Alexandria University
Faculty Of Engineering
Computer & Systems Engineering Department

جامعة الاسكندرية
كلية الهندسة

- **OS lab #1 report**
- **The report includes the following:**
  - A description of the overall organization of your code and the major functions.
  - Sample runs.
  - screenshots for the processes hierarchy in KSysguard (or any similar package) during the execution of your shell program.

- **Student info:**

Adel Mahmoud Mohamed Abdelrahman	20010769

Description of the overal organization of my code:

1. The code is written in C programming language.
2. It's well organised-just like the pseudo code-and heavily commented.
3. The code is broken into function each with its own job and all the functions work along together to serve the **myShell()** function which include the main loop of the shell.
4. The functions are charcterized into the following:
    - Functions to handle replacing the '$' in the expression with the value of the environment variables(if any).
    - Function to extract the args[] array which will have the command ready for for execute_command().
    - Function to handle the built in commands.
    - Function is called on the child termination to reap the zombie process and to log the termination to a text file called "terminationLog.txt".
    - Function to set up the environment.

Now let's see the major functions and their details in my code.

Major functions in my code:

1. The included libraries and global variables:
    - The maximum length of the command input is assumed to be 1024.
    - The maximum length of the arguments array is assumed to be 64.

```c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <signal.h>
#include <sys/wait.h>
#include <sys/resource.h>
#include <string.h>
#include <stdbool.h>
#define MAX_COMMAND_LENGTH 1024
#define MAX_NUM_ARGS 64

// declare the pointer to the txt file of the logs
FILE *termination_logs_file;
```

2. Function to remove the double quotes of a string by moving the ptr to the first char one step ahead and termite the string at the last quote it returns a string without double quotes.

```c
char *remove_double_quotes(char *str)
{

// if we have a double ended quotes string then we have to get rid of t
hese quotes
    // otherwise return the string as it is
    if (str[0] == '"' && str[strlen(str) - 1] == '"')
    {
        str[strlen(str) - 1] = '\0';
        return str + 1;
    }
    return str;
}
```

3. Function to set up the command input by removing the leading and trailing spaces form it.

```c
char *set_up_command(char *command)
{

// function to remove the leading and trailing spaces of a command strin
g(input);
    while (command[0] == ' ')
    {
        command = command + 1;
    }
    while (command[strlen(command) - 1] == ' ')
    {
        command[strlen(command) - 1] = '\0';
    };
    return command;
}
```

4. Function is called at export command and its description is as commented

```c
void export(char str[])
{
    // this function is called when the export command is entered

// it adds the new variable with its associated value to the environment variables

// uses strsep() to seperate the string at the '=' sign and the variable is returned

// then str is now pointing to whatever after '=' so i've used strdup()
    // to make a duplicate of what's remaining (value of variable).
    char *variable;
    char *value;
    variable = strsep(&str, "=");
    value = strdup(str);
    setenv(variable, remove_double_quotes(value), 1);
}
```

5. Function to evaluate a the command string at ‘$’ so the variable after it is replaced with its exported value if found in the environment variables and if not found then we get rid of both the ‘$’ sign and the variable following it
    - **Note**: the variable length is begging at the index following the ‘$’ index till the nearest space, another ‘$’ sign or the end of the string just like the terminal of linux way of evaluation.

```c
char *replace_env(char *str, int found_at)
{

// this function takes on the command string and then sees what's after '$' and

// this is assumed to be a variable stored in the environment variables

// so if the variable is stored then it's value is plugged in the string
    // otherwise the "$varaible" are removed form the string

    char temp[50]; //to hold the variable found after '$' in t str
    char result[1024];
// to hold the string after we evaluated the '$' if found in env

    int count = 0;

    char *val;
    bool variable_found_in_environment = false;
    for (int i = found_at + 1; i < strlen(str); i++)
    {
        if (str[i] != '$' && str[i] != ' ' && str[i] != '"')
            temp[count++] = str[i];
        else
            break;
    }
    temp[count++] = '\0';

    if (getenv(temp) != NULL)
    {
        variable_found_in_environment = true;
        val = getenv(temp);
    }

    int it = 0, i = 0;
    while (i < strlen(str))
    {
        if (i < found_at)
        {
            result[it++] = str[i++];
        }
        else if (i == found_at)
        {
            if (variable_found_in_environment)
            {
                for (int j = 0; j < strlen(val); j++)
                {
                    result[it++] = val[j];
                }
            }
            i += strlen(temp) + 1;
        }
        else
        {
            result[it++] = str[i++];
        }
    }
    result[it++] = '\0';
    strcpy(str, result);
    return str;
}
```

6. Recursive function that goes through the command string and if a ‘$’ is detected then the previous function above is called and the recursive calls and repeated until there’s no ‘$’ signs in the command string.

```c
char *send_str_to_replace(char *str)
{

// it's a recursive function that's called whenever there's still '$' in the command

// when '$' is encountered then we call the replace_env() with the index of '$' passed

// recursively, the '$' signs are removed untill the command is '$' free then guess what !
    // that's the base case so we're done
    bool interpolation_found = false;
    int i;
    for (i = 0; i < strlen(str); i++)
    {
        if (str[i] == '$')
        {
            interpolation_found = true;
            break;
        }
    }
    if (interpolation_found)
    {
        char *arr = replace_env(str, i);
        return send_str_to_replace(arr);
    }
    else
        return str;
}
```

7. Function to extract the argos array and this function calls the previous two function i.e the args array is formed when the command is well-defined and all the ‘$’ signs are gone.

```c
void *extracting_args(char command[], char *args[MAX_NUM_ARGS])
{

// this function calls the recursive function above at the beggining to remove all '$'

// and then we have our command ready and clean without these disturbing '$'

// so, we slice our string into array of strings called char *args[]

// based on the first word of the comand {cd, ls, etc..} the slicing of the remaining string is defined
    char *well_defined_command = send_str_to_replace(command);
    strcpy(command, well_defined_command);
    int count = 0;
    args[count++] = strsep(&command, " ");
 // get the key word for the command


// if we have echo or export with double quotes then we need to have all of what's left in one take
    if ((strcmp(args[0], "export") == 0 && strchr(command, '"') && command[strlen(command) - 1] == '"') || strcmp(args[0], "echo") == 0)
    {
        args[count++] = command;
    }

// otherwise we split the string to substrings and the delimeter is the space in the command
    else
    {
        while (command != NULL)
        {
            args[count++] = strsep(&command, " ");
        }
    }
}
```

8. Function to execute the built in commands that are required {cd, export, echo} and in case of cd command a function called cd_handler() is called which handles all the different cases of cd that are required.

```c
void execute_shell_bultin(char *args[])
{

// function that handles type 0 input which is the shell built in commands{export, cd, echo}
    if (strcmp(args[0], "cd") == 0)
    {
        cd_handler(args);
    }

    else if (strcmp(args[0], "echo") == 0)
    {
        // if it's echo then remove the quotes and you're good to go
        // note the '$' is gone, we handled it
        printf("%s\n", remove_double_quotes(args[1]));
    }

    else if (strcmp(args[0], "export") == 0)
    {
        // if the keyword was export then call the export function
        //  to add a new variable to the environment
        export(remove_double_quotes(args[1]));
    }
}
```

9. Function that executes the non-builtin commands and that's done by forking a new child process and have it call **execvp()** function and the parameters for the command are passed and we'll have the parent process wait by calling **waitpid()** until the child terminates except when a background command is encountered and that's done by passing a boolean to this function indicating whether the command is a background one or not, here we'll pass `**WNOHANG**` option to **waitpid()** to prevent the parent from being blocked.

```c
void execute_command(char *args[], bool background_encountered)
{
    // function to execute non builtin commands

// by forking a new child prcess which executes the execvp() function

    int status;
         // int variable to indicate the exit status of child process
    pid_t pid = fork(); // create a child process

    if (pid == 0)
    {
        // child thread of execution is here
        execvp(args[0], args);

// if you reach here then there's an error in the execvp() so we'll pr
int out an error message
        printf("Error: command not found\n");
        exit(1);
    }
    else if (pid > 0)
    {
        // parent thread of execution is here

// we'll condition now on whether the command is execued in the backgr
ound or not

// if it is then we'll send the option WHOHANG to the waitpid() as a p
arameter
        // to prevent parent blocking
        if (background_encountered)
            waitpid(pid, &status, WNOHANG);

        // other wise the parent waits for his child of course!
        else
            waitpid(pid, &status, 0);
    }

    else
    {
        // fork failed
        perror("failed to fork a new child");
        exit(1);
    }
}
```

10. Function to set up the environment for the shell and that's done by changing the directory as you wish but i chose the current working one which is the one containing the lab C file.

```c
void setup_environment()
{

// function to change the directory to the current working directory
    char cwd[1000];
    getcwd(cwd, 1000);
    chdir(cwd);
}
```

11. Function is invoked whenever a child terminates and in this function any zombie process is reaped off and then we log the successful termination to a text file names "terminationLogs.txt"

```c
void on_child_exit()
{
    // this is called when the signal on chid exit is recieved
    // the zombie is reaped off if it's there

    int status;
    pid_t pid;

    while (1)
    {
        pid = wait3(&status, WNOHANG, (struct rusage *)NULL);
        if (pid == 0 || pid == -1)
            break;
    }

    // here we got out of the infinte loop
    // so we log that the child's terminated successfully

    termination_logs_file = fopen("Termination_logs.txt", "a+");
    fprintf(termination_logs_file, "Child process was terminated\n");
    fclose(termination_logs_file);
}
```

12. The shell loop that parses the command input entered by the user and defines the command type and then calls the right function for execution

```c
void myShell()
{
    // here's where the shell infinte loop is defined
    // the loop is only broken by the exit command
    bool command_is_not_exit = true;
    do
    {
        int input_type = 0;
                                // variable to indicate the type of command whether it's builtin or not
        bool background_command_encountered = false;
// set true when background command encountered '&'
        char current_working_directory[1024];
        getcwd(current_working_directory, 1024);
        printf("\033[31m");
        printf(":%s >> ", current_working_directory);
// priniting the awsome red current working directory
        printf("\033[0m");
        char command[MAX_COMMAND_LENGTH];

        char *args[MAX_NUM_ARGS];

        for (int i = 0; args[i] != NULL; i++)
        {
            // initializing the args array with NULL
            args[i] = NULL;
        }


// get the next command form the user and store it in command string
        fgets(command, MAX_COMMAND_LENGTH, stdin);

        // terminate the command with the null character
        command[strlen(command) - 1] = '\0';


// delete the leading and the reailing spaces form the command string
        strcpy(command, set_up_command(command));

        if (command[strlen(command) - 1] == '&')
        {
            // if we have a command that's executed in the background
            // then set the boolean
            background_command_encountered = true;
            command[strlen(command) - 1] = '\0';
        }
```

The rest of the function is to be continued in the next page:

```c
        // extracting the args array from the command string
        extracting_args(command, args);

        if (args[0] == NULL)
        {
            // empty command, continue loop
            continue;
        }

        if (strcmp(args[0], "cd") == 0 || strcmp(args[0], "export") == 0 || strcmp(args[0], "echo") == 0)
        {
            input_type = 0;
        }

        else if (strcmp(args[0], "exit") == 0)
        {
            command_is_not_exit = false;
        }

        else
        {
            input_type = 1;
        }

        if (input_type)
        {
            // note that the background boolean is passed as well
            // to make the right waitpid() for the parent process
            execute_command(args, background_command_encountered);
        }

        else
        {
            execute_shell_bultin(args);
        }
    } while (command_is_not_exit);
}

```

13. Last but not least, the main function, here we first use the signal() function to receive a signal and when the first parameter is SIGCHLD this denotes that the signal is on the termination of the child process, and the second parameter is the function to be invoked when the signal is received and

this function is defined above previously, and then the environment is set and the shell loop begins.

```c
int main()
{
    fclose(fopen("Termination_logs.txt", "w"));
    signal(SIGCHLD, on_child_exit);
    setup_environment();
    myShell();
    return 0;
}

```

Sample Runs:

Note:

- The program promt is coloured red
- Echo command is handled just like the linux terminal and all the cases are tried out.

```
adel@Adel: ~/Desktop/Lab1
adel@Adel:~/Desktop/Lab1$ gcc main.c -o main
adel@Adel:~/Desktop/Lab1$ ./main
:/home/adel/Desktop/Lab1 >> ls
main  main.c  tempCodeRunnerFile.c  Termination_logs.txt
:/home/adel/Desktop/Lab1 >> ls -l
total 44
-rwxrwxr-x 1 adel adel 22360 02:17 8  مار main
-rw-rw-r-- 1 adel adel 10942 01:35 8  مار main.c
-rw-rw-r-- 1 adel adel   153 02:13 8  مار tempCodeRunnerFile.c
-rw-rw-r-- 1 adel adel    29 02:18 8  مار Termination_logs.txt
:/home/adel/Desktop/Lab1 >> mkdir test
:/home/adel/Desktop/Lab1 >> ls
main  main.c  tempCodeRunnerFile.c  Termination_logs.txt  test
:/home/adel/Desktop/Lab1 >> ls -a -l -h
total 60K
drwxrwxr-x 4 adel adel 4.0K 02:19 8  مار .
drwxr-xr-x 4 adel adel 4.0K 02:01 8  مار ..
-rwxrwxr-x 1 adel adel  22K 02:17 8  مار main
-rw-rw-r-- 1 adel adel  11K 01:35 8  مار main.c
-rw-rw-r-- 1 adel adel  153 02:13 8  مار tempCodeRunnerFile.c
-rw-rw-r-- 1 adel adel  116 02:19 8  مار Termination_logs.txt
drwxrwxr-x 2 adel adel 4.0K 02:19 8  مار test
drwxrwxr-x 2 adel adel 4.0K 23:14 4  مار .vscode
:/home/adel/Desktop/Lab1 >> exit
adel@Adel:~/Desktop/Lab1$
```

```
adel@Adel:~/Desktop/Lab1$ gcc main.c -o main
adel@Adel:~/Desktop/Lab1$ ./main
:/home/adel/Desktop/Lab1 >> ls
main  main.c  tempCodeRunnerFile.c  Termination_logs.txt
:/home/adel/Desktop/Lab1 >> touch someFile.txt
:/home/adel/Desktop/Lab1 >> ls
main  main.c  someFile.txt  tempCodeRunnerFile.c  Termination_logs.txt
:/home/adel/Desktop/Lab1 >> rm someFile.tst
rm: cannot remove 'someFile.tst': No such file or directory
:/home/adel/Desktop/Lab1 >> rm someFile.txt
:/home/adel/Desktop/Lab1 >> ls
main  main.c  tempCodeRunnerFile.c  Termination_logs.txt
:/home/adel/Desktop/Lab1 >> exit
adel@Adel:~/Desktop/Lab1$
```

adel@Adel: ~/Desktop/Lab1

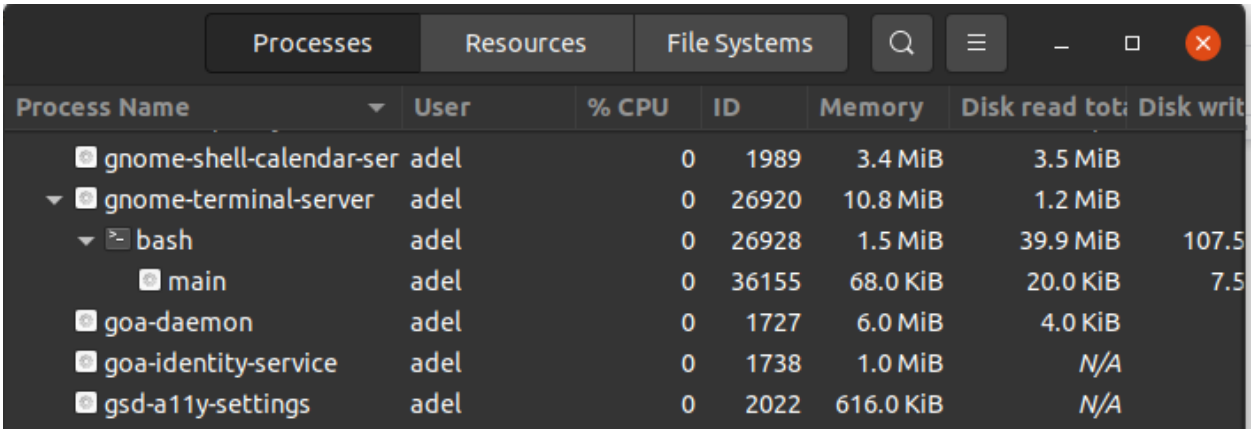
```
adel@Adel:~/Desktop/Lab1$ ./main
:/home/adel/Desktop/Lab1 >> pwd
/home/adel/Desktop/Lab1
:/home/adel/Desktop/Lab1 >> export x="-a -l -h"
:/home/adel/Desktop/Lab1 >> ls $x
total 60K
drwxrwxr-x 4 adel adel 4.0K 02:25 8  مار .
drwxr-xr-x 4 adel adel 4.0K 02:01 8  مار ..
-rwxrwxr-x 1 adel adel  22K 02:25 8  مار main
-rw-rw-r-- 1 adel adel  11K 01:35 8  مار main.c
-rw-rw-r-- 1 adel adel  153 02:13 8  مار tempCodeRunnerFile.c
-rw-rw-r-- 1 adel adel   29 02:26 8  مار Termination_logs.txt
drwxrwxr-x 2 adel adel 4.0K 02:19 8  مار test
drwxrwxr-x 2 adel adel 4.0K 23:14 4  مار .vscode
:/home/adel/Desktop/Lab1 >> export "x=hello"
:/home/adel/Desktop/Lab1 >> export y="world"
:/home/adel/Desktop/Lab1 >> echo "$x$y"
helloworld
:/home/adel/Desktop/Lab1 >> echo "$x$yz"
hello
:/home/adel/Desktop/Lab1 >> echo "$xyz"

:/home/adel/Desktop/Lab1 >> echo "$x$xy$y"
helloworld
:/home/adel/Desktop/Lab1 >> echo "WOW"
WOW
:/home/adel/Desktop/Lab1 >> export x=5
:/home/adel/Desktop/Lab1 >> echo "Hello $x"
Hello 5
```

adel@Adel: ~/Desktop/Lab1

```
adel@Adel:~/Desktop/Lab1$ gcc main.c -o main
adel@Adel:~/Desktop/Lab1$ ./main
:/home/adel/Desktop/Lab1 >> cd
:/home/adel >> cd Desktop
:/home/adel/Desktop >> cd Lab1
:/home/adel/Desktop/Lab1 >> cd ..
:/home/adel/Desktop >> cd ..
:/home/adel >> cd ..
:/home >> cd ..
:/ >> cd ..
:/ >> cd ..
:/ >> cd ~
:/home/adel >> cd ..
:/home >> cd ..
:/ >> cd ~/Desktop
:/home/adel/Desktop >> cd Lab1
:/home/adel/Desktop/Lab1 >> pwd
/home/adel/Desktop/Lab1
:/home/adel/Desktop/Lab1 >> heeyyy
Error: command not found
:/home/adel/Desktop/Lab1 >>
```

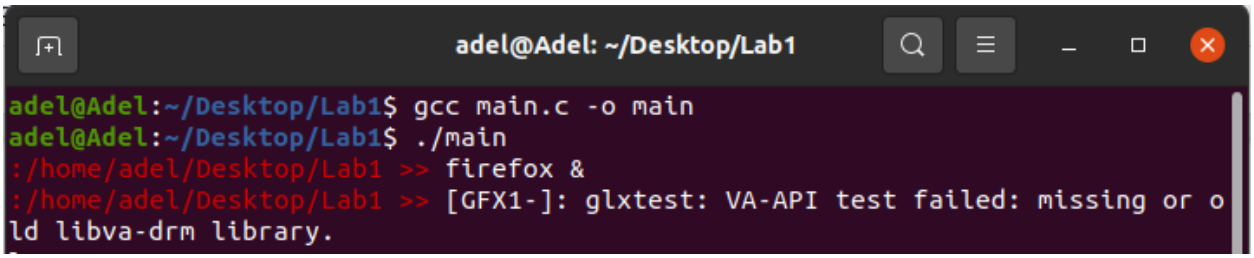
Now let's try out some on-the-background commands and see what's shown in the system monitor provided by gnome.

- First we show the monitor with no app's been run by my simple shell yet.



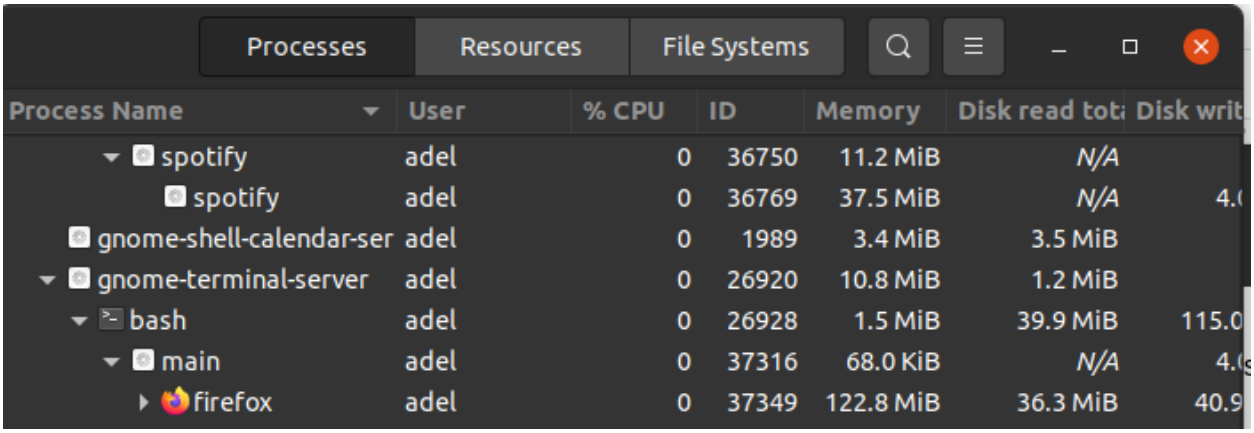
Process Name	User	% CPU	ID	Memory	Disk read tot	Disk writ
gnome-shell-calendar-ser	adel	0	1989	3.4 MiB	3.5 MiB	
gnome-terminal-server	adel	0	26920	10.8 MiB	1.2 MiB	
bash	adel	0	26928	1.5 MiB	39.9 MiB	107.5
main	adel	0	36155	68.0 KiB	20.0 KiB	7.5
goa-daemon	adel	0	1727	6.0 MiB	4.0 KiB	
goa-identity-service	adel	0	1738	1.0 MiB	N/A	
gsd-a11y-settings	adel	0	2022	616.0 KiB	N/A	

- Now, let's have some fun opening firefox by my shell as a background process by.



```
adel@Adel: ~/Desktop/Lab1
adel@Adel:~/Desktop/Lab1$ gcc main.c -o main
adel@Adel:~/Desktop/Lab1$ ./main
:/home/adel/Desktop/Lab1 >> firefox &
:/home/adel/Desktop/Lab1 >> [GFX1-]: glxtest: VA-API test failed: missing or o
ld libva-drm library.
```

- Now, let's see what happened to the dependencies by the monitor.

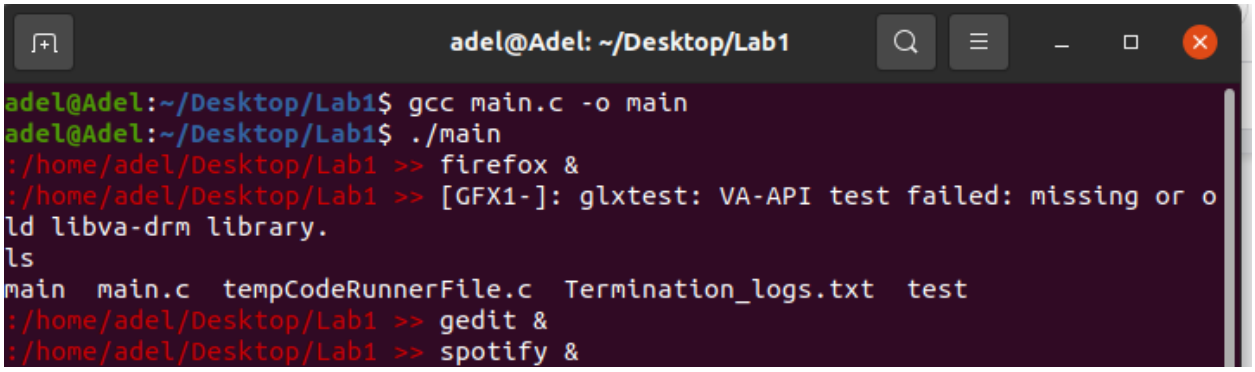


Process Name	User	% CPU	ID	Memory	Disk read tot	Disk writ
spotify	adel	0	36750	11.2 MiB	N/A	
spotify	adel	0	36769	37.5 MiB	N/A	4.(
gnome-shell-calendar-ser	adel	0	1989	3.4 MiB	3.5 MiB	
gnome-terminal-server	adel	0	26920	10.8 MiB	1.2 MiB	
bash	adel	0	26928	1.5 MiB	39.9 MiB	115.0
main	adel	0	37316	68.0 KiB	N/A	4.(
firefox	adel	0	37349	122.8 MiB	36.3 MiB	40.9

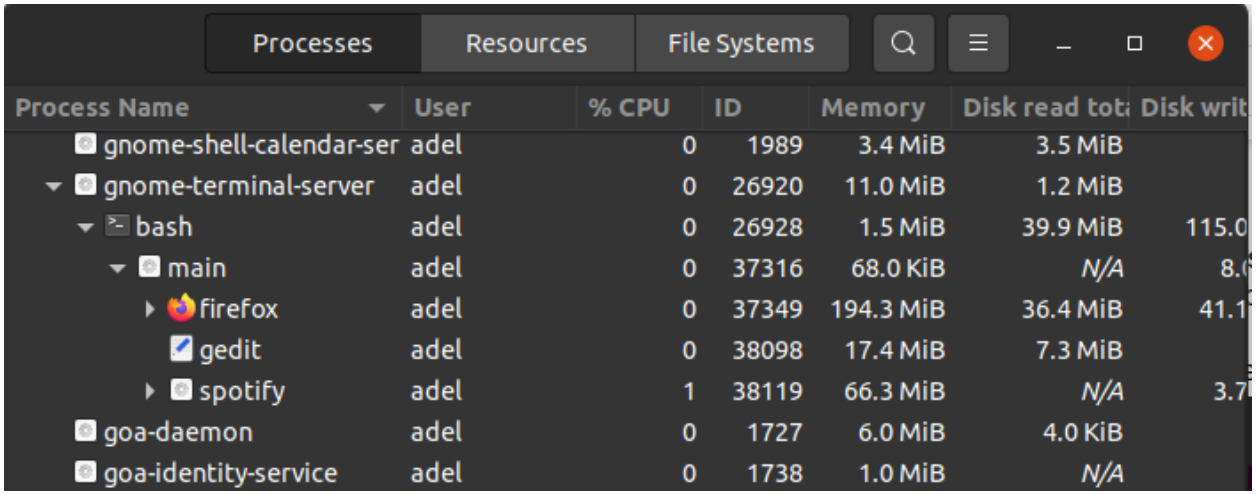
Yes, firefox now's been appended as a child process to the main process which considered the parent because it's the one who's forked firefox.

But because we followed the command by '&' it's run as a background process i.e the parent didn't wait until it terminates because of `**HNOHANG**`, remember !

- Now let's make have much more fun opening another apps by my simple shell !



- Now, we assume all the three apps {firefox, gedit, sportify} to be children of main this time, right ??



Yes, you're right! They are all appended to the main process now .

- One last thing to be shown is the contents of the log file so we'll run some commands from the beginning and we'll see.

```
adel@Adel: ~/Desktop/Lab1
adel@Adel:~/Desktop/Lab1$ gcc main.c -o main
adel@Adel:~/Desktop/Lab1$ ./main
:/home/adel/Desktop/Lab1 >> ls
main  main.c  tempCodeRunnerFile.c  Termination_logs.txt
:/home/adel/Desktop/Lab1 >> pwd
/home/adel/Desktop/Lab1
:/home/adel/Desktop/Lab1 >> ls -l
total 44
-rwxrwxr-x 1 adel adel 22360 03:22 8  مار main
-rw-rw-r-- 1 adel adel 10942 01:35 8  مار main.c
-rw-rw-r-- 1 adel adel   153 02:13 8  مار tempCodeRunnerFile.c
-rw-rw-r-- 1 adel adel    58 03:22 8  مار Termination_logs.txt
:/home/adel/Desktop/Lab1 >> touch heyyy.txt
:/home/adel/Desktop/Lab1 >> ls
heyyy.txt  main  main.c  tempCodeRunnerFile.c  Termination_logs.txt
:/home/adel/Desktop/Lab1 >> ls -l
total 44
-rw-rw-r-- 1 adel adel     0 03:23 8  مار heyyy.txt
-rwxrwxr-x 1 adel adel 22360 03:22 8  مار main
-rw-rw-r-- 1 adel adel 10942 01:35 8  مار main.c
-rw-rw-r-- 1 adel adel   153 02:13 8  مار tempCodeRunnerFile.c
-rw-rw-r-- 1 adel adel   145 03:23 8  مار Termination_logs.txt
:/home/adel/Desktop/Lab1 >> rm heyyy.txt
:/home/adel/Desktop/Lab1 >> ls
main  main.c  tempCodeRunnerFile.c  Termination_logs.txt
:/home/adel/Desktop/Lab1 >>
```

Here i ran 8 simple commands and now let's see what's logged in "terminationLogs.txt" file.

```
Open   *Termination_logs.txt   Save
~/Desktop/Lab1
1 Child process was terminated
2 Child process was terminated
3 Child process was terminated
4 Child process was terminated
5 Child process was terminated
6 Child process was terminated
7 Child process was terminated
8 Child process was terminated
9
```

Yes, exactly eight logs are printed out.
Note: that the logging process is done when the signal on the child termination is received and after the zombie has been reaped off :(

NOTE !!!!

- **The video of the test cases has made the zip file too large to be uploaded to the form.**

- **so, the whole lab contents {video, source code, and the report} are uploaded to Drive in this folder.**

- ***Click here to get to the Drive folder.***