Operating system

Lab 1

4 functions :

1.to handler signal

2.to handle error

3.to write in a log file when a child process is terminated

4.handling cd

#include <stdio.h>

#include <stdlib.h>

#include<signal.h>

#include <string.h>

#include <sys/wait.h>

#include <errno.h>

#include <unistd.h>

#include <sys/stat.h>

//change directory handler

int cd(char\*path)

{

return chdir(path);

}

//signal interrupt handler

void handle\_sigint()

{

printf("Child process terminated\n");

Write\_in\_log\_file();

}

//writting in a log file

void Write\_in\_log\_file()

{

FILE\*f ;

f=fopen("logfile.log","a");

fprintf(f,"child process Terminated\n"); // fprintf(f,"child process Terminated with id = %d\n",getpid());

fclose(f);

}

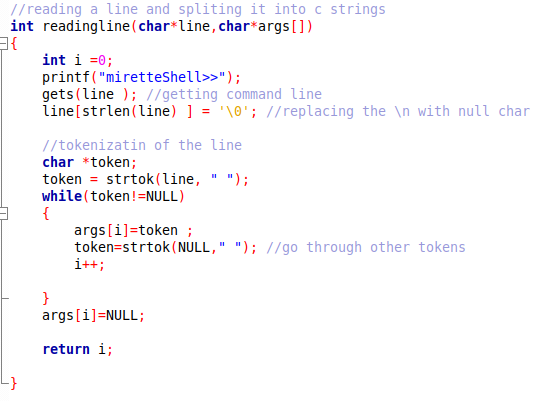
//handle user errors

void handle\_error(int error)

{

printf("Error: %s\n", strerror(errno));

}

Function to read a line and tokinze it in array of args

In the main

We check for conditions l

Then we do fork to create a child process

If the id is 0 -> It’s the child

If -1 -> then failed to create chiled

If +ve -> it’s the parent so I have to check if there is “&”

If present parent continue and doesn’t have to wait else it should wait to child to terminate

int main ()

{

char\*args [10] ; //array of strings

char\*line [200] ; //my line command

int run =1 ;

int status ;

while(run)

{

int size= readingline(line,args);

if (args[0] == NULL)

continue;

if (strcmp(args[0],"exit")==0)

{

run=0 ;

break ;

}

if(strcmp(args[0],"cd")==0)

{

if(cd(args[1])<0)

{

perror(args[1]);

printf("cd failed\n");

}

continue;

}

pid\_t pid = fork();

if(pid==0)

{

int err=execvp (args[0],args);

if (err == -1)

handle\_error(err);

}

else if (pid == -1 )

{

printf("faild of creation of child");

}

else //parent

{

signal(SIGCHLD,handle\_sigint);

if((size -1)>0 && strcmp(args[size-1],"&")==0)

continue;

waitpid(pid,&status,0);

}

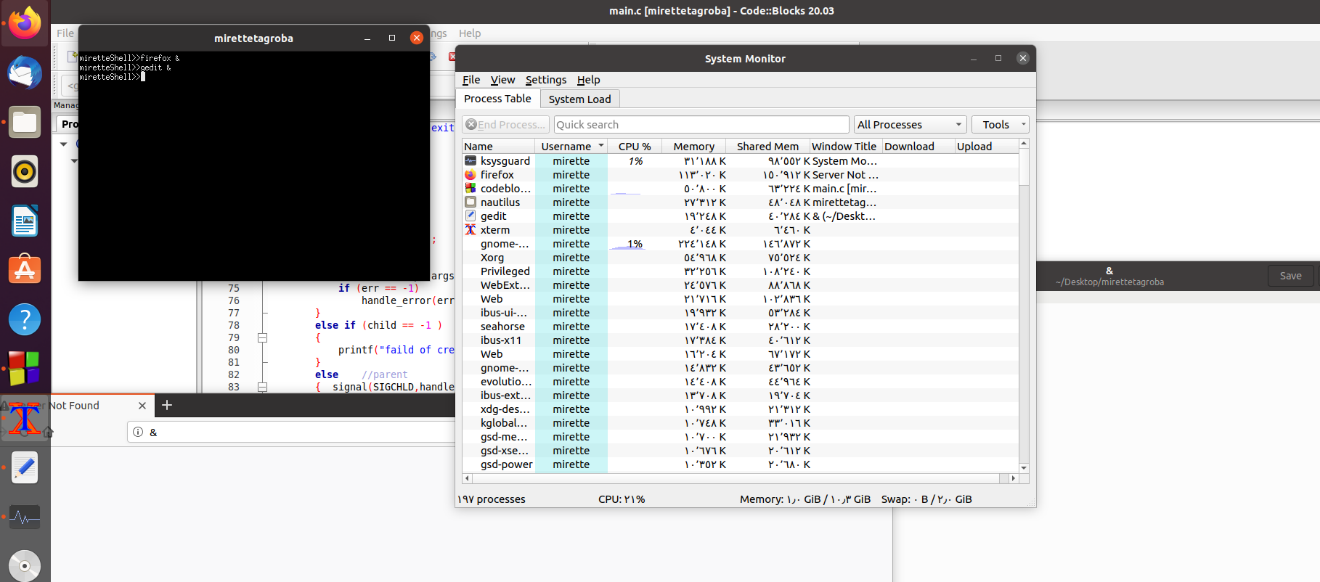
}

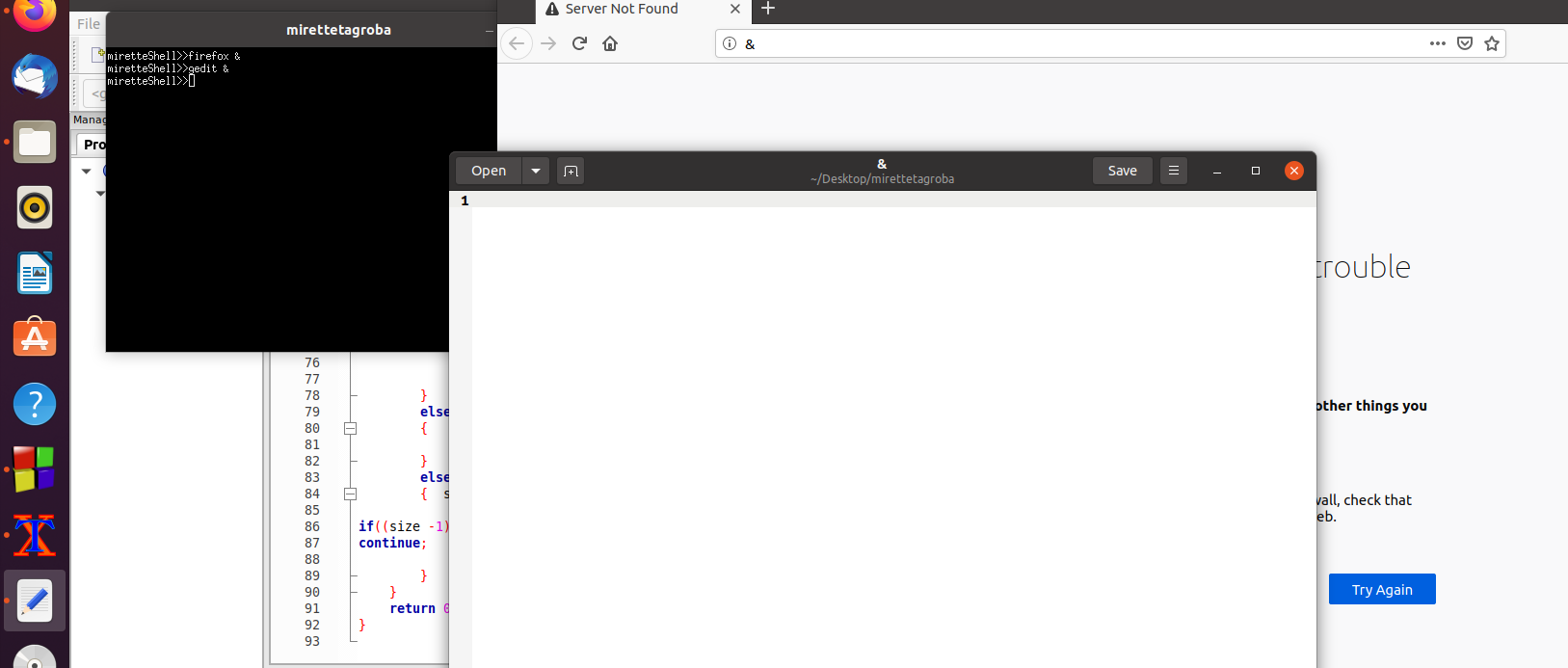
return 0;

}

System monitoring &other screenshots

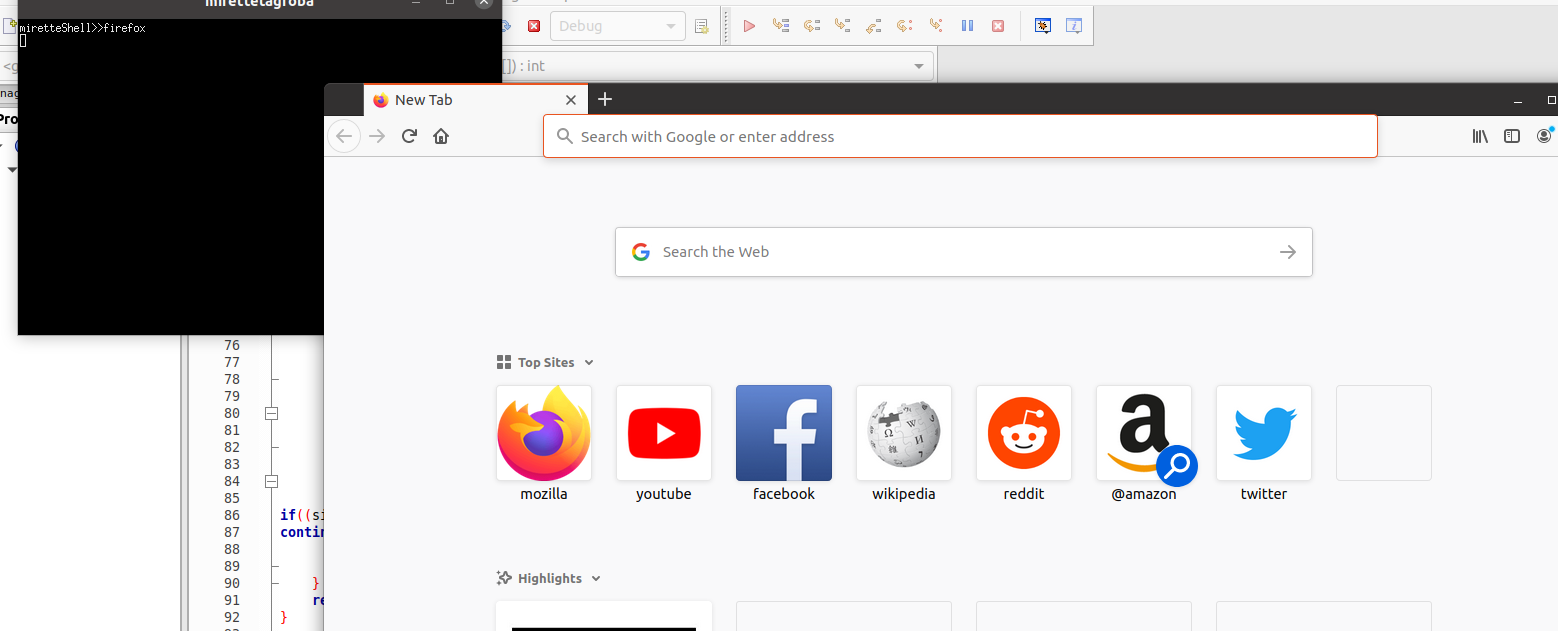
ksysguard



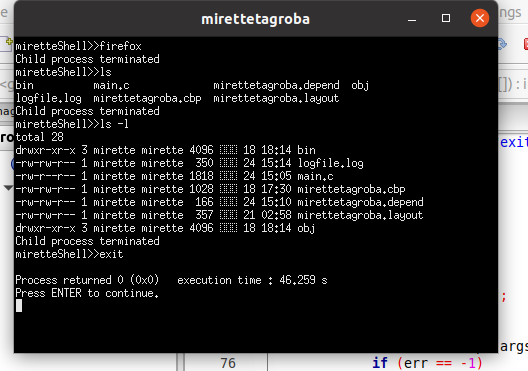
foreground &background

foreground only

the terminal is blocked until the child process terminates



Other command

* Like ls
* Ls -l
* Exit