**A picture containing text, sign

Description automatically generated**

**Operating Systems – CSE x61**

**Simple Shell**

By students

|  |
| --- |
| **Ali Mohamed Taima Hummus** |
| **AbdelRahman Ahmed Hassan AlSindiony** |
| **Mohamed Eid Mohamed Ali** |

**Alexandria University**

**Faculty of Engineering**

**Spring 2022**

# Program Overview

The program is written In C programming Language for use on UNIX/Linux operating systems (OSs) as a simple shell. It’s implemented by splitting the algorithm into multiple functions as a procedural programming method, following is the algorithm, code description, results for sample runs, and processes hierarchy.

# Program Algorithm

1- Get input from the user and store it as string (buf).

2- Check (buf) for the special cases, empty input, exit, or help, and handle their occurrences.

3- Convert the user string (buf) to an array of strings (command) with the command at the first index and the parameters follow it.

4- Check (command) for cd command and handle its occurrence.

5- Fork a new child process to handle the user required process and report the log file.

6- Pass the (command) array to execvp function to proceed.

7- Wait the process until finishing its job and terminate it only if & doesn’t exist at the end of the command, otherwise, skip waiting this process.

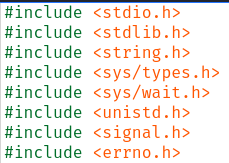
8- Handle (SIGCHLD) signal, recording the termination of a process to the log file.

9- In steps 7, 8, keep record of zombie processes, those who have been skipped without waiting and were terminated later.

10- Bury Zombie processes -if they existed- before prompting for a new input.

# Code Description

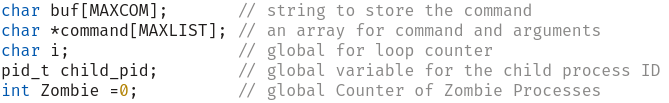
* As usual, the C code starts with the libraries needed



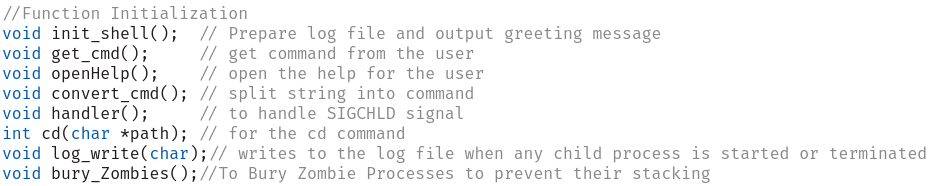
* Next, we define the global constants used in the program



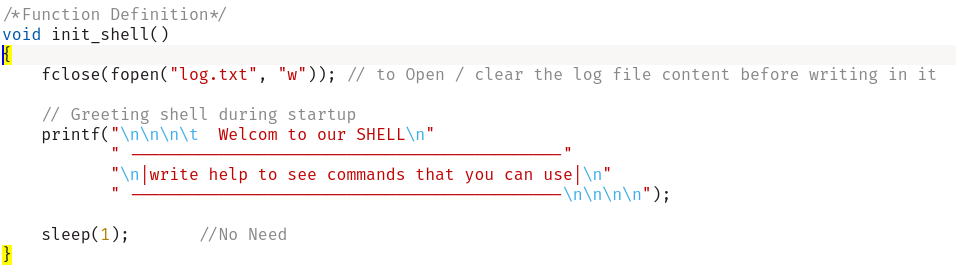
* Also, we define the global variables



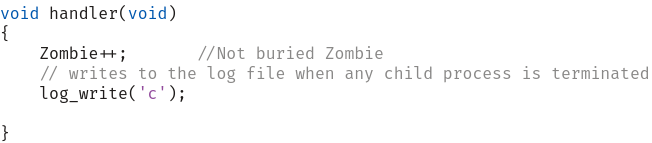
* Then, we initialize the functions that will be used later



* The main function calls another function called init\_shell which creates or clears the log file before writing on it to avoid appending on garbage data, it also prints a well-organized message to welcome the user.



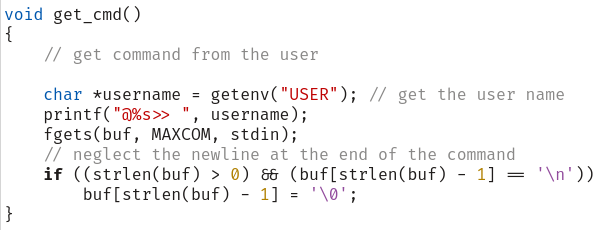
* Then the main function prepares for receiving SIGCHLD signal and calls a function called handler which in turn calls another function called log\_write to write into the log file that a process was terminated.



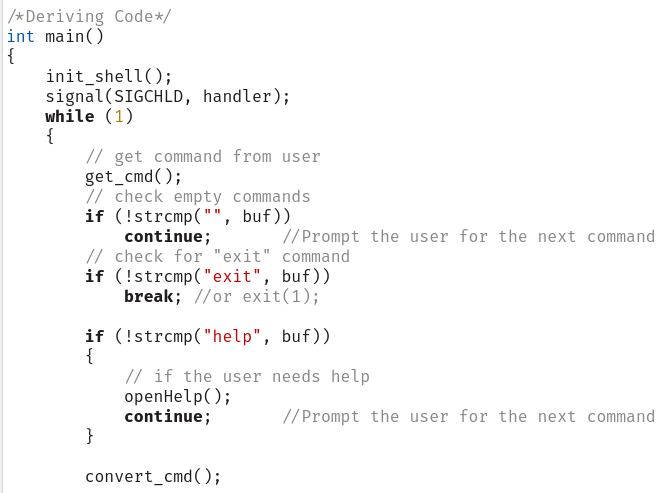


Note: each time a process terminates, we increment the zombie counter, in contrast, we decrement it after each *wait* call, so, the variable Zombie represents the number of available zombie processes

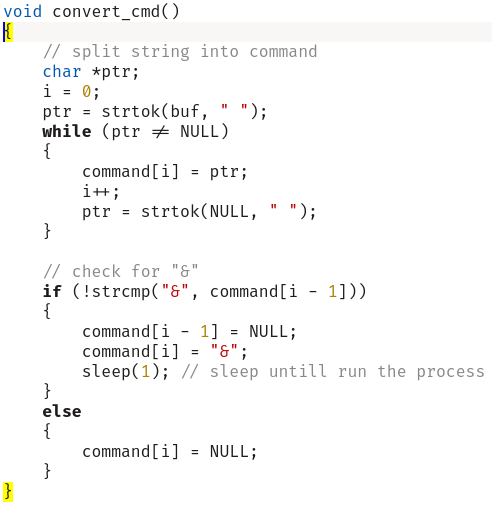
* The main function calls get\_cmd function which takes string input from the user, by taking data of size MAXCOM from stdin file to buf array, also the function prints the user environment variable before each prompt.



* Three conditional statements now appear to implement step 2 in the algorithm for checking empty input, exit, or help. The latter calls a function to print the available commands to the user.



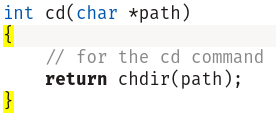
* Now, another function *convert\_cmd* call appears to convert the input stored at variable buf to the desired form of command to be passed to *execvp* function later, which is step 3 at the algorithm.



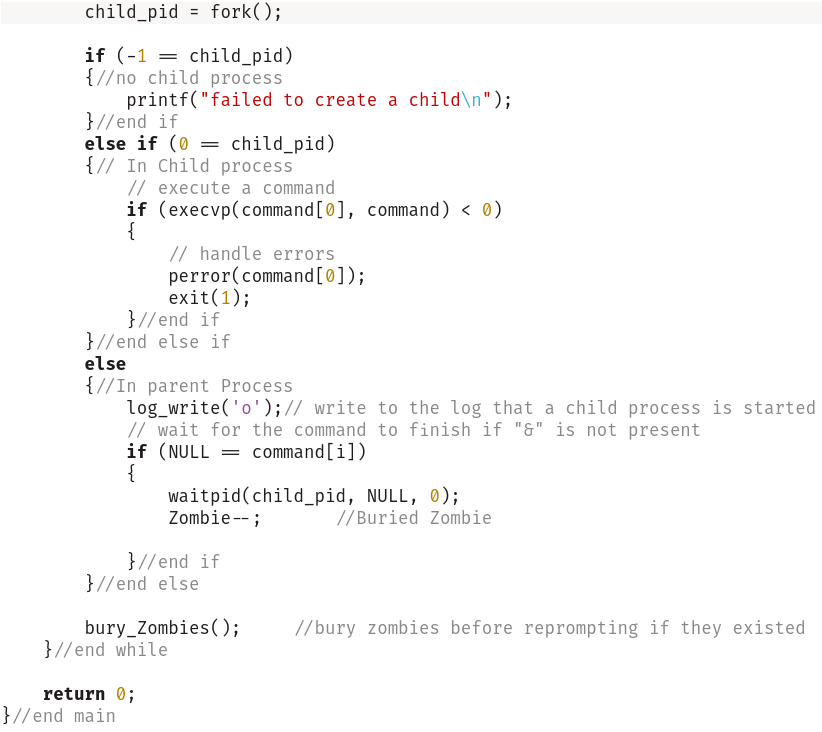
* Now, we implement step 5 at the algorithm, checking the special condition for *cd* command and handling it using a user defined function called *cd* with the path as the parameter at index 1 of the command array.



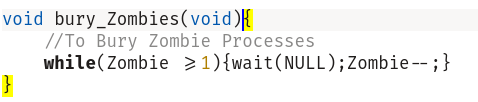
*cd* is defined as:



* Implementing step 5, we fork a new child process



* Forking a new process we face three conditions, First, no process initialized, where the program prints an error to the user with that fact. Second, the child process is running, where the program passes the commands to be executed to the system call function *execvp*, finally the program issues *exit* system call to terminate the process after its execution. Third, the parent process is running, where it calls the function *log\_write* to write that a process has started, then it waits for the child process to terminate if and only if the & character didn’t exist, otherwise it is skipped letting the process run at the background.
* Finally, the program buries any zombie process before prompting again using the function call *bury\_Zombies* , following is its defining code



with the variable *Zombie* equals the number of backgrounds terminated processes.

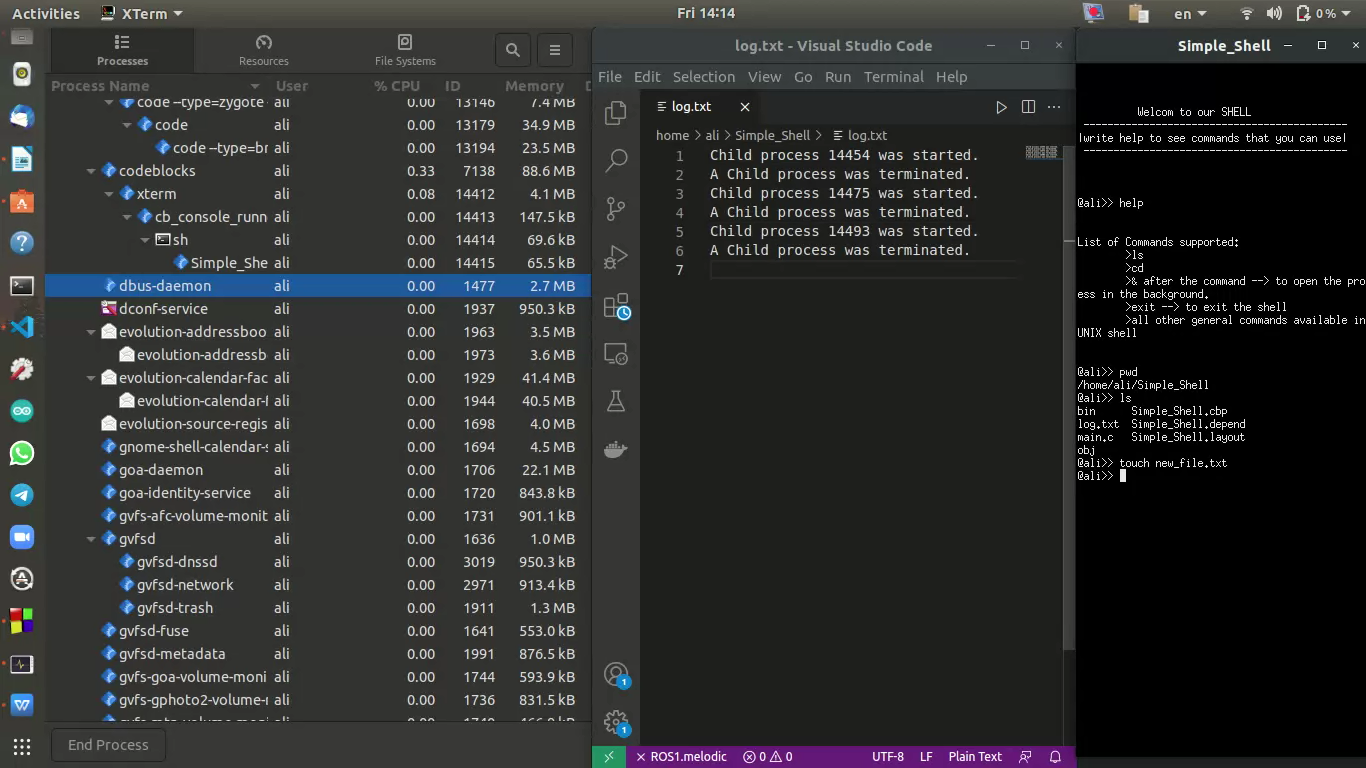
# Results and Sample Runs

>>> help, then >>>pwd

Graphical user interface, text

Description automatically generated

>>> ls



>>> touch new\_file.txt

Graphical user interface, text

Description automatically generated

>>> rm new\_file.txt

Graphical user interface, text

Description automatically generated

>>> firefox

Graphical user interface, text

Description automatically generated

>>>ls -l

Firefox was closed but still zombie processes Graphical user interface, text

Description automatically generated

Zombie processes fully terminated after executing any command, i.e, >>>pwd Graphical user interface, text

Description automatically generated

Graphical user interface, text

Description automatically generated

>>> exit