Programming Assignment (PA) – 1
(Shell Command Execution Simulation in C)

CS307- Operating Systems

30 March 2021

Solution Report

Ataollah Hosseinzadeh Fard
ID: 28610

Problem and Solution Description

It is needed to first set a command and an option to following statement, and then execute it in a C program.

man ***<command>*** | grep ***<option>*** > output.txt

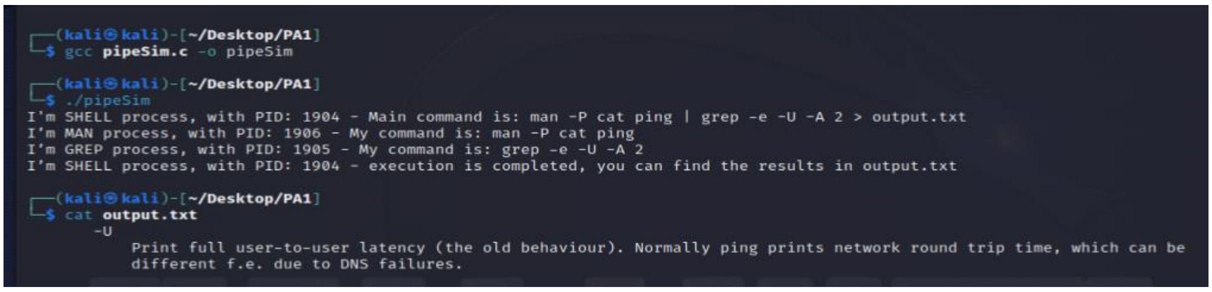
Therefore, I selected “ping” as command and “-U” as option, but I added some extra flags to solve some simple problems. So, the bash command that I must execute in C is following:

man -P cat ping | grep -e -U -A 2 > output.txt

“ping” is a commonly used tool that is used to test the reachability of a network host or server by sending a series of Internet Control Message Protocol (ICMP) packets to it and measuring the response time. “man ping” opens manual page of “ping” command, but “man” command uses “more” tool as pager which I thought it might cause some I/O problems, so with using “-P” flag I specified that I want “cat” tool to be used as pager; therefore, “man -P cat ping” opens manual page of “ping” using “cat” as pager.

the option for “grep” that I selected, “-U”, show definition of “-U” flag in “ping” tool, but there is a little problem, while using “grep” if the pattern given has ‘-‘ in its beginning, “grep” will not understand it as pattern and will think that the given option is a flag, so in order to solve this issue “-e” flag should be given so if “-e” flag is given, the pattern can start with ‘-‘. Thus, “grep -e -U” will search for ‘-U’ inside manual page of “ping” tool. There is also another problem with “grep”, when a pattern given, it searches for pattern and will only print single line of each match, but here I want full definition of ‘-U’ flag. I looked up and found that ‘-U’ has only 2 lines of definition, so by giving following “-A 2”, I can specify that after the match print extra 2 lines as well to “grep”. To sum up, “man -P cat ping | grep -e -U -A 2”, will find definition of “-U” flag of “ping” command from manual page and print it. But “> output.txt” says that output should be written into “output.txt”.

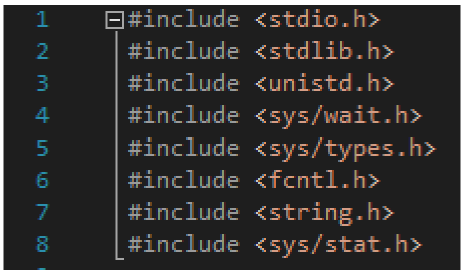
Following image is a demo of running my C program and reading output.txt.



```
┌──(kali㉿kali)-[~/Desktop/PA1]
└─$ gcc pipeSim.c -o pipeSim

┌──(kali㉿kali)-[~/Desktop/PA1]
└─$ ./pipeSim
I'm SHELL process, with PID: 1904 - Main command is: man -P cat ping | grep -e -U -A 2 > output.txt
I'm MAN process, with PID: 1906 - My command is: man -P cat ping
I'm GREP process, with PID: 1905 - My command is: grep -e -U -A 2
I'm SHELL process, with PID: 1904 - execution is completed, you can find the results in output.txt

┌──(kali㉿kali)-[~/Desktop/PA1]
└─$ cat output.txt
       -U
           Print full user-to-user latency (the old behaviour). Normally ping prints network round trip time, which can be
           different f.e. due to DNS failures.
```

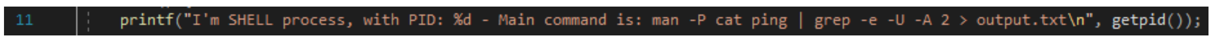
Implementation

Step 1) added needed libraries.



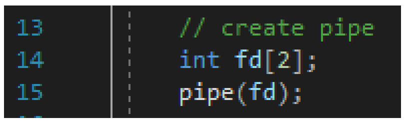
```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#include <sys/types.h>
#include <fcntl.h>
#include <string.h>
#include <sys/stat.h>
```

Step 2) print a message that indicates we are in parent process, with its process ID.



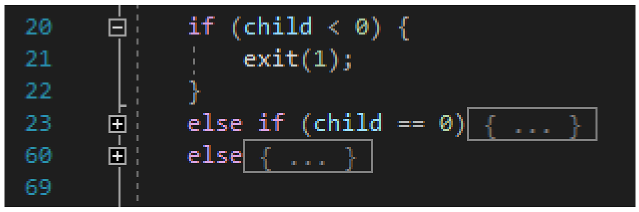
```
printf("I'm SHELL process, with PID: %d - Main command is: man -P cat ping | grep -e -U -A 2 > output.txt\n", getpid());
```

Step 3) created a pipe, to be able to write to and read from Anon file.



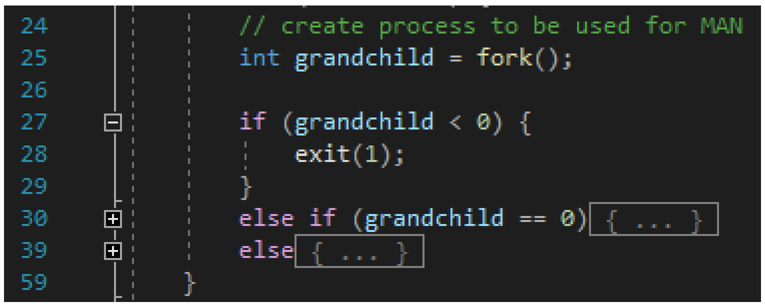
```
// create pipe
int fd[2];
pipe(fd);
```

Step 4) forked and created a child process that will be used for “grep” command execution. In this part, if child integer is negative, the fork operation is failed, if it is 0, we are in child process, and if it is positive integer, we are in parent process.



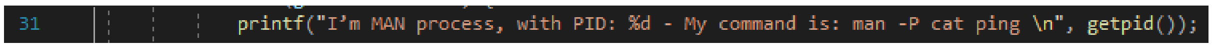
```
if (child < 0) {
    exit(1);
}
else if (child == 0){ ... }
else{ ... }
```

Step 5) forked and created a grandchild process that will be used for “man” command execution. In this part, if child integer is negative, the fork operation is failed, if it is 0, we are in child process, and if it is positive integer, we are in parent process.



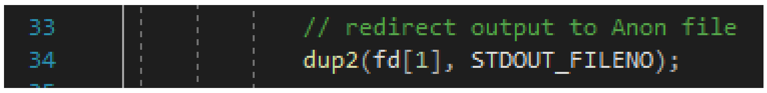
```
    // create process to be used for MAN
    int grandchild = fork();

    if (grandchild < 0) {
        exit(1);
    }
    else if (grandchild == 0){ ... }
    else{ ... }
}
```

Step 6) If we are inside the grandchild process, it should print indicator message with its process ID.



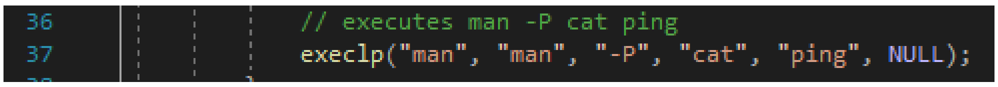
```
printf("I’m MAN process, with PID: %d - My command is: man -P cat ping \n", getpid());
```

Step 7) now program should redirect the output from console to Anon file, this means whatever output we will get, it will be written to Anon file.



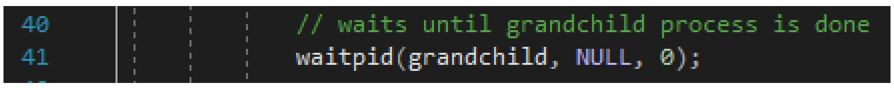
```
// redirect output to Anon file
dup2(fd[1], STDOUT_FILENO);
```

Step 8) now we should execute “man -P cat ping”. “man option” is a command that opens manual page of given option. “-P option”, normally “man” uses “more” as pager, program to display, but with using “-P” flag we can specify any other pager option to use, in this case I used “cat” as pager. Basically, this command opens manual page of “ping” command using “cat” pager.



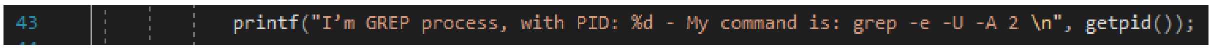
```
// executes man -P cat ping
execlp("man", "man", "-P", "cat", "ping", NULL);
```

Step 9) child process, should wait for grandchild process to finish.



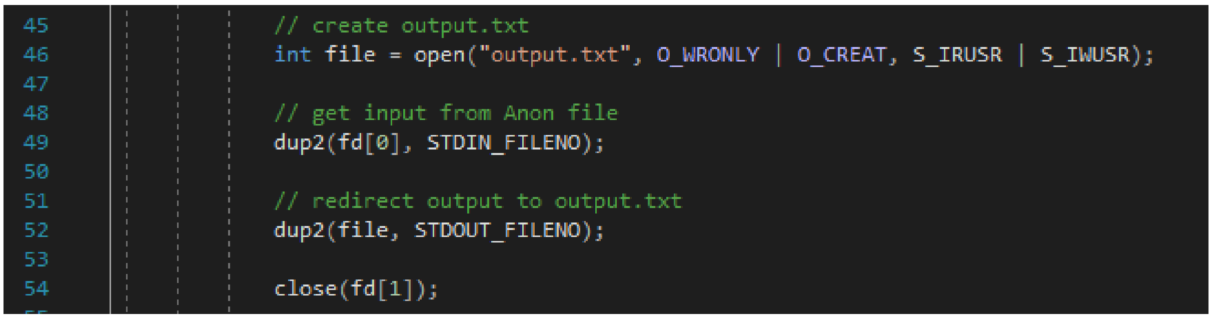
```
// waits until grandchild process is done
waitpid(grandchild, NULL, 0);
```

Step 10) if we are in child process, it should print indicator message with its process ID.



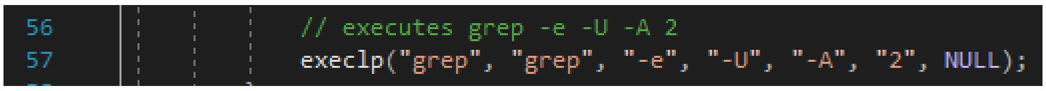
```c
            printf("I’m GREP process, with PID: %d - My command is: grep -e -U -A 2 \n", getpid());
```

Step 11) inside child process, “output.txt” is created, getting input from console is redirected to Anon file, this means that as input program will read from Anon file. Outputs redirected to “output.txt”, this means that instead of printing result in console, result will be written to txt file. And, since there is no need to write to Anon file anymore, we close write-end of pipe; also, this ensures that there will be no extra input in Anon file and act like a “EOF” operator.



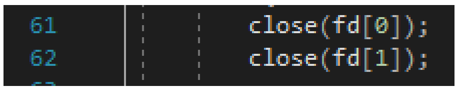
```c
            // create output.txt
            int file = open("output.txt", O_WRONLY | O_CREAT, S_IRUSR | S_IWUSR);

            // get input from Anon file
            dup2(fd[0], STDIN_FILENO);

            // redirect output to output.txt
            dup2(file, STDOUT_FILENO);

            close(fd[1]);
```

Step 12) now we should execute “| grep -e -U -A 2”. “| grep option” is a command that will get output of previous command and try to find any matching part with pattern of “option”. However, in most of the commands there are flags starting with “-“, this means that if try to search for a pattern starting with a “-“, we will be giving flag instead of pattern; Therefore, to solve this problem I used “-e” flag, this ensures that we can give a pattern starting with “-“ so my pattern is “-U”, it will try to find any match with this pattern. Also, by default grep displays exact line of match, but here I want full description of “-U” flag; Therefore, I used “-A 2” this means that display 2 more lines after match, since I knew that “-U” has only 2 lines of description.



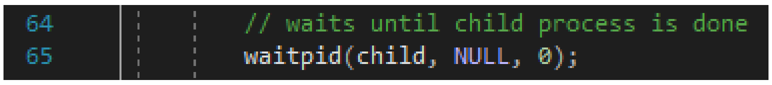
```c
            // executes grep -e -U -A 2
            execlp("grep", "grep", "-e", "-U", "-A", "2", NULL);
```

Step 13) close both write-end and read-end of pipe, since we are done with using Anon file.



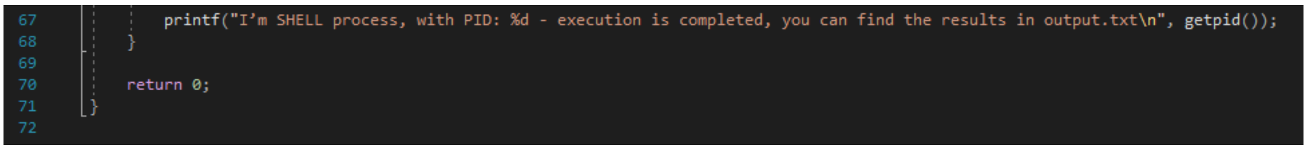
```c
        close(fd[0]);
        close(fd[1]);
```

Step 14) parent process, should wait for child process to finish.



```c
        // waits until child process is done
        waitpid(child, NULL, 0);
```

Step 15) now that program is done, it should print the termination message with parent’s process ID and tell where the results are located.



```c
        printf("I’m SHELL process, with PID: %d - execution is completed, you can find the results in output.txt\n", getpid());
    }

    return 0;
}
```