**Report 1**

**Operating Systems Asgn.1**

**Simple Unix shell**

**Ziyad Ahmed Elbanna**

**Id :27**

**Description:**

**The program is written in C language. Its function is to implement the Terminal Bash shell in the C language with all its built-in commands and properties.**

**User only enters the commands to be executed and waits for the process to be done.**

**Read\_parse\_line:**

**This function reads the whole line entered by the user and checks for errors in it, then sends the line to the array of characters which is arg. Array**

**Func. included : read\_line, remove\_endofLine**

**Signal\_handler:**

**This function is the basic function used for signal handler of the child process which is terminated and sends the SIGCHLD signal**

**Log:**

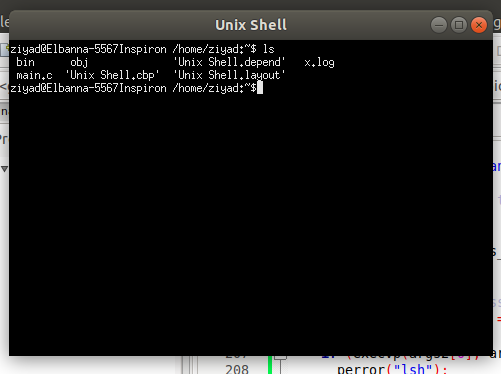
**This function writes the log message to the .log file by passing the message into it as a parameter and the log file writes the child which exited the process.**

**Main:**

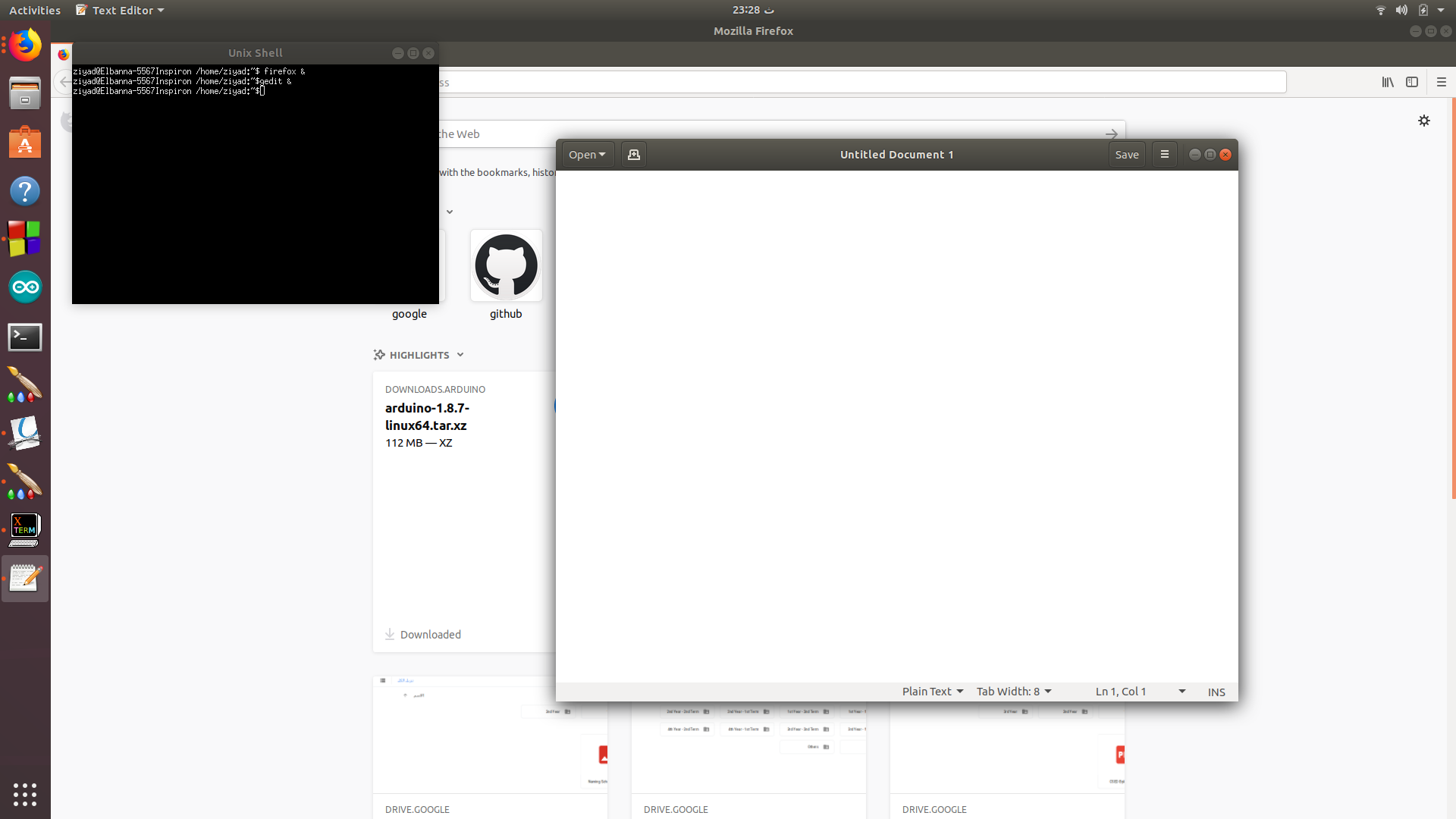
**An infinite loop under the condition of no ‘exit’ command. Which takes the input of the user and executes all their commands**

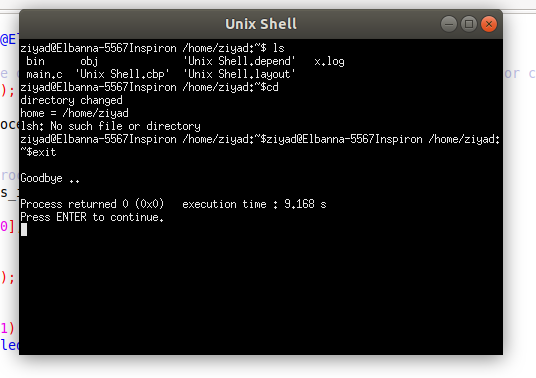
**Sample Runs:**

**1- ls command**

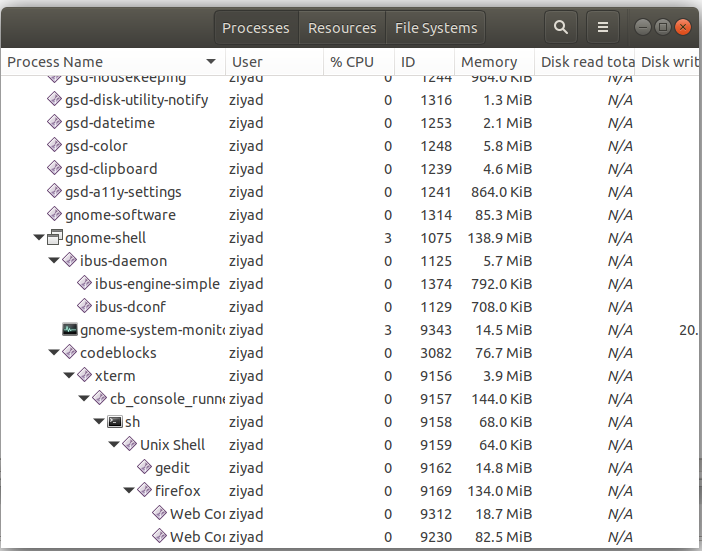


**2- Firefox and Gedit (background processes)**

**3- exit command**



**Process Hierarchy (System Monitor):**



* Firefox and gedit are child processes to Unix shell

**Source Code :-**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <string.h>**

**#include <unistd.h>**

**#include <signal.h>**

**#include <sys/wait.h>**

**#include <sys/types.h>**

**#define MAXLEN 10**

**#define MAXCHAR 100**

**#define DEL " "**

**#include <stdbool.h>**

**extern int bg =0;**

**bool cd =0;**

**int read\_parse\_line(char inputline[],char\* inputargs[],char\* inputargs2[])**

**{**

**readline(inputline);**

**processline(inputline,inputargs,inputargs2);**

**return 1;**

**}**

**void readline(char inputline[])**

**{**

**char\* ret = fgets(inputline,MAXCHAR,stdin);**

**remove\_endofline(inputline);**

**if ((!strcmp(inputline,"exit")) || ret == NULL)**

**{**

**printf("%s", "\nGoodbye ..\n");**

**exit(0);**

**}**

**if ((!strcmp(inputline,"cd")))**

**{**

**cd =1;**

**}**

**else cd =0;**

**}**

**void remove\_endofline(char inputline[])**

**{**

**int i=0;**

**while (inputline[i]!='\n')**

**{**

**i++;**

**}**

**inputline[i]='\0';**

**}**

**int processline(char line[],char\* args[],char\* args2[])**

**{**

**int i =0;**

**int u =1;**

**args[i] = strtok(line,DEL);**

**bg =0;**

**if (args[i] == NULL)**

**{**

**printf("NO COMMAND\n");**

**return 1;**

**}**

**else {**

**args2[i] = args[i];**

**}**

**while (args[i] != NULL)**

**{**

**if (!strcmp(args[i],"&"))**

**{**

**bg =1;**

**}**

**else bg =0;**

**i++;**

**args[i] = strtok(NULL, DEL);**

**}**

**for (u=0;u<i;u++)**

**{**

**if (strcmp(args[u],"&"))**

**{**

**args2[u] = args[u];**

**}**

**else break;**

**}**

**return 1;**

**}**

**void signalhandler (int sig)**

**{**

**pid\_t pid;**

**pid = wait(NULL);**

**char buf [100];**

**//printf("Pid %d exited.\n", pid);**

**int pd = pid;**

**sprintf(buf,"Child Pid %d was terminated.\n",pd);**

**Log (buf);**

**}**

**bool LogCreated = false;**

**void Log (char \*message, int pid)**

**{**

**FILE \*file;**

**if (!LogCreated) {**

**file = fopen("x.log", "w");**

**LogCreated = true;**

**}**

**else**

**file = fopen("x.log", "a");**

**if (file == NULL) {**

**if (LogCreated)**

**LogCreated = false;**

**return;**

**}**

**else**

**{**

**fputs(message, file);**

**fclose(file);**

**}**

**if (file)**

**fclose(file);**

**}**

**int main(int argc, char\*\* argv)**

**{**

**char\* args[MAXLEN]; // the array of all arguments in the shell**

**char\* args2[MAXLEN]; // the array of the arguments without ampersand**

**char line[MAXCHAR]; // entered line in the shell command**

**int status;**

**int result = 0;**

**const char\* home = getenv("HOME");**

**printf("ziyad@Elbanna-5567Inspiron %s:~$ ", home);**

**while (read\_parse\_line(line,args,args2)) // while there is a line to read**

**{**

**if(cd == 1)**

**{**

**if (argc ==1)**

**{**

**home = getenv("HOME");**

**int i = chdir(home);**

**if(i < 0)**

**printf("directory couldn't be changed\n");**

**else{**

**printf("directory changed\n");**

**printf("home = %s\n", home);**

**}**

**}**

**result = chdir(argv[1]);**

**if(result == 0){**

**printf("directory changed\n");**

**}**

**}**

**signal(SIGCHLD, signalhandler); //To handle the signals and see which child exited first**

**if (bg == 1) // if its a background process do the following**

**{**

**printf("ziyad@Elbanna-5567Inspiron %s:~$", home);**

**int background\_process\_id;**

**pid\_t pid = fork();**

**if (pid == 0)**

**{**

**background\_process\_id = getpid();**

**if (execvp(args2[0], args2) == -1) {**

**perror("lsh");**

**}**

**exit(EXIT\_FAILURE);**

**}**

**else if (pid == -1) {**

**printf("\nFailed forking child..");**

**return;**

**}**

**else { // in parent do nothing**

**}**

**}**

**else //bg =0 not a background process**

**{**

**printf("ziyad@Elbanna-5567Inspiron %s:~$", home);**

**// pid is a variable of type pid\_t to determine the process id (whether in parent or child)**

**pid\_t pid = fork();**

**int background\_process\_id;**

**if (pid == 0)**

**{**

**//the child process**

**background\_process\_id = getpid();**

**if (execvp(args2[0], args2) == -1) {**

**perror("lsh");**

**}**

**exit(EXIT\_FAILURE);**

**}**

**else if (pid == -1) {**

**printf("\nFailed forking child..");**

**return;**

**}**

**else**

**{**

**//parent process**

**waitpid(pid,NULL,0);**

**}**

**}**

**//**

**}**

**//**

**return 0;**

**}**