Simple Shell

Name : Hamza Hassan Mohammed

Id : 26

Index

1- Description

- The Shell provides an interface to take the input and execute the programs based on that input by either providing an internal implementation to the commands or calling executables existing in the bin folder.

- The shell is implemented in C and making system to calls using “execvp” command.

2- Basic idea

- The shell simply consists of an infinite loop which could be broken by the user entering ‘exit’ command.
- Each iteration of the loop consists mainly of 5 steps
    - Taking input from the user
    - Splitting the input as needed
    - Forking the parent process
    - Calling “execvp” in the child process to execute the entered command.
    - Wait for the child process to terminate (unless it is supposed to run in the background) .

3- Main Functions

- Handling the parent process properly

```
if (!background) {
    fprintf(debug, "Waiting for the child process to end \n");
    int status;
    fprintf(debug , "\n**We are in parent now with pid = %d and It is goint to wait for the child **\n", getpid());
    while (wait(&status) != child_pid) {
    }
} else {
    fprintf(debug , "\n** We are in the parent now with pid = %d but we aren't going to wait for background child \n",
           getpid());
    fprintf(debug, "Parent will not wait deliberately for the child  \n");
}
```

When we fork the parent process, the child process is created and executed parallelly with the parent process.
Waiting for the child process to end (using "waitpid" function) is necessary so that when the child process terminates, a signal is sent to the parent to clear his child's memory and contents.
An exception here is when the child process is supposed to be a background process so in this case, we have to make the parent continue its execution and not to wait until the child end.

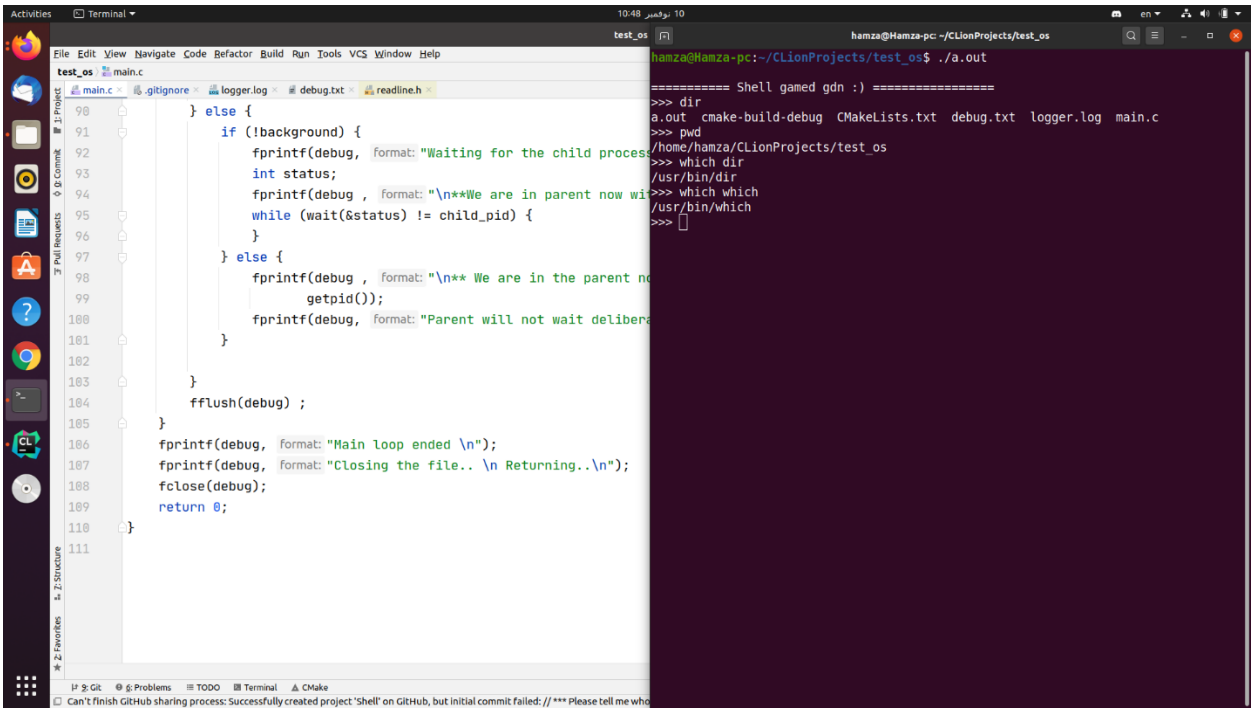
- Handler

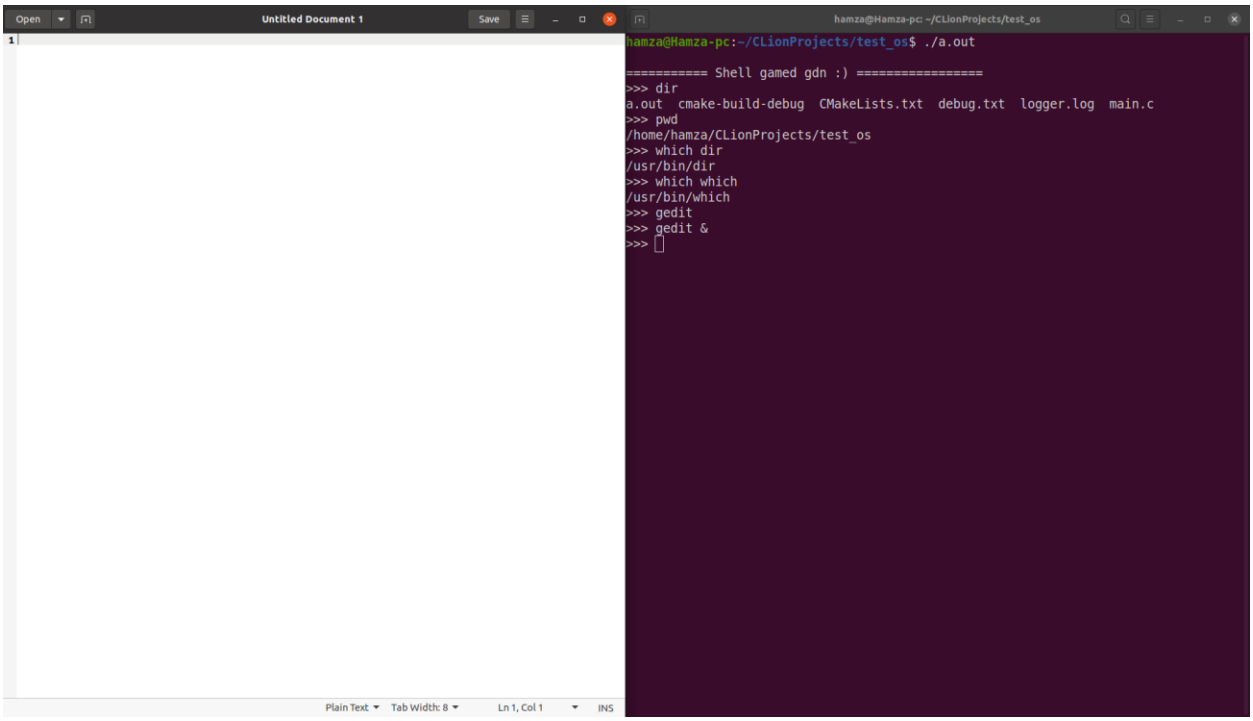
```
void handler() {
    int status;
    wait(&status); // clean the child :)
    fprintf(logger, "Child process in terminated with status  = %d  \n ", status);
    fflush(logger) ;
//    printf("Child process in terminated with status  = %d  \n ", status);
    //signal(SIGINT , handler) ;
}
```

Before even forking a process we define a signal as shown below.
signal(SIGCHLD, handler);

this signal is sent to the parent and calls the handler function whenever a process in terminated so that we avoid the Zombie processes to accumulating in the process table.

4- Sample runs and shots





systemd	hamza	Slee	20.6 MiB	0	Today 7:1	1044	1.9 MiB	493.8 MiB	4.8 MiB	N/A
gnome-terminal-serve	hamza	Slee	939.6 MiB	0	Today 10:	10798	12.5 MiB	24.9 MiB	N/A	N/A
bash	hamza	Slee	10.6 MiB	0	Today 10:	10809	1.5 MiB	512.2 MiB	281.4 MiB	N/A
a.out	hamza	Slee	2.4 MiB	0	Today 10:	14906	104.0 KiB	13.1 MiB	52.0 KiB	N/A
gedit	hamza	Slee	801.4 MiB	0	Today 10:	15019	18.8 MiB	N/A	N/A	N/A