LAB1 | 2023

Simple Shell Multi-Processing

مروان احمد ربيع احمد **19016605**

1 Introduction

It is required to implement a Unix shell program. A shell is simply a program that conveniently allows you to run other programs. Read up on your favorite shell to see what it does.

My shell supports the following commands:

- The internal shell command "exit" which terminates the shell
- A command with no arguments "ls, cp, rm . . . etc"
- A command with arguments "ls –l"
- A command, with or without arguments, executed in the background using & "gedit &"
- Shell builtin commands "cd, echo and export"

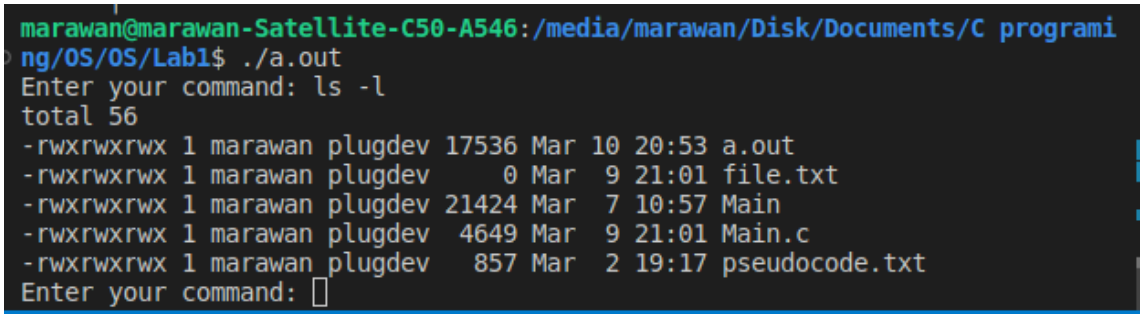

Figure 1: A command with arguments

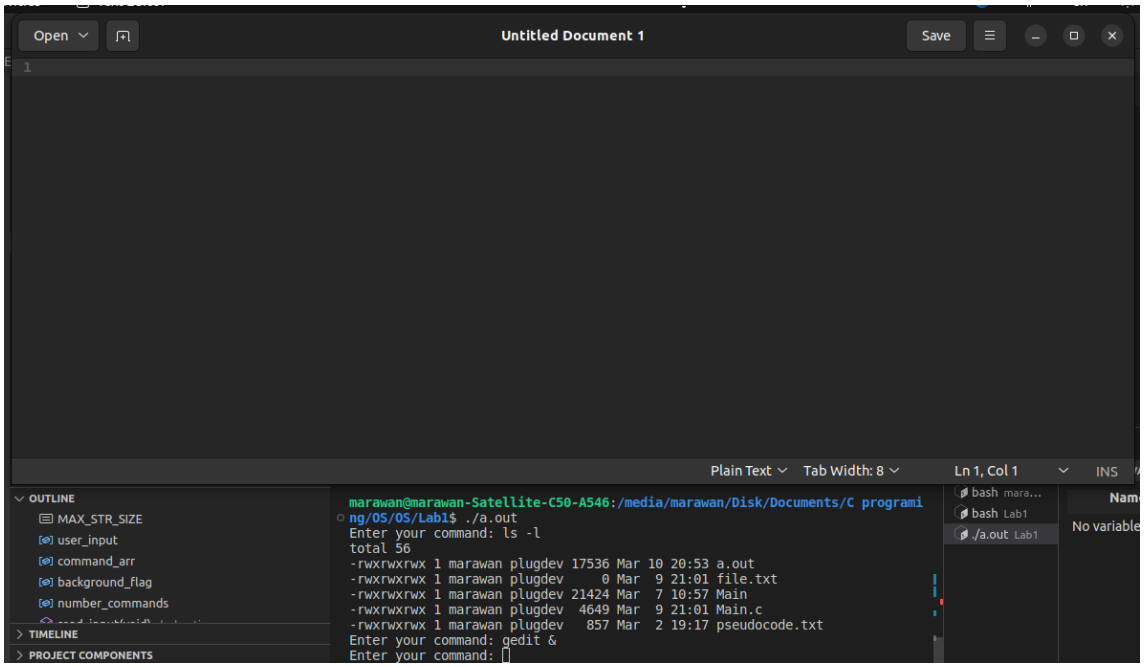

Figure 2: Background process

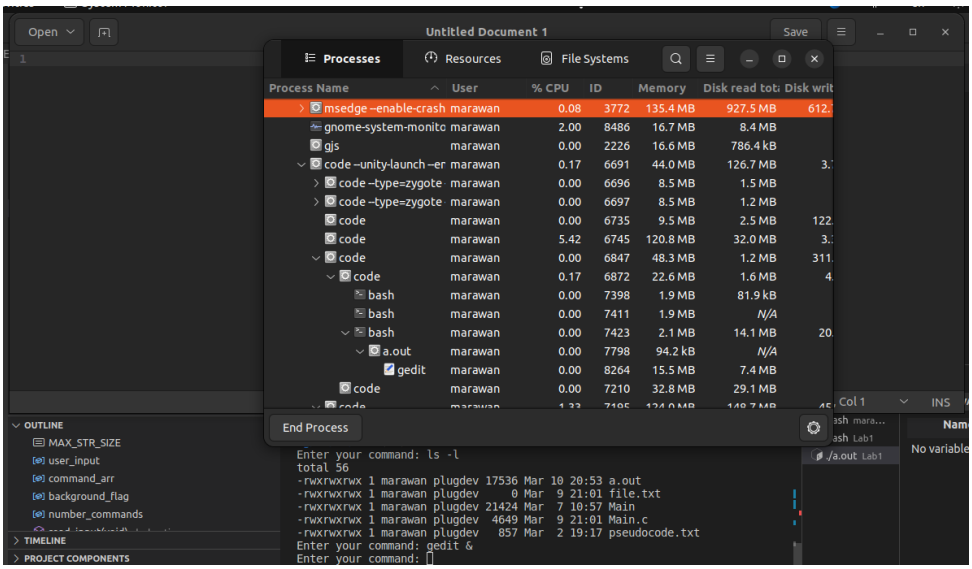

Figure 3: System monitor

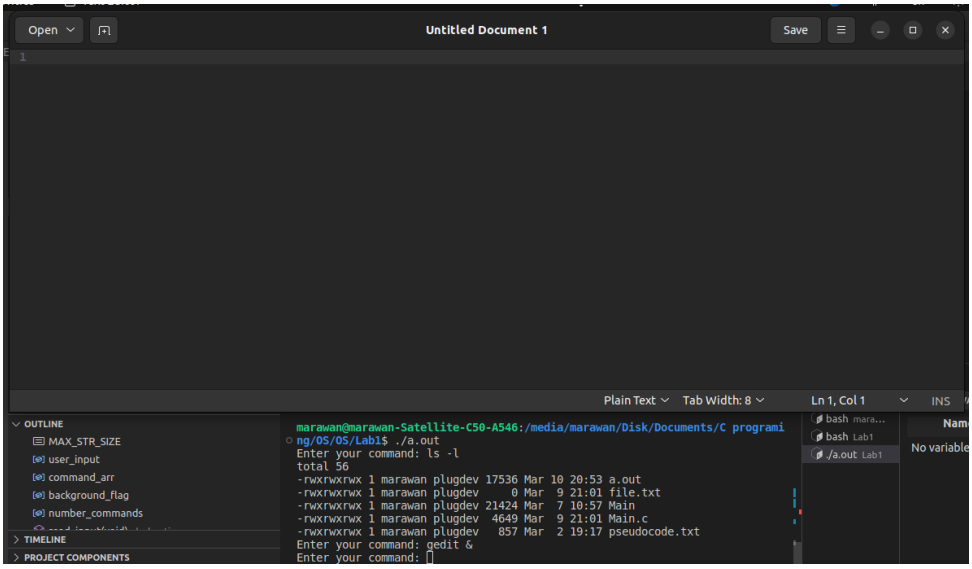

Figure 4: txt file

```c
#include<stdio.h>
#include<sys/types.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include<sys/wait.h>

#include <time.h>

#define MAX_STR_SIZE 100
char user_input[MAX_STR_SIZE];
char *command_arr[MAX_STR_SIZE]; //[MAX_STR_SIZE];
char command_parameter[MAX_STR_SIZE];

int background_flag;
int number_commands;

void read_input(void);
```

```
19 void parse_input(void);
20 int is_builtin(void);
21 void execute_command(void);
22 void signal_handler(int);
23
24 void setup_environment(void){
25     char directory[MAX_STR_SIZE];
26     getcwd(directory,sizeof(directory));
27     chdir(directory);
28 }
29
30 void builtin_cd(void)
31 {
32     if ((number_commands==2) || !(strcmp(command_arr[1], "~"))){
33         chdir(getenv("HOME"));
34     }
35     else if (chdir(command_arr[1]) != 0) {
36         printf("error in cd\n");
37     }
38 }
39
40 void builtin_echo(void)
41 {
42     for(int i = 0; i <= strlen(command_arr[1]); i++) {
43         if (command_arr[1][i] == '"') continue;
44         printf("%c",command_arr[1][i]);
45     }
46     for(int j = 2; j < (number_commands-1); j++){
47         if (j == (number_commands -2)) command_arr[j][strlen(
    command_arr[j])-1] ='\0';
48         printf(" %s", command_arr[j]);
49     }
50     printf("\n");
51     return;
52 }
53
54 void builtin_export(void){
55     putenv(strdup(command_parameter));
56     return;
57 }
58
59
60 int main(void){
61     setup_environment();
62     do{
63         read_input();
64         parse_input ();
65         if (is_builtin()){
66             if(!(strcmp(command_arr[0],"cd"))){
67                 builtin_cd();
68             }
69             else if (!(strcmp(command_arr[0],"echo"))){
70                 builtin_echo();
71             }
72             else if (!(strcmp(command_arr[0],"export"))){
73                 builtin_export();
74             }
75         }
76         else if(!(strcmp(command_arr[0],"exit"))){
77             return;
78         }
```

```
79
80          else{
81              execute_command();
82          }
83      }
84      while (1);
85      return 0;
86  }
87
88  void read_input(void){
89      printf("Enter your command: ");
90      char input_str_buffer[MAX_STR_SIZE];
91      fgets(input_str_buffer, MAX_STR_SIZE, stdin);
92      input_str_buffer[strcspn(input_str_buffer, "\n")] = '\0';
93      int input_index;
94      int output_index = 0;
95
96      for (input_index = 0; input_index < strlen(input_str_buffer)+1;
        input_index++){
97          if (input_str_buffer[input_index] == '$'){
98              char temp [MAX_STR_SIZE];
99              temp[0] = input_str_buffer[input_index+1];
100             temp[1] = '\0';
101             strcpy(temp, getenv(temp));
102             for (int j = 0; j<strlen(temp); j++){
103                 if (temp[j] !='" '){
104                     user_input[output_index] = temp[j];
105                     output_index++;
106                 }
107             }
108             input_index++;
109         }
110         else{
111             user_input[output_index] = input_str_buffer[input_index];
112             output_index++;
113         }
114     }
115 }
116
117 void parse_input (void){
118     int input_length = 0;
119     number_commands = 0;
120     background_flag = 0;
121
122     char * temp = strtok (user_input," ");
123     strcpy(command_parameter, user_input+strlen(temp)+1);
124     while (temp != NULL)
125     {
126         if (!(strcmp(temp, "&"))){
127             background_flag = 1;
128             temp = strtok (NULL, " ");
129             continue;
130         }
131         command_arr[number_commands] = temp;
132         temp = strtok (NULL, " ");
133         number_commands++;
134     }
135     command_arr[number_commands++] = NULL;
136 }
137
138 int is_builtin(void){
```

```
139     if(!(strcmp(command_arr[0], "cd")) || !(strcmp(command_arr[0], "
    echo")) || !(strcmp(command_arr[0], "export"))){
140         return 1;
141     }
142     else{
143         return 0;
144     }
145 }
146 
147 void execute_command(void){
148     pid_t status;
149 
150     int child_id = fork();
151     if (child_id < 0){
152         fprintf(stderr,"Fork Failed");
153         exit(0);
154     }
155 
156     else if (child_id == 0){
157         /*child process*/
158         if(execvp(command_arr[0], command_arr)<0){
159             printf("Error\n");
160         }
161         exit(0);
162     }
163     else {
164         // waiting for child to terminate
165         if(!(background_flag)){
166             waitpid(child_id,NULL,0);
167         }
168         else{
169             signal(SIGCHLD,signal_handler);
170         }
171         return;
172     }
173 }
174 
175 void signal_handler(int sig)
176 {
177     while (waitpid((pid_t)(-1), 0, WNOHANG) > 0) {}
178 
179     time_t rawtime;
180     struct tm * timeinfo;
181 
182     time ( &rawtime );
183     timeinfo = localtime ( &rawtime );
184     FILE *f = fopen("/media/marawan/Disk/Documents/C programing/OS/OS/
    Lab1/file.txt", "a");
185     if (f == NULL)
186     {
187         printf("Error opening file!\n");
188         exit(1);
189     }
190 
191     fprintf(f, "Child was closed in %s \n.........\n", asctime (
    timeinfo));
192 
193     fclose(f);
194 }
```