

Mehmet Acar
1801042095
HW2 – Report

How I Solved the Problem?

The program is a terminal program written in C that takes a command from the user and combines pipe, input redirection, and output redirection. The output of one command before a pipe acts as input for the other command after the pipe, and the output of one command before or after a redirection operator acts as input for another command before or after according to redirection operator. The program uses pipe(), dup2() and execvp() system calls to implement these features.

To handle pipe, I parse the command according to '|' symbol. The number of commands was calculated using the command_count variable. A loop was used to create pipes for communication between commands. The pipe() function was used to create pipes, and the pipes array was used to store the file descriptors for the pipes. In pipes array, each pipe has its descriptors. For example, 0th and 1st index of pipes array is relevant with first pipe, 2nd and 3th index of pipes array is relevant with second pipe and continues like that.

Then, in loop, to handle input and output redirection, I redirect the input and output of the child process as specified by the command. If the '<' symbol was found in the command line, the input was redirected from the specified file. This was done by opening the file using the open() system call with the O_RDONLY flag, then using the dup2() system call to redirect the input of the child process to the opened file descriptor. If the '>' symbol was found in the command line, the output was redirected to the specified file. This was done by opening the file using the open() system call with the O_WRONLY | O_CREAT | O_TRUNC flags, then using the dup2() system call to redirect the output of the child process to the opened file descriptor. If '<' symbol and '>' symbol can not be found in command line, this means that there is a pipe between two commands and dup2() function was used to redirect the output of the previous command to the input of the current command. After dup2(), unused pipe file descriptors are closing.

Then, the child processes in loop was logged a file about its pids and its commands. File name is given according to current timestamp.

Then, I execute the command using execvp() system call. In the parent process, the program closes any unused pipe file descriptors and waits for all child processes to terminate with wait() function.

Also, I handled some signals like SIGINT, SIGTERM, SIGTSTP and SIGQUIT and SIGKILL. When this signal is received, program prints signal message. After printing signal message, except SIGKILL signal, program continues to take command from user. Finally, if user wants to finishing the shell program, user should enter “:q” to the terminal.

Which Requirements I Met?

I completed all the parts.

The program was able to successfully handle input and output redirection as well as pipes between multiple commands.

If user run the program in a wrong way, program prints usage information.

If user enters wrong command, program gives a error message.

Also, I create a log file with a name current timestamp and this file includes all pids of child processes with their corresponding commands.

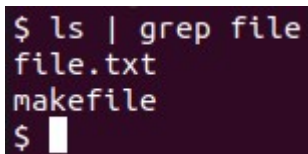
It handles signals successfully and when SIGKILL signal is given to the program, program is terminating. Otherwise, program continues to take a command from user.

Clean up after child processes is done. Also parent process waits for all child processes to terminate with wait() function and this event prevents zombie process.

Program does not give memory leak.

Test Cases

1- Command has only pipe



```
$ ls | grep file
file.txt
makefile
$
```

In order to testing this case, I used ls | grep file command.

This will list all files in the current directory and then filter out the output to show only files that contain the string “file”.

2- Command has only '<'

```
$ cat file.txt
mehmet
furkan
mustafa
ahmet
emir
$ sort < file.txt
ahmet
emir
furkan
mehmet
mustafa
$
```

sort < file.txt command sort the contents of the “file.txt” file and output the results to the console.

In order to testing this case, I used sort < file.txt command. But before applying this command , I used cat file.txt command. Because we need to know file.txt content before printing sorted version of this file.

3- Command has only '>'

```
$ cat file.txt
mehmet
furkan
mustafa
ahmet
emir
$ cat file.txt > file2.txt
$ cat file2.txt
mehmet
furkan
mustafa
ahmet
emir
$
```

cat file.txt > file2.txt command write the content of “file.txt” to “file2.txt”.

In order to testing this case, I used cat file.txt > file2.txt command. But before applying this command , I used cat file.txt command. Because we need to know file.txt content before writing file.txt to the file2.txt. Then, in order to check that “file.txt” is written to “file2.txt” successfully, I used cat file2.txt command.

4- Command has combination of pipe and redirection

```
$ cat file.txt
mehmet
furkan
mustafa
ahmet
emir
$ cat file.txt | sort > file3.txt
$ cat file3.txt
ahmet
emir
furkan
mehmet
mustafa
$
```

cat file.txt | sort > file3.txt command sort the contents of the “file.txt” file and then, write the content of “file.txt” to “file3.txt”.

In order to testing this case, I used cat file.txt | sort > file3.txt command. But before applying this command , I used cat file.txt command. Because we need to know “file.txt” content before sorting this file and writing to the another file. Then, in order to check that sorted version of “file.txt” is written to the “file3.txt” successfully, I used cat file3.txt command.

5- Signal Handling

```
$ ^C
Received SIGINT signal
$ ^Z
Received SIGTSTP signal
$ ^\
Received SIGQUIT signal
$
```

When program receives SIGINT , SIGTSTP or SIGQUIT signal, program prints received signal name and continues to take command from user.

```
Received SIGTERM signal
$
```

When I send the SIGTERM signal to the program from another terminal, program prints signal name and continues to take command from user

```
$ Killed
```

When I send the SIGKILL signal to the program from another terminal, program terminates.

6- Entering Wrong Command

```
$ mehmet
exec error: No such file or directory
Please enter true command
: Invalid argument
$
```

7- Entering “:q” for exiting from the program

```
$ :q
mehmet@mehmet-ubuntu:~/Desktop/system-prog-2023-hw/hw2
```

8- Controlling Memory Leak with Valgrind

```
==38803==
==38803== HEAP SUMMARY:
==38803==    in use at exit: 0 bytes in 0 blocks
==38803==   total heap usage: 2 allocs, 2 frees, 2,048 bytes allocated
==38803==
==38803== All heap blocks were freed -- no leaks are possible
==38803==
==38803== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

As you see from this figure, program does not give memory leak error.

Also, in my code, in order to show that SIGKILL signal is uncatchable, I have signal handler about SIGKILL signal and when you run the code with valgrind, you see that SIGKILL signal is uncatchable.

```
==38511== Warning: ignored attempt to set SIGKILL handler in sigaction();
==38511==    the SIGKILL signal is uncatchable
$
```

9- When user run the program in a wrong way

```
mehmet@mehmet-ubuntu:~/Desktop/system-prog-2023-hw/hw2$ ./hw2 mehmet
Usage like this: ./hw2
: Success
```