has been entered as input and manipulates them accordingly. While shell code runs, we forked the child process and wrote in it.

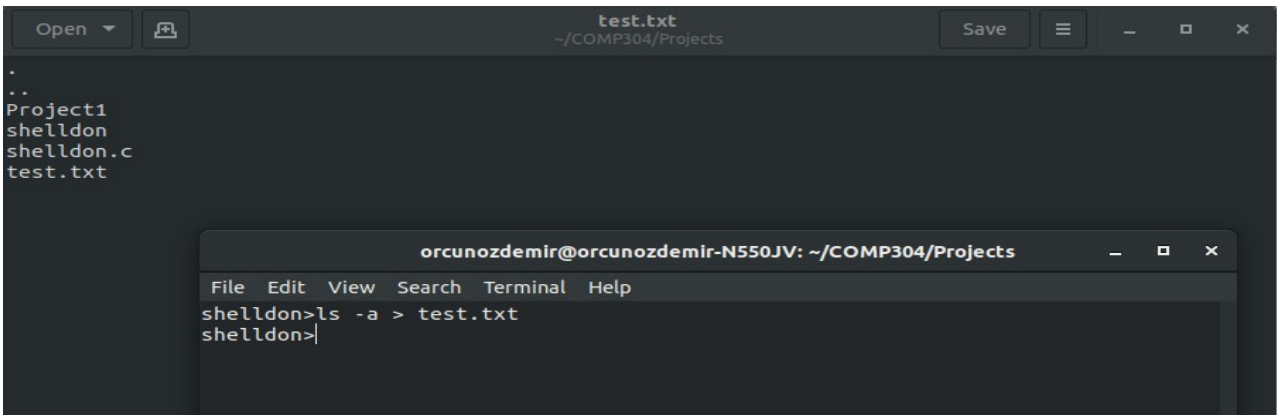
Part 1

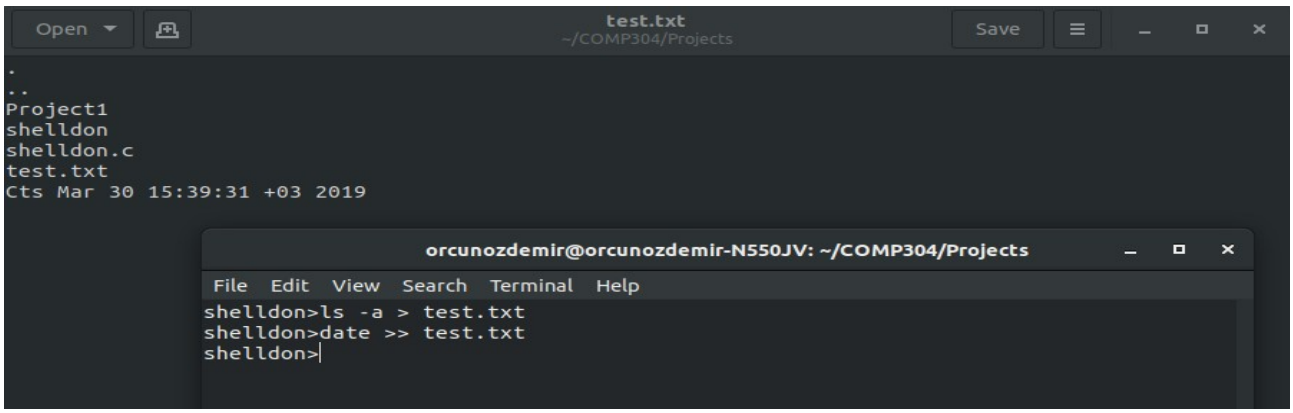
We developed the shelldon code that uses all terminal commands except cd. First, code is redirected in a temporary file that includes user command combined with which command. (Absolute path is written and used as an argument in execv function.)

```
shelldon>ls -a
.  ..  CodeSearch1  Hayaletler.mp3  Project1  shelldon  shelldon.c  test.txt
shelldon>pwd
/home/orcunozdemir/COMP304/Projects
shelldon>gcc
gcc: fatal error: no input files
compilation terminated.
shelldon>date
Cts Mar 30 17:27:43 +03 2019
shelldon>mkdir test
shelldon>ls -a
.   CodeSearch1     Project1  shelldon.c  test.txt
..  Hayaletler.mp3  shelldon  test
shelldon>rmdir test
shelldon>ls -a
.  ..  CodeSearch1  Hayaletler.mp3  Project1  shelldon  shelldon.c  test.txt
```

Part 2

Redirection > command succesfully truncates ls -a command to a test.txt file, in addition >> command successfully appends date command to a test.txt. These examples, can be observed in following figures.





When user enters some commands in shelldon and wants to execute the latest command they executed, a simple !! command in shelldon will be enough. Also, !! will not be in the command history. As a more developed approach, when user wants to execute specific command that they already entered, simple ! N in shelldon will be enough, N is the latest Nth command user entered. Example output can be observed in the next figure.

```
shelldon>ls
userCommandPath: --> /bin/ls
'Backup Codes'                  crontab_music  'Kernel Module'   shelldon.c
 COMP304-Project1-Spring19.pdf   denemeeee       shelldon         shelldon.c~
shelldon>ps -l
userCommandPath: --> /bin/ps
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000 25794 25690  0  80   0 -  7809 wait   pts/0    00:00:00 bash
0 S  1000 25886 25794  0  80   0 -  1130 wait   pts/0    00:00:00 shelldon
4 R  1000 25901 25886  0  80   0 -  9358 -      pts/0    00:00:00 ps
shelldon>pwd
userCommandPath: --> /bin/pwd
/home/bugra/Desktop/Comp 304/Projects/P1
shelldon>history

4 history
3 pwd
2 ps -l
1 ls
shelldon>!3

/home/bugra/Desktop/Comp 304/Projects/P1
shelldon>ls
userCommandPath: --> /bin/ls
'Backup Codes'                  crontab_music  'Kernel Module'   shelldon.c
 COMP304-Project1-Spring19.pdf   denemeeee       shelldon         shelldon.c~
shelldon>history

6 history
5 ls
4 history
3 pwd
2 ps -l
1 ls
shelldon>!!
'Backup Codes'                  crontab_music  'Kernel Module'   shelldon.c
 COMP304-Project1-Spring19.pdf   denemeeee       shelldon         shelldon.c~
shelldon>
```

Part 3

```
shelldon>codesearch "foo"
182: ./shelldon.c ->  if (args[1][0] == '"' && args[2] == NULL) /* Regular Code
Search: 'codesearch " foo "' */
198: ./shelldon.c ->  { /*targeted search: codeseach "foo" -f ./include/util.h *
/
212: ./shelldon.c ->  { /* shelldon> codesearch -r "foo" //recursive usage */
539: ./shelldon.c ->  for (i = 0; i < numberOfLines; i++) /* 45: ./foo.c -> void
 foo(int a, int b); */

shelldon>
```

```
File  Edit  View  Search  Terminal  Help
shelldon>codesearch "foo" -f ./CodeSearch1/CodeSearch2/CodeSearch3/CodeSearch3Te
st.c
182: ./CodeSearch1/CodeSearch2/CodeSearch3/CodeSearch3Test.c ->  if (args[1][0]
== '"' && args[2] == NULL) /* Regular Code Search: 'codesearch " foo "' */
198: ./CodeSearch1/CodeSearch2/CodeSearch3/CodeSearch3Test.c ->  { /*targeted se
arch: codeseach "foo" -f ./include/util.h */
212: ./CodeSearch1/CodeSearch2/CodeSearch3/CodeSearch3Test.c ->  { /* shelldon>
codesearch -r "foo" //recursive usage */
539: ./CodeSearch1/CodeSearch2/CodeSearch3/CodeSearch3Test.c ->  for (i = 0; i <
 numberOfLines; i++) /* 45: ./foo.c -> void foo(int a, int b); */
```

```
shelldon>codesearch -r "foo"
182: ./CodeSearch1/CodeSearch2/CodeSearch3/CodeSearch3Test.c ->  if (args[1][0] == '"' && args[2] == NULL) /* Regular Code Search: 'codesearch " foo "' */
198: ./CodeSearch1/CodeSearch2/CodeSearch3/CodeSearch3Test.c ->  { /*targeted search: codeseach "foo" -f ./include/util.h */
212: ./CodeSearch1/CodeSearch2/CodeSearch3/CodeSearch3Test.c ->  { /* shelldon> codesearch -r "foo" //recursive usage */
539: ./CodeSearch1/CodeSearch2/CodeSearch3/CodeSearch3Test.c ->  for (i = 0; i < numberOfLines; i++) /* 45: ./foo.c -> void foo(int a, int b); */
182: ./CodeSearch1/CodeSearch1Test.c ->  if (args[1][0] == '"' && args[2] == NULL) /* Regular Code Search: 'codesearch " foo "' */
198: ./CodeSearch1/CodeSearch1Test.c ->  { /*targeted search: codeseach "foo" -f ./include/util.h */
212: ./CodeSearch1/CodeSearch1Test.c ->  { /* shelldon> codesearch -r "foo" //recursive usage */
539: ./CodeSearch1/CodeSearch1Test.c ->  for (i = 0; i < numberOfLines; i++) /* 45: ./foo.c -> void foo(int a, int b); */
182: ./shelldon.c ->  if (args[1][0] == '"' && args[2] == NULL) /* Regular Code Search: 'codesearch " foo "' */
198: ./shelldon.c ->  { /*targeted search: codeseach "foo" -f ./include/util.h */
212: ./shelldon.c ->  { /* shelldon> codesearch -r "foo" //recursive usage */
539: ./shelldon.c ->  for (i = 0; i < numberOfLines; i++) /* 45: ./foo.c -> void foo(int a, int b); */
182: ./Project1/shelldon.c ->  if (args[1][0] == '"' && args[2] == NULL) /* Regular Code Search: 'codesearch " foo "' */
198: ./Project1/shelldon.c ->  { /*targeted search: codeseach "foo" -f ./include/util.h */
212: ./Project1/shelldon.c ->  { /* shelldon> codesearch -r "foo" //recursive usage */
539: ./Project1/shelldon.c ->  for (i = 0; i < numberOfLines; i++) /* 45: ./foo.c -> void foo(int a, int b); */

shelldon>
```

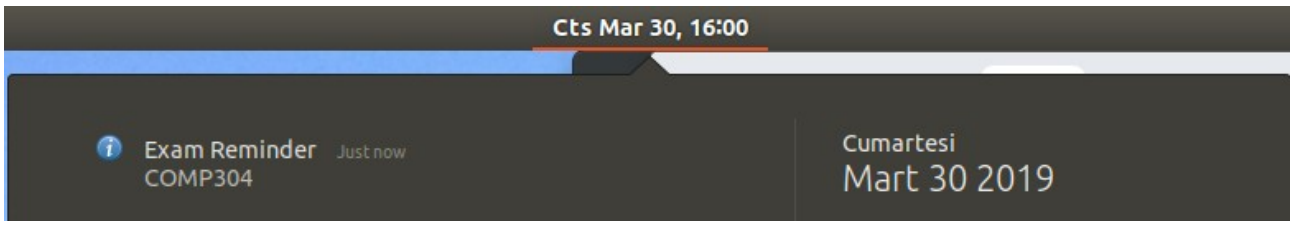
Codesearch command takes an input from the user and if there are not any arguments between codesearch and input that is going to be searched, codesearch searches its current directory.

Then, returns the line of the code, name of the function and name of the code. If codesearch has an “-r” argument between input and codesearch, codesearch searches recursively and looks for current directory and its sub-directories. Then, returns the line of the code, name of the function and the name of the code. If codesearch han an “-f” argument between input and codesearch, codesearch searches the second input which is a directory and only checks whether the first input is in second input directory or not.

```
shelldon>birdakika 7.15 Hayaletler.mp3
no crontab for orcunozdemir
shelldon>crontab -l
5 7. * * * mpg321 /home/orcunozdemir/COMP304/Projects/Hayaletler.mp3
6 7. * * * pkill mpg321
shelldon>
```

Birdakika command takes two inputs which are targeted time and name of the song. After, shelldon runs the command, and create a txt file to bash, then we bashed into a crontabfile through certain parsing operations. After, we schedule the process to crontab, we need to kill it since song has to play only for a minute. Thus, we did similar operation but this time we used pkill and scheduled it into a minute later then the first schedule.

```
shelldon>exam -add COMP304 31.03 16:00
modifedDay: 30
LINE 3:  echo "00 16 30 03 * XDG_RUNTIME_DIR=/run/user/$(id -u) notify-send 'Exa
m Reminder'  'COMP304' " >> cronFileExam
```



Exam -add command takes three inputs which are course name, exam date and exam time. When user uses this command, program parses the date,time and course name to similar file we implemented in birdakika command. In this case, crontab schedules a notify-sender from a day before the exam date. When, the time is come, a pop up shows up in linux and warns user about the exam. We had an issue about crontab scheduling, terminal command of notify-send was not the same as crontab schedule command. Therefore, after some research we found a certain command and parsed into crontab to handle the issue.

Part 4

In this part, we iterated all the process until we find the process id with p, which is the user input. When the target process is found, a recursive function searches for all the process tree starting from target process(as root). In each recursive call, all the children processes are examined and the one with minimum PID(oldest) is printed out.