Gebze Technical Univesity

CSE 344 System Programming

-

Homework #2 Report

Abdulsamed Aslan

200104004098

Problem overview

I was expected to develop a terminal emulator capable of handling up to 20 shell commands by using **`fork(),`** `execl(),` `wait()` and `exit()` system calls. We are not allowed to use system(). Basically, terminal emulator have a parent process and child processes and Child processes represent a single command.

Terminal emulator creates required pipes and enters a loop to create child processes utilize fork system call. Each child handle the redirections to use pipes "|" properly. Then Child process call the `execv()` system call to execute given command. Furthermore, pids, command and exit status are logged in a separate file that corresponding to the current timestamp.

Problem solving approach

Getting commands:

First of all, Program create sigaction struct to handle SIGINT and SIGTSTP signalS as the program must wait for ":q" to finalize. Then it enters to infinite loop and gets the command from user. Otherwise, the function removes any blank spaces from the input using remove_blanks(), and parses the input for pipe characters using parse_pipes() to break it into separate commands.

Redirection signs:

If the input contains only one command and has < or > character, the handle_redirection_fork() function is called with the command and the redirection character < or >. I use this method because if the given command doesn't have any pipe it should create childs. In handle_redirection_fork function, I create child to execute given process.

```
remove_blanks(str);
numOfCommand = parse_pipes(str, commands);

if (numOfCommand == 1 && strchr(commands[0], '<') != NULL)
      handle_redirection_fork(commands[0], '<');

else if(numOfCommand == 1 && strchr(commands[0], '>') != NULL)
       handle_redirection_fork(commands[0], '>');

else
       terminal_emulator(commands, numOfCommand);
```

Pipes:

The terminal emulator creates as many as pipes as the number of given pipes. After that, it enters the for loop that runs as many as the number of commands because each command is represented by a child process.

Child Processes:

In the for loop, parent process calls fork() system call to create a new child. If number of commands is bigger than 1, unnecessary file descriptors that are connected to the pipes is closed. It is important to ensure that the file descriptors are being correctly handled and that they are being closed when they are no longer needed. If file descriptors are not properly closed, it can lead to resource leaks, which can cause issues with system performance or even lead to crashes.Also the standard output or standard input file descriptors are connected to the required pipe's read or write end.

```
close(fd[i-1][1]); // Close the read end of pipe i-1
close(fd[i][0]); // Close the write end of pipe i
dup2(fd[i][1], STDOUT_FILENO); // Redirect stdout to the write end of pipe i
dup2(fd[i-1][0], STDIN_FILENO);// Redirect stdout to the write end of pipe i-1
```

Redirection in child processes:

If the command contains the '<' or '>' symbols, the handle_redirection() function is called with the current command. Because I handled "<" and ">" separate from pipes.

```
if (strchr(commands[i], '<') != NULL )
     handle_redirection(commands[i], '<');


else if(strchr(commands[i], '>') != NULL)
      handle_redirection(commands[i], '>');
```

execvp:

If the command does not contain either symbol, the code splits the command into arguments using the getArguments() function and stores them in an array called words. It then calls the execvp() function to execute the command, passing in the first element of words as the command to execute and words as the argument list.

```
else{
      free_words = getArguments(commands[i], words, 10);
      execvp(words[0], words);
      perror("execvp");
      fclose(logFile);
      for (int i = 0; i <= free_words; i++)
           free(words[i]);
exit(EXIT_FAILURE);
}           }
```

Handle Redirection:

The handle_redirection() function handles input and output redirection for a given command. It takes a command string and a character that specifies the direction of redirection ('<' for input redirection or '>' for output redirection). It uses parse_redirection() and getArguments() to separate the command and its arguments and the filename to redirect to, and then uses open(), dup2(), and close() to set up the redirection. Finally, it executes the command with execvp() and returns true. Unlike handle_redirection_fork(), handle_redirection() doesn't create any child.

If the commands don’t consist any redirection sign, the program calls execvp with arguments.The "execvp" function is then used to execute the shell command. If the execution fails, the program prints an error message using the "perror" function and exits with a failure status code.

Parent Process:

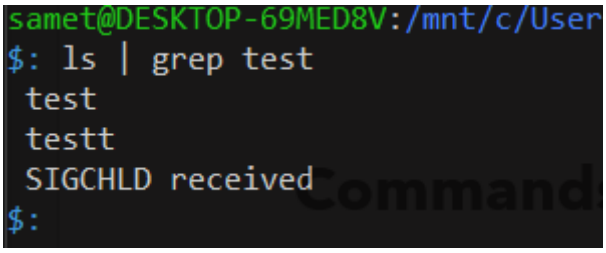
Parent process sets up a signal handler using the "sigaction" function to handle the signals SIGTERM and SIGCHLD. It then closes all the pipe file descriptors in the parent process to ensure that they are no longer being used.

Next, Parent process waits for each child process to finish using the "wait" function. If the wait is interrupted by a signal, it checks whether the signal is SIGCHLD, indicating that a child process has finished. If a child process has finished, parent process prints information about its execution status to a log file using the "fprintf" function. This includes the process ID, command, and either the exit status or the signal that caused the termination.

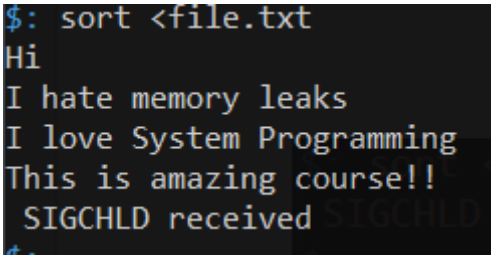
Error messages are signals that occur during execution are printed to the screen and it program return to the prompt receive new commands.

Commands I Tested

`LS | GREP TEST`



`Sort < file.txt`



`cat file1.txt > file2.txt`

```
$: cat file.txt > file2.txt
 SIGCHLD received
$:
```

file2.txt

```
1   Hi
2   This is amazing course!!
3   I love System Programming
4   I hate memory leaks
```

`Ls | grep file > file3.txt`

```
$: ls | grep file > file3.txt
 SIGCHLD received
 SIGCHLD received
$:
```

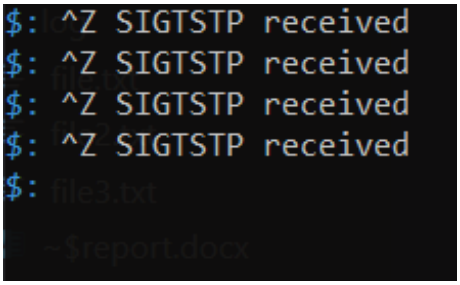
file3.txt

```
1   file.txt
2   file2.txt
3   file3.txt
4   makefile
5
```

`SIGINT (CONTROL-C) and SIGCHLD HANDLING`

```
samet@DESKTOP-69MED8V:/mnt/c/Users/90507/Des
$: sleep 60
^C SIGINT received. Use ':q' for exiting.
$:  SIGCHLD received
$:
```

`SIGSTP (CONTROL-Z)`



Compiling and Running

Compile the code

`make`

Run the code

`./hw2`

Run the code with valgrind

`make valgrind`

clean the unnecessary files

`make clean`