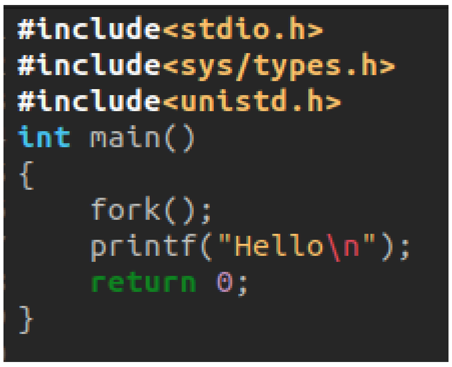
Nithin S
221IT085

IT253 Lab Assignment 4

1. Write Programs using the following system calls of the LINUX operating system: a. fork, exec, getpid, exit, wait, close, opendir, readdir, stat.

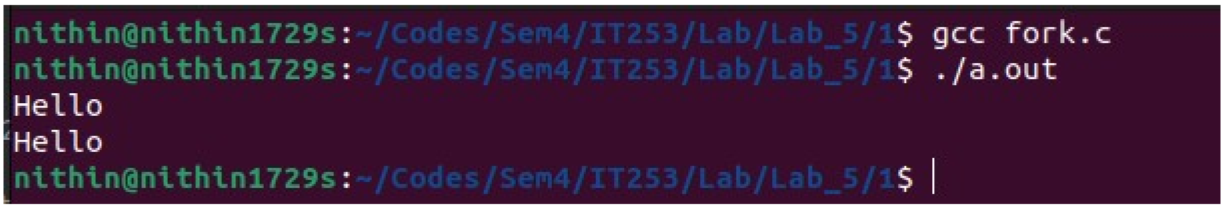
Fork()

Code



```c
#include<stdio.h>
#include<sys/types.h>
#include<unistd.h>
int main()
{
    fork();
    printf("Hello\n");
    return 0;
}
```

Output

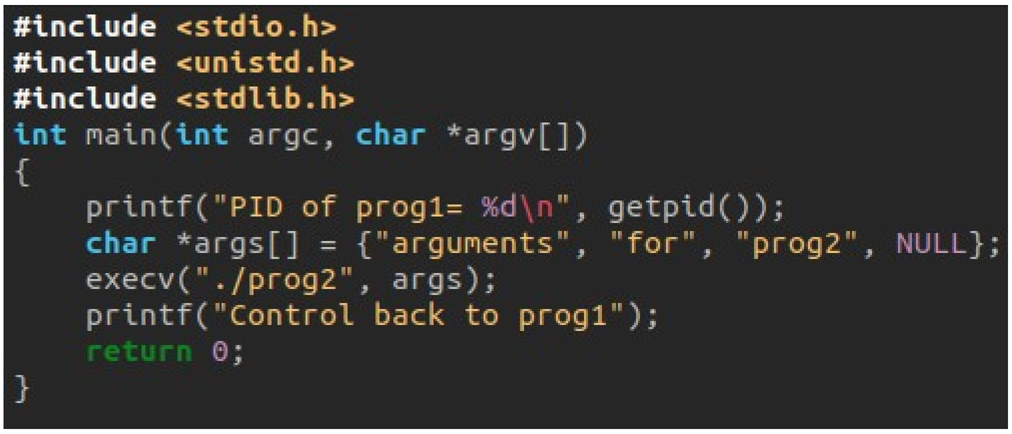


```
nithin@nithin1729s:~/Codes/Sem4/IT253/Lab/Lab_5/1$ gcc fork.c
nithin@nithin1729s:~/Codes/Sem4/IT253/Lab/Lab_5/1$ ./a.out
Hello
Hello
nithin@nithin1729s:~/Codes/Sem4/IT253/Lab/Lab_5/1$ 
```

Exec()

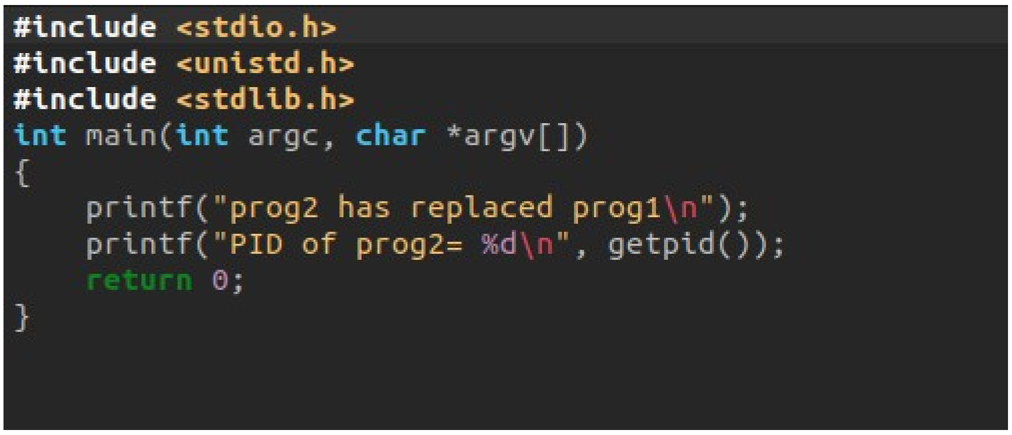
Code

prog1



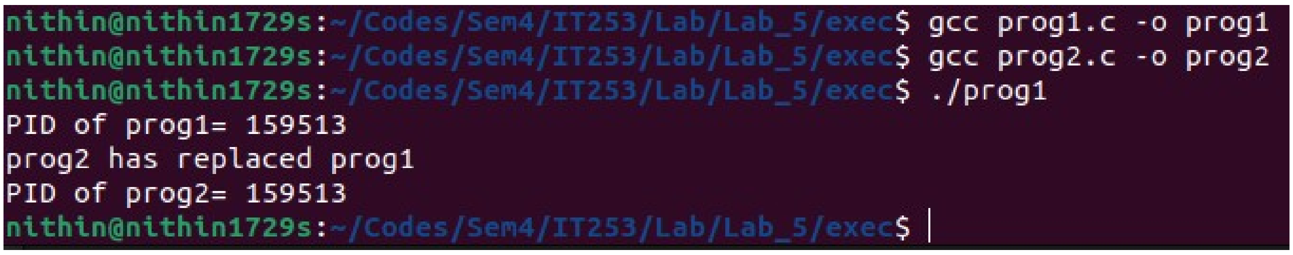
```c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
    printf("PID of prog1= %d\n", getpid());
    char *args[] = {"arguments", "for", "prog2", NULL};
    execv("./prog2", args);
    printf("Control back to prog1");
    return 0;
}
```

prog2



```c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
    printf("prog2 has replaced prog1\n");
    printf("PID of prog2= %d\n", getpid());
    return 0;
}
```

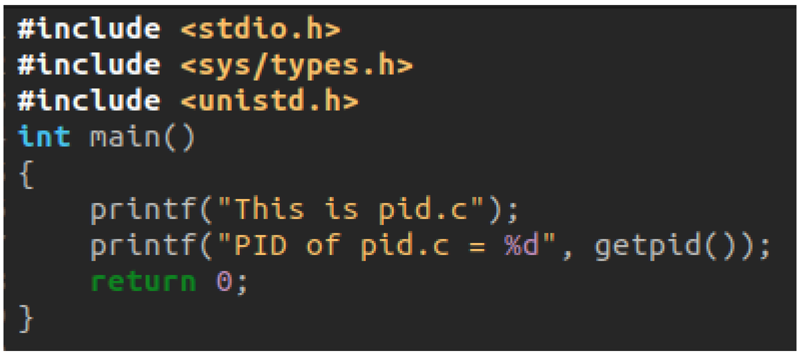
Output



```
nithin@nithin1729s:~/Codes/Sem4/IT253/Lab/Lab_5/exec$ gcc prog1.c -o prog1
nithin@nithin1729s:~/Codes/Sem4/IT253/Lab/Lab_5/exec$ gcc prog2.c -o prog2
nithin@nithin1729s:~/Codes/Sem4/IT253/Lab/Lab_5/exec$ ./prog1
PID of prog1= 159513
prog2 has replaced prog1
PID of prog2= 159513
nithin@nithin1729s:~/Codes/Sem4/IT253/Lab/Lab_5/exec$
```

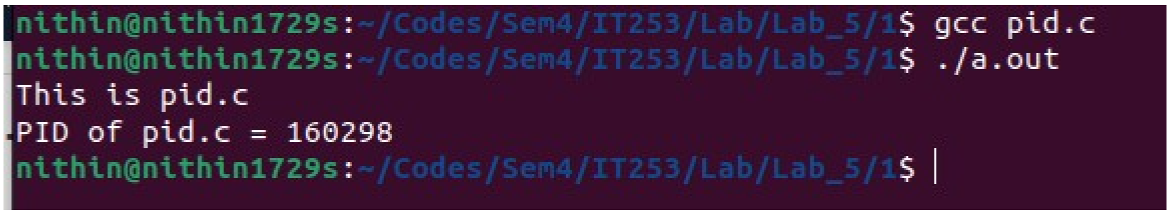
getpid()

Code



```c
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
    printf("This is pid.c");
    printf("PID of pid.c = %d", getpid());
    return 0;
}
```

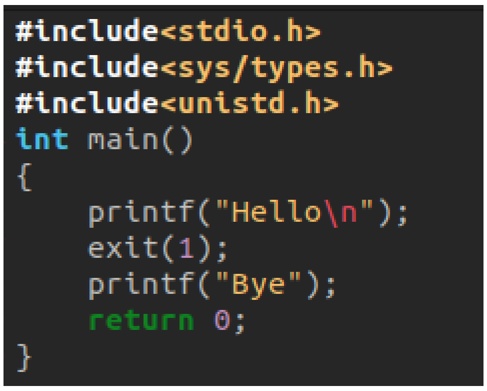
Output



```
nithin@nithin1729s:~/Codes/Sem4/IT253/Lab/Lab_5/1$ gcc pid.c
nithin@nithin1729s:~/Codes/Sem4/IT253/Lab/Lab_5/1$ ./a.out
This is pid.c
PID of pid.c = 160298
nithin@nithin1729s:~/Codes/Sem4/IT253/Lab/Lab_5/1$
```

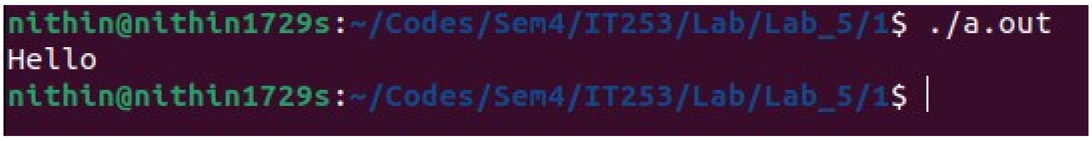
exit()

Code



```c
#include<stdio.h>
#include<sys/types.h>
#include<unistd.h>
int main()
{
    printf("Hello\n");
    exit(1);
    printf("Bye");
    return 0;
}
```

Output

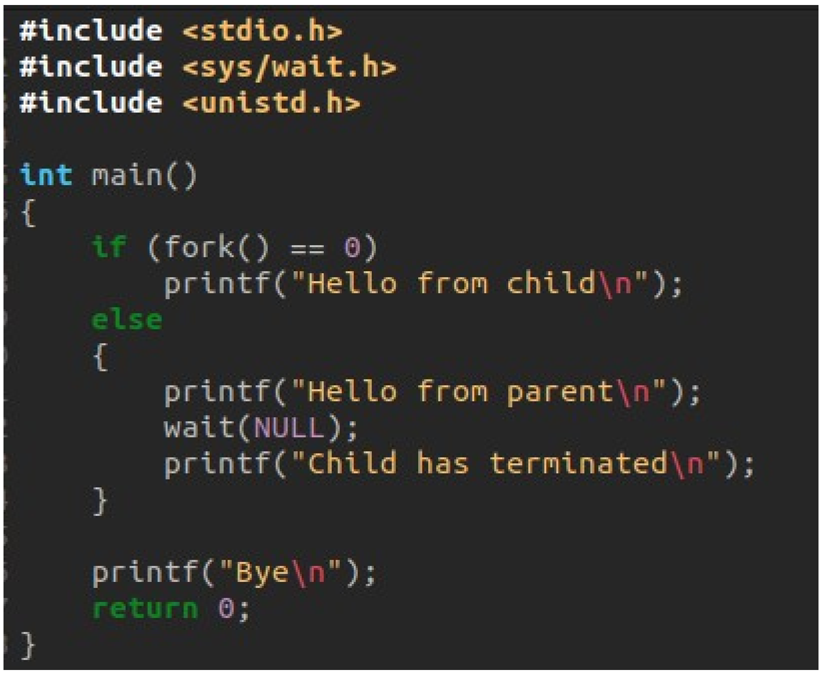


```
nithin@nithin1729s:~/Codes/Sem4/IT253/Lab/Lab_5/1$ ./a.out
Hello
nithin@nithin1729s:~/Codes/Sem4/IT253/Lab/Lab_5/1$
```

Note: Bye was not printed in the output as the proccess terminated before that.

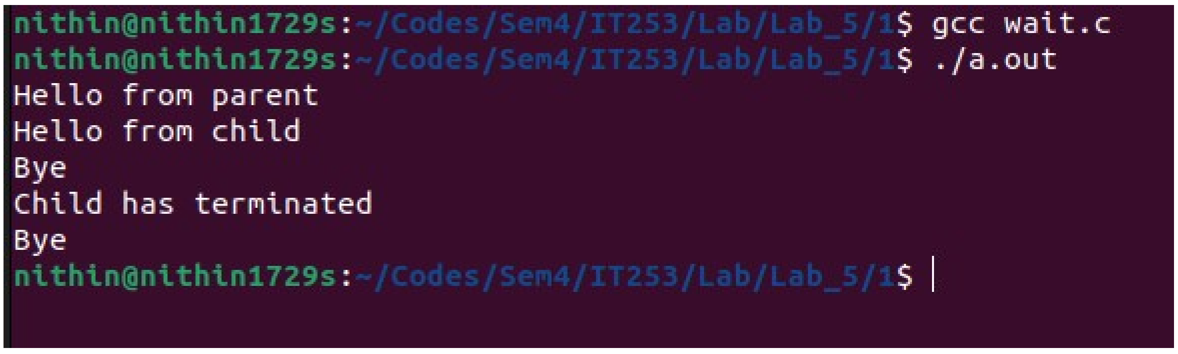
Wait()

Code



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

int main()
{
    if (fork() == 0)
        printf("Hello from child\n");
    else
    {
        printf("Hello from parent\n");
        wait(NULL);
        printf("Child has terminated\n");
    }

    printf("Bye\n");
    return 0;
}
```

OUTPUT

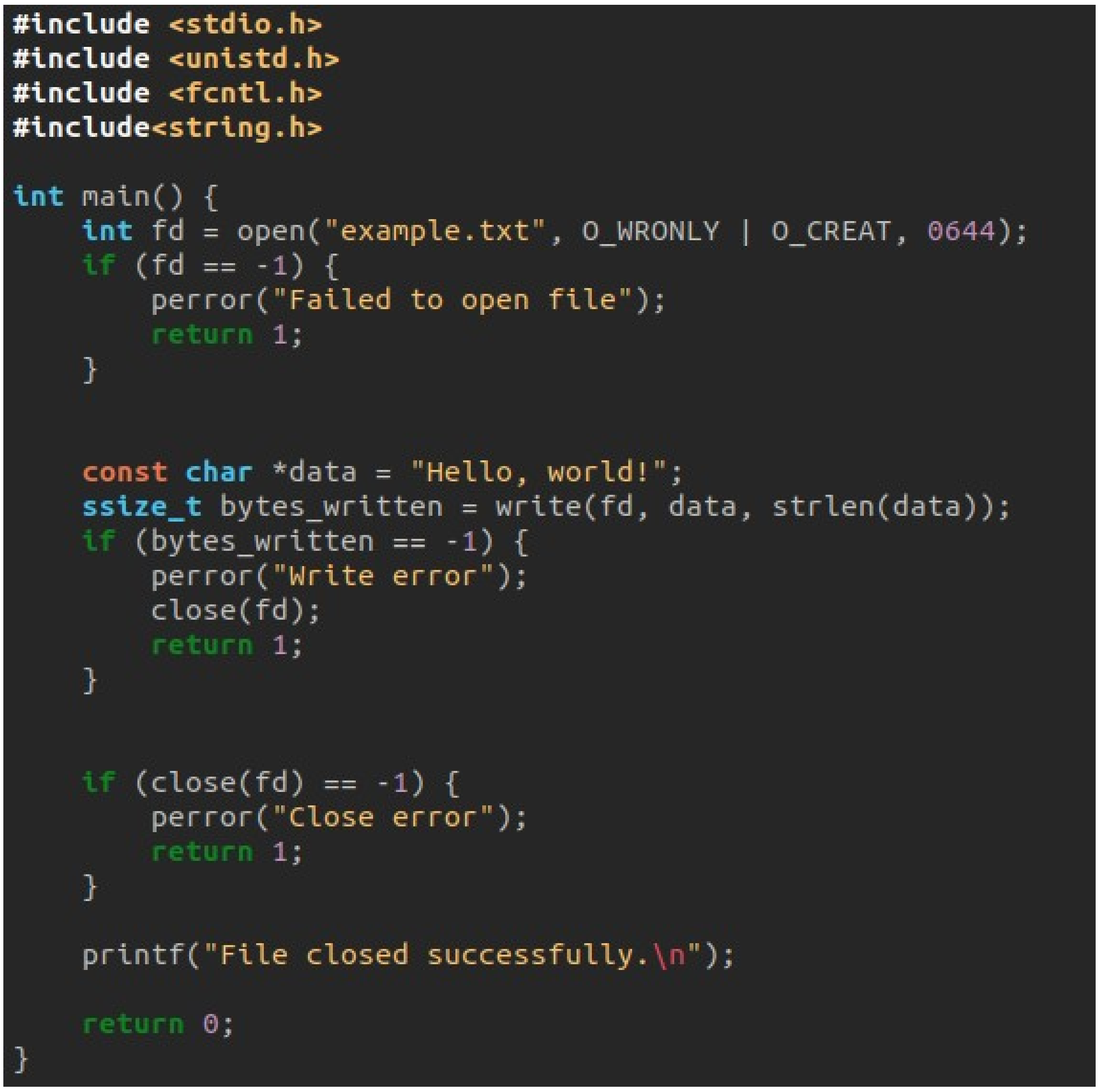


```
nithin@nithin1729s:~/Codes/Sem4/IT253/Lab/Lab_5/1$ gcc wait.c
nithin@nithin1729s:~/Codes/Sem4/IT253/Lab/Lab_5/1$ ./a.out
Hello from parent
Hello from child
Bye
Child has terminated
Bye
nithin@nithin1729s:~/Codes/Sem4/IT253/Lab/Lab_5/1$
```

Note: In the parent process, wait(NULL) is called, which makes the parent process wait for the child process to terminate. The wait() function suspends the execution of the parent process until the child process terminates.

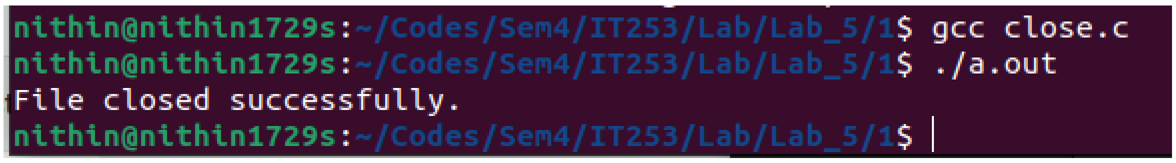
Close()

Code

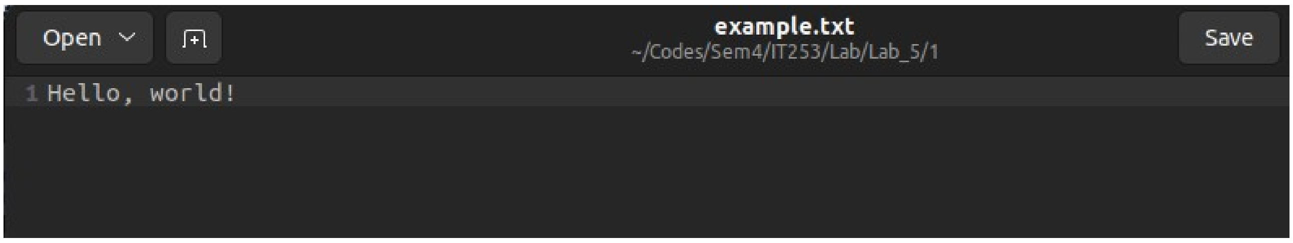


```c
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include<string.h>

int main() {
    int fd = open("example.txt", O_WRONLY | O_CREAT, 0644);
    if (fd == -1) {
        perror("Failed to open file");
        return 1;
    }


    const char *data = "Hello, world!";
    ssize_t bytes_written = write(fd, data, strlen(data));
    if (bytes_written == -1) {
        perror("Write error");
        close(fd);
        return 1;
    }


    if (close(fd) == -1) {
        perror("Close error");
        return 1;
    }

    printf("File closed successfully.\n");

    return 0;
}
```

Output



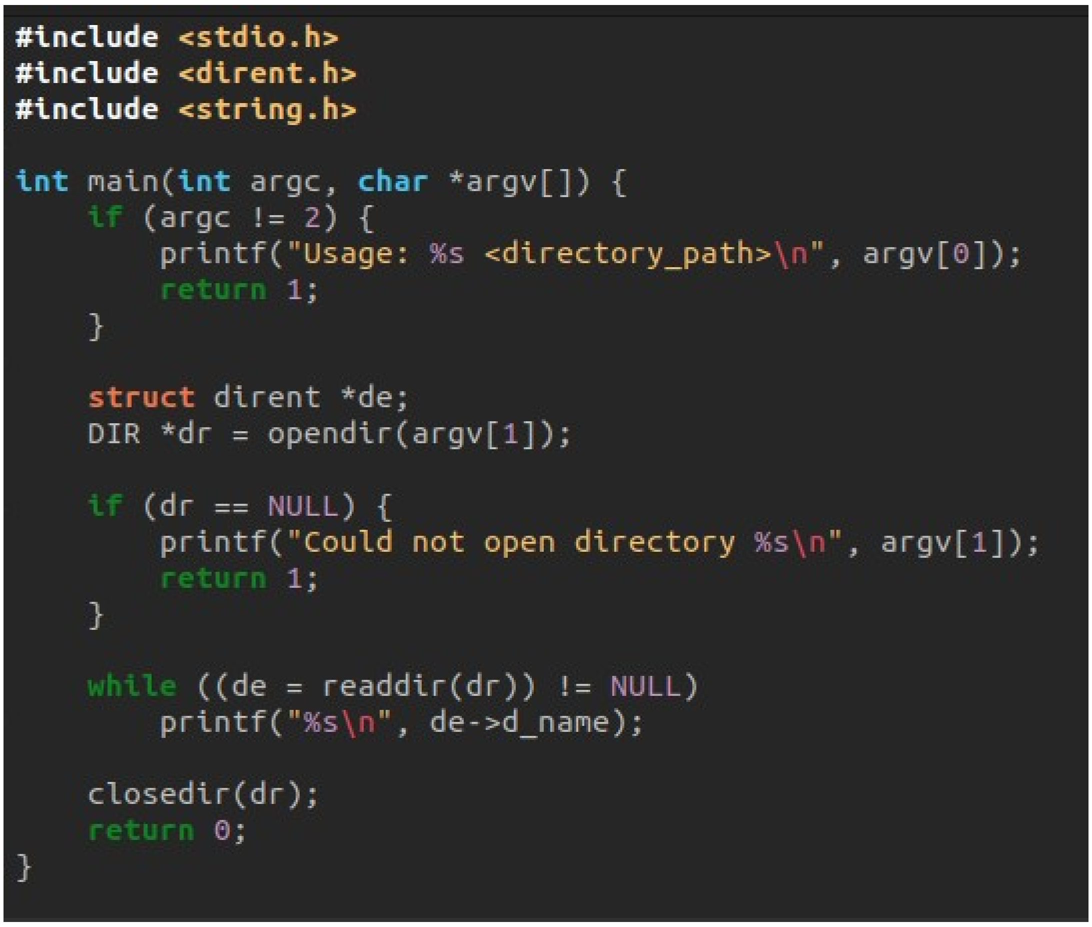
```
nithin@nithin1729s:~/Codes/Sem4/IT253/Lab/Lab_5/1$ gcc close.c
nithin@nithin1729s:~/Codes/Sem4/IT253/Lab/Lab_5/1$ ./a.out
File closed successfully.
nithin@nithin1729s:~/Codes/Sem4/IT253/Lab/Lab_5/1$ 
```



```
Hello, world!
```

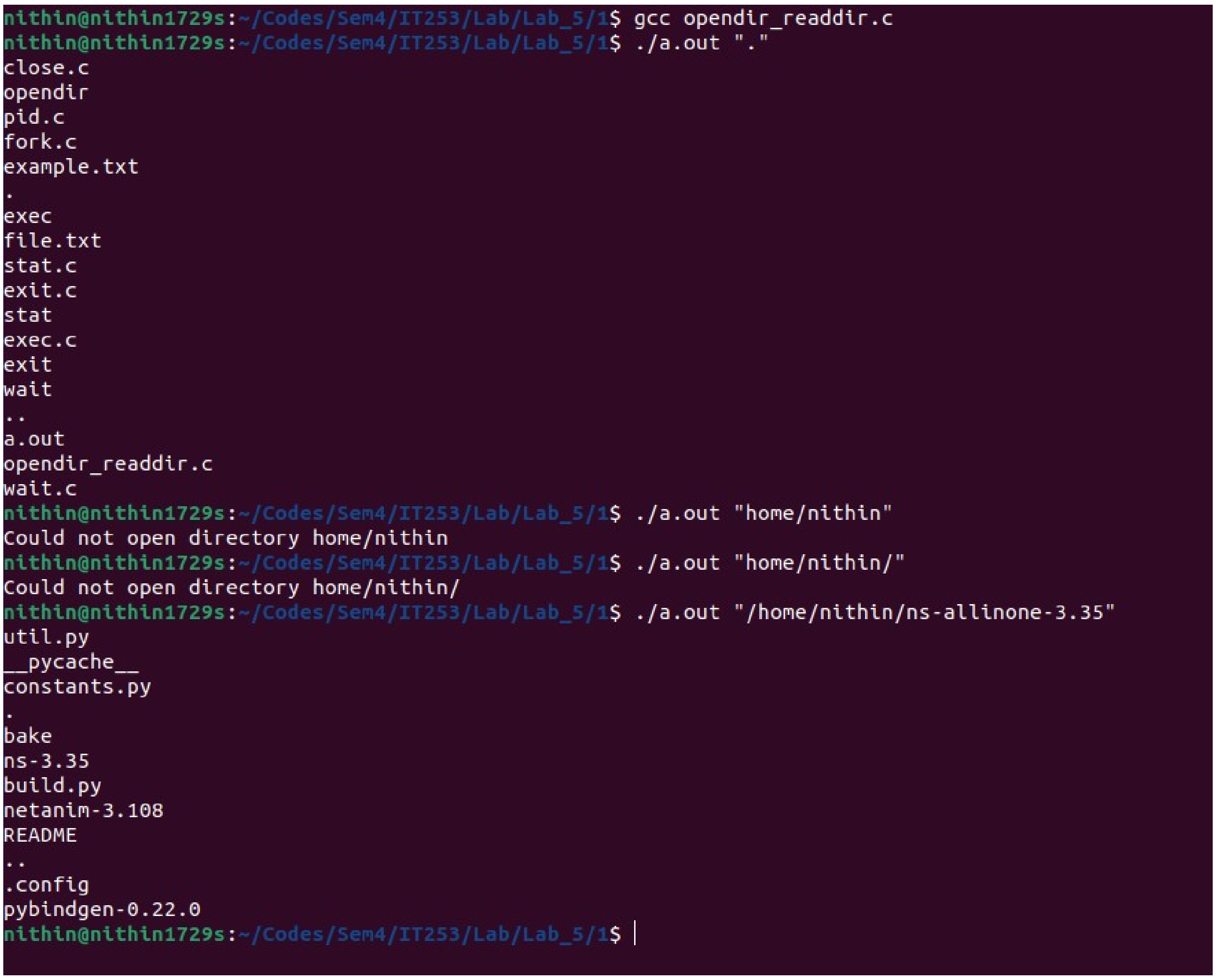
opendir() and readdir()

Code



```c
#include <stdio.h>
#include <dirent.h>
#include <string.h>

int main(int argc, char *argv[]) {
    if (argc != 2) {
        printf("Usage: %s <directory_path>\n", argv[0]);
        return 1;
    }

    struct dirent *de;
    DIR *dr = opendir(argv[1]);

    if (dr == NULL) {
        printf("Could not open directory %s\n", argv[1]);
        return 1;
    }

    while ((de = readdir(dr)) != NULL)
        printf("%s\n", de->d_name);

    closedir(dr);
    return 0;
}
```

Output

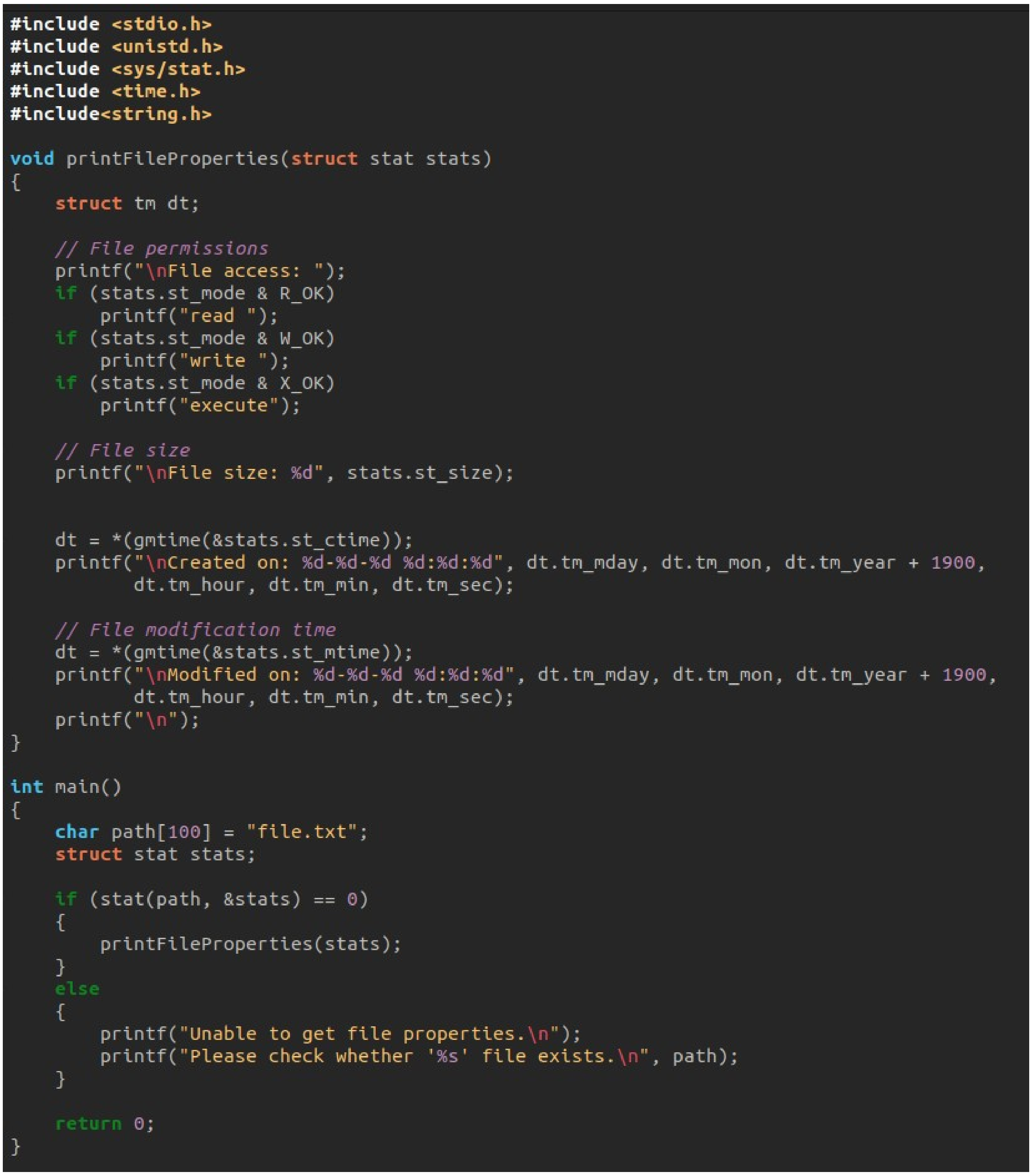


```
nithin@nithin1729s:~/Codes/Sem4/IT253/Lab/Lab_5/1$ gcc opendir_readdir.c
nithin@nithin1729s:~/Codes/Sem4/IT253/Lab/Lab_5/1$ ./a.out "."
close.c
opendir
pid.c
fork.c
example.txt
.
exec
file.txt
stat.c
exit.c
stat
exec.c
exit
wait
..
a.out
opendir_readdir.c
wait.c
nithin@nithin1729s:~/Codes/Sem4/IT253/Lab/Lab_5/1$ ./a.out "home/nithin"
Could not open directory home/nithin
nithin@nithin1729s:~/Codes/Sem4/IT253/Lab/Lab_5/1$ ./a.out "home/nithin/"
Could not open directory home/nithin/
nithin@nithin1729s:~/Codes/Sem4/IT253/Lab/Lab_5/1$ ./a.out "/home/nithin/ns-allinone-3.35"
util.py
__pycache__
constants.py
.
bake
ns-3.35
build.py
netanim-3.108
README
..
.config
pybindgen-0.22.0
nithin@nithin1729s:~/Codes/Sem4/IT253/Lab/Lab_5/1$
```

stat

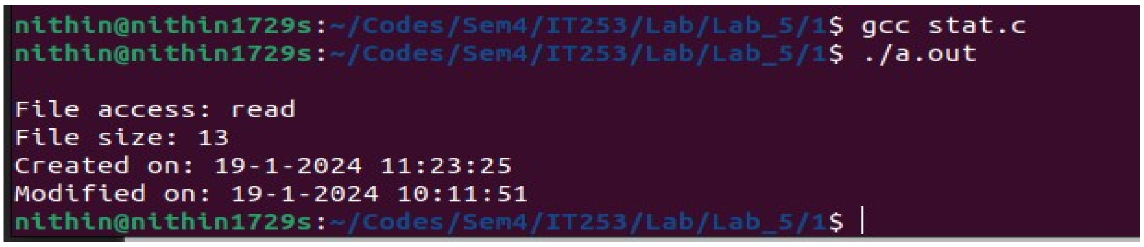
Code

Note: The code is printing the properties of a file named “file.txt”



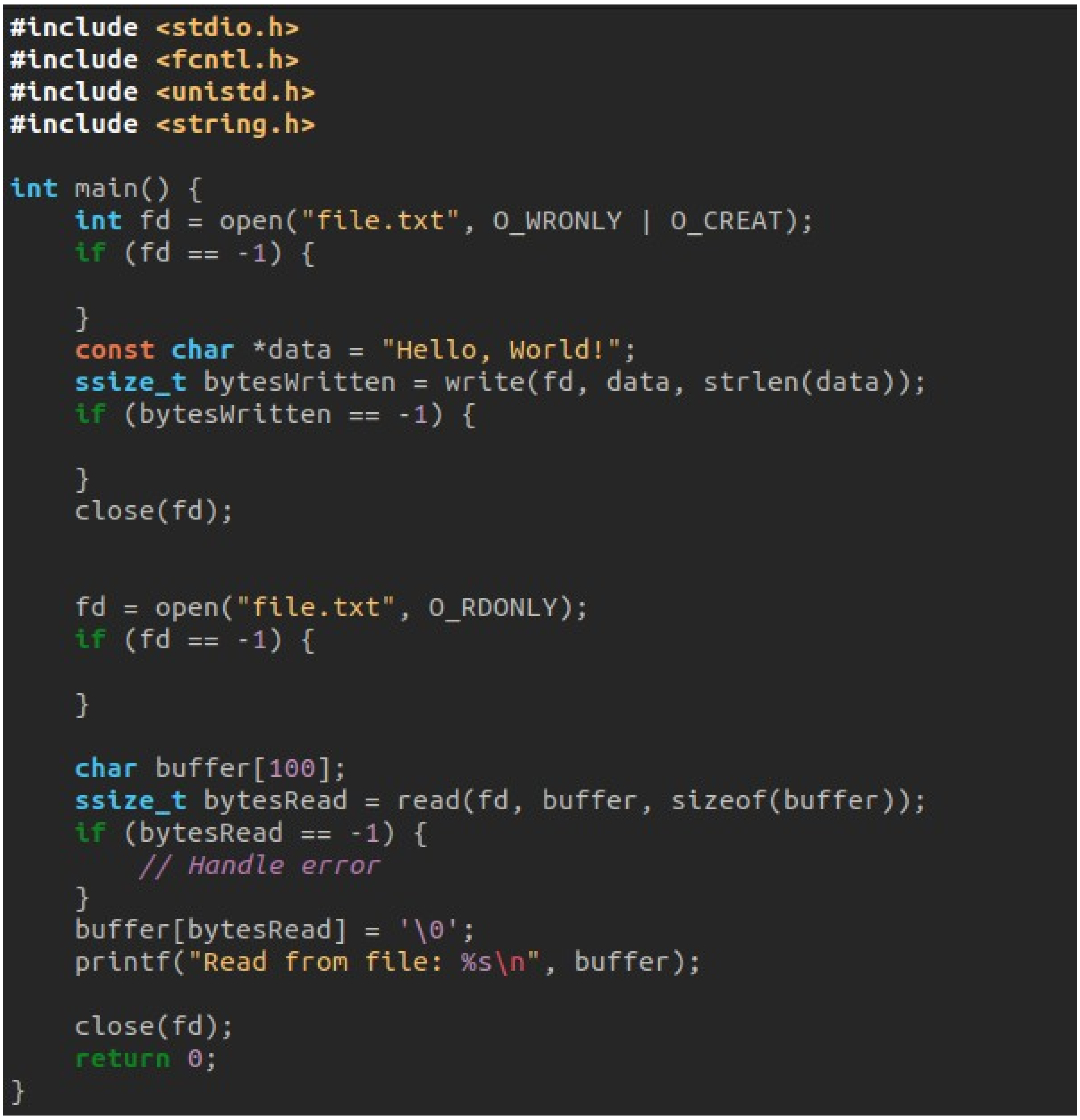
```c
#include <stdio.h>
#include <unistd.h>
#include <sys/stat.h>
#include <time.h>
#include<string.h>

void printFileProperties(struct stat stats)
{
    struct tm dt;

    // File permissions
    printf("\nFile access: ");
    if (stats.st_mode & R_OK)
        printf("read ");
    if (stats.st_mode & W_OK)
        printf("write ");
    if (stats.st_mode & X_OK)
        printf("execute");

    // File size
    printf("\nFile size: %d", stats.st_size);


    dt = *(gmtime(&stats.st_ctime));
    printf("\nCreated on: %d-%d-%d %d:%d:%d", dt.tm_mday, dt.tm_mon, dt.tm_year + 1900,
           dt.tm_hour, dt.tm_min, dt.tm_sec);

    // File modification time
    dt = *(gmtime(&stats.st_mtime));
    printf("\nModified on: %d-%d-%d %d:%d:%d", dt.tm_mday, dt.tm_mon, dt.tm_year + 1900,
           dt.tm_hour, dt.tm_min, dt.tm_sec);
    printf("\n");
}

int main()
{
    char path[100] = "file.txt";
    struct stat stats;

    if (stat(path, &stats) == 0)
    {
        printFileProperties(stats);
    }
    else
    {
        printf("Unable to get file properties.\n");
        printf("Please check whether '%s' file exists.\n", path);
    }

    return 0;
}
```

Output



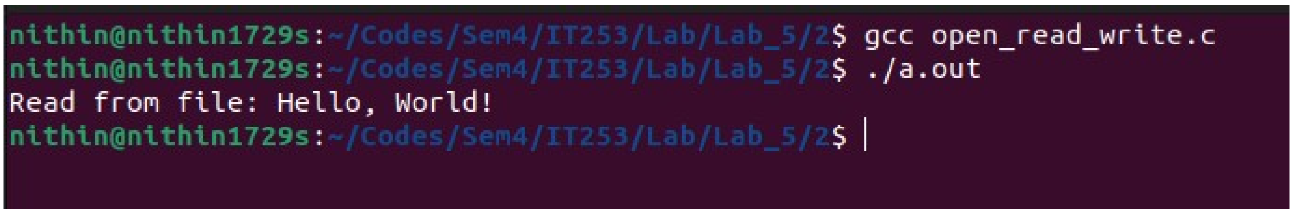
```
nithin@nithin1729s:~/Codes/Sem4/IT253/Lab/Lab_5/1$ gcc stat.c
nithin@nithin1729s:~/Codes/Sem4/IT253/Lab/Lab_5/1$ ./a.out

File access: read
File size: 13
Created on: 19-1-2024 11:23:25
Modified on: 19-1-2024 10:11:51
nithin@nithin1729s:~/Codes/Sem4/IT253/Lab/Lab_5/1$ |
```

Q2. Write Programs using I/O system calls of LINUX operating system: open, read, write etc.



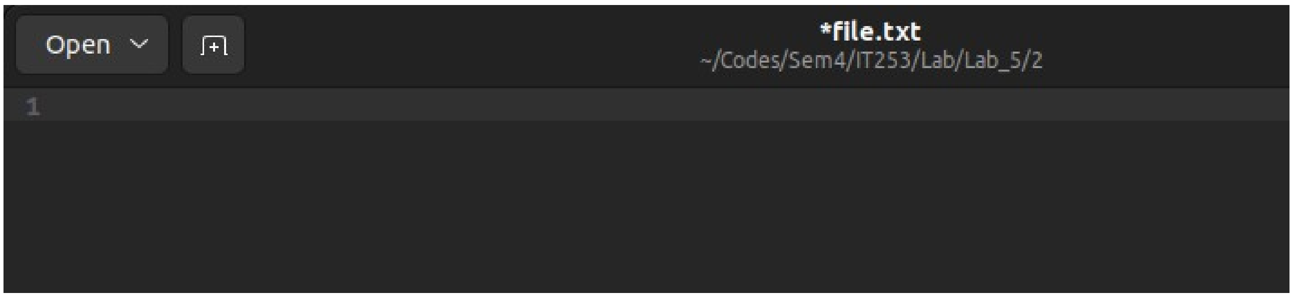
```c
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#include <string.h>

int main() {
    int fd = open("file.txt", O_WRONLY | O_CREAT);
    if (fd == -1) {

    }
    const char *data = "Hello, World!";
    ssize_t bytesWritten = write(fd, data, strlen(data));
    if (bytesWritten == -1) {

    }
    close(fd);


    fd = open("file.txt", O_RDONLY);
    if (fd == -1) {

    }

    char buffer[100];
    ssize_t bytesRead = read(fd, buffer, sizeof(buffer));
    if (bytesRead == -1) {
        // Handle error
    }
    buffer[bytesRead] = '\0';
    printf("Read from file: %s\n", buffer);

    close(fd);
    return 0;
}
```

Output

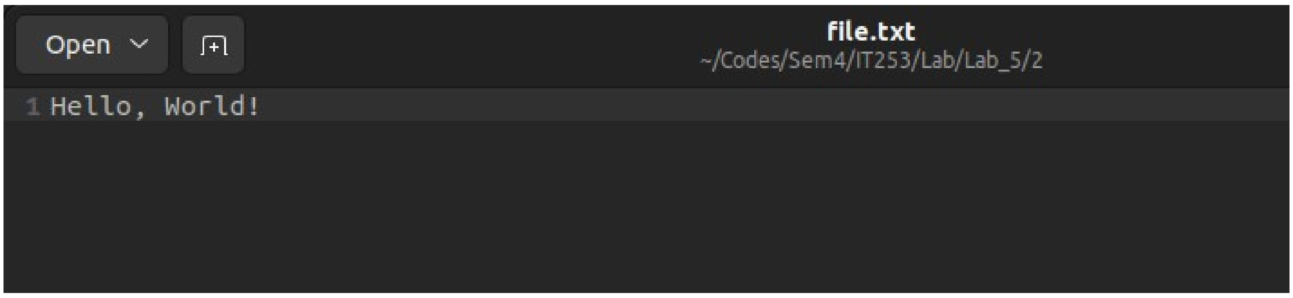


```
nithin@nithin1729s:~/Codes/Sem4/IT253/Lab/Lab_5/2$ gcc open_read_write.c
nithin@nithin1729s:~/Codes/Sem4/IT253/Lab/Lab_5/2$ ./a.out
Read from file: Hello, World!
nithin@nithin1729s:~/Codes/Sem4/IT253/Lab/Lab_5/2$ 
```

Before:



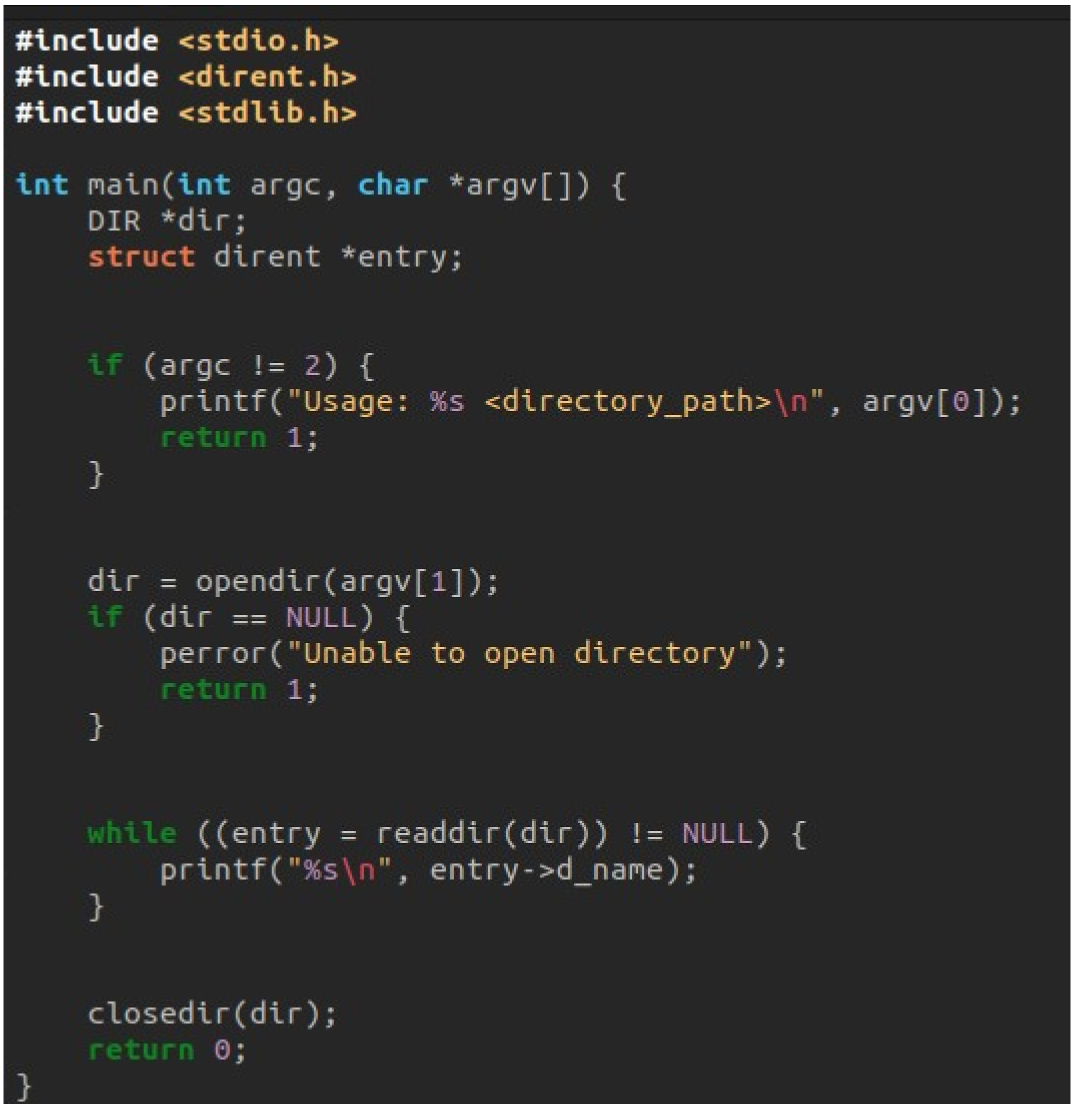
after:



```
Hello, World!
```

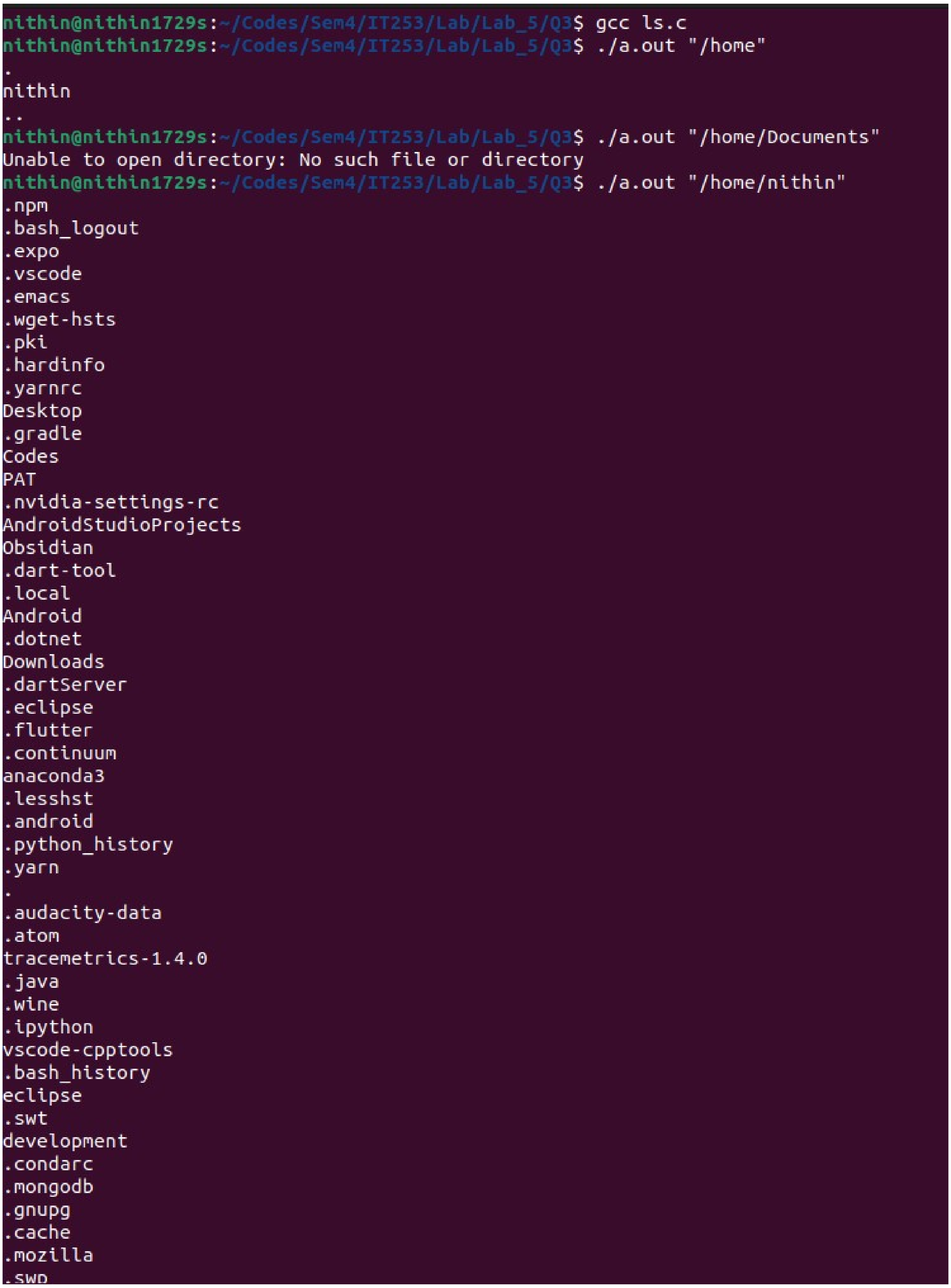
Q3.Write Programs using C to simulate LINUX commands
a. ls command, b. grep command

ls



```c
#include <stdio.h>
#include <dirent.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
    DIR *dir;
    struct dirent *entry;


    if (argc != 2) {
        printf("Usage: %s <directory_path>\n", argv[0]);
        return 1;
    }


    dir = opendir(argv[1]);
    if (dir == NULL) {
        perror("Unable to open directory");
        return 1;
    }


    while ((entry = readdir(dir)) != NULL) {
        printf("%s\n", entry->d_name);
    }


    closedir(dir);
    return 0;
}
```

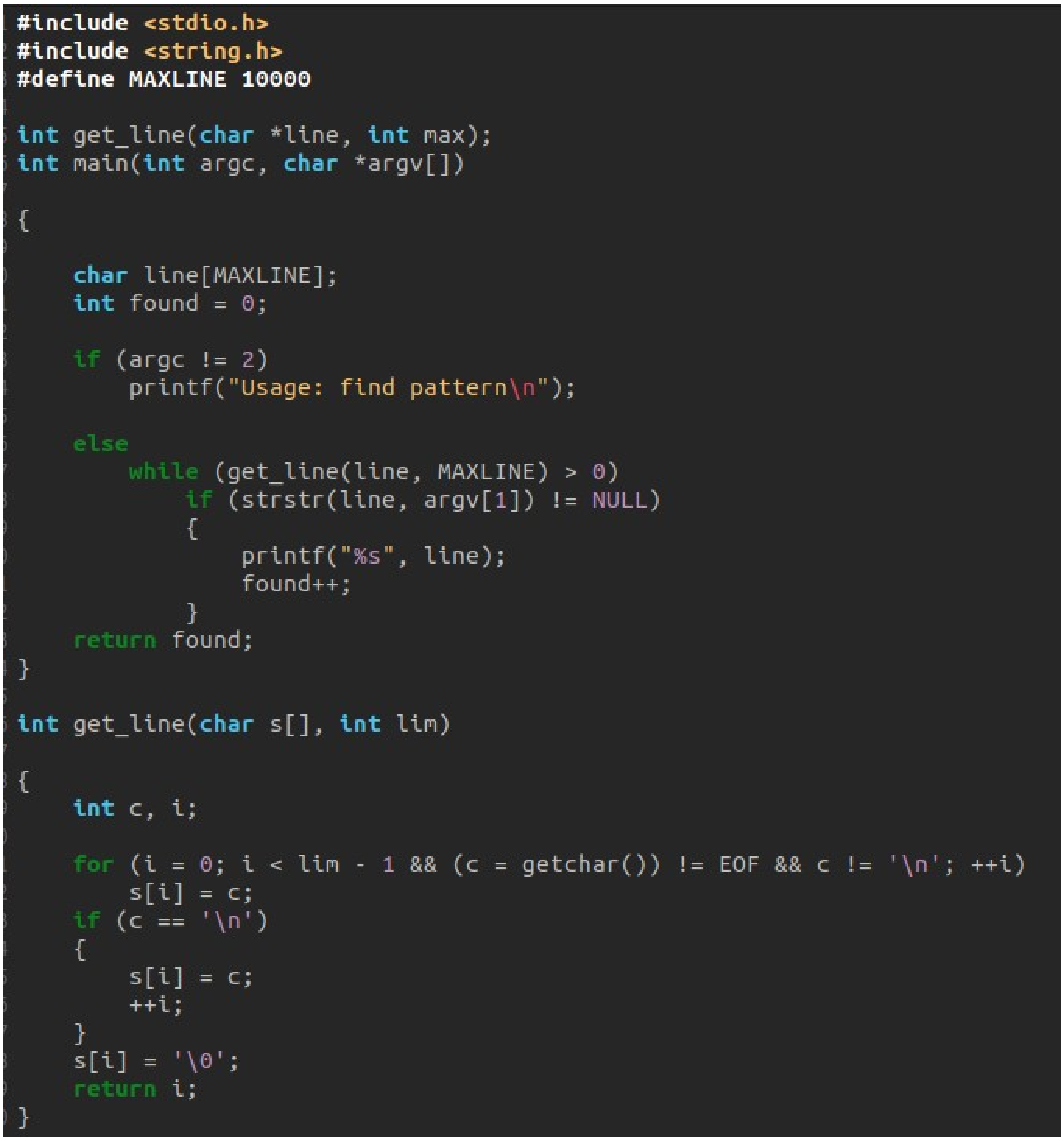
Output



```
nithin@nithin1729s:~/Codes/Sem4/IT253/Lab/Lab_5/Q3$ gcc ls.c
nithin@nithin1729s:~/Codes/Sem4/IT253/Lab/Lab_5/Q3$ ./a.out "/home"
.
nithin
..
nithin@nithin1729s:~/Codes/Sem4/IT253/Lab/Lab_5/Q3$ ./a.out "/home/Documents"
Unable to open directory: No such file or directory
nithin@nithin1729s:~/Codes/Sem4/IT253/Lab/Lab_5/Q3$ ./a.out "/home/nithin"
.npm
.bash_logout
.expo
.vscode
.emacs
.wget-hsts
.pki
.hardinfo
.yarnrc
Desktop
.gradle
Codes
PAT
.nvidia-settings-rc
AndroidStudioProjects
Obsidian
.dart-tool
.local
Android
.dotnet
Downloads
.dartServer
.eclipse
.flutter
.continuum
anaconda3
.lesshst
.android
.python_history
.yarn
.
.audacity-data
.atom
tracemetrics-1.4.0
.java
.wine
.ipython
vscode-cpptools
.bash_history
eclipse
.swt
development
.condarc
.mongodb
.gnupg
.cache
.mozilla
.swp
```

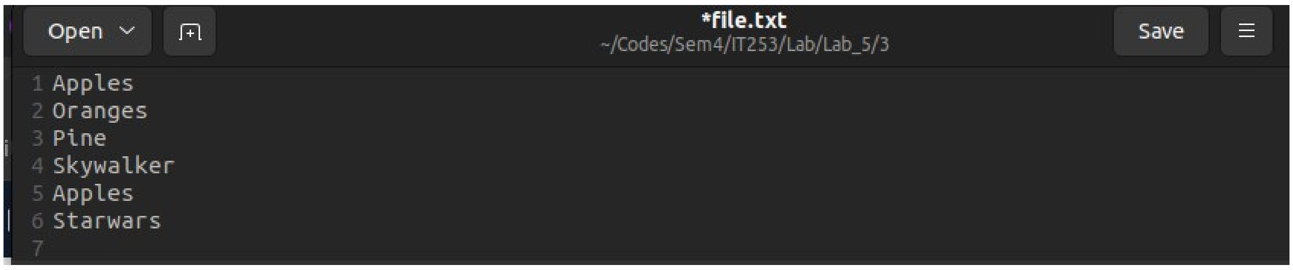
grep

Code

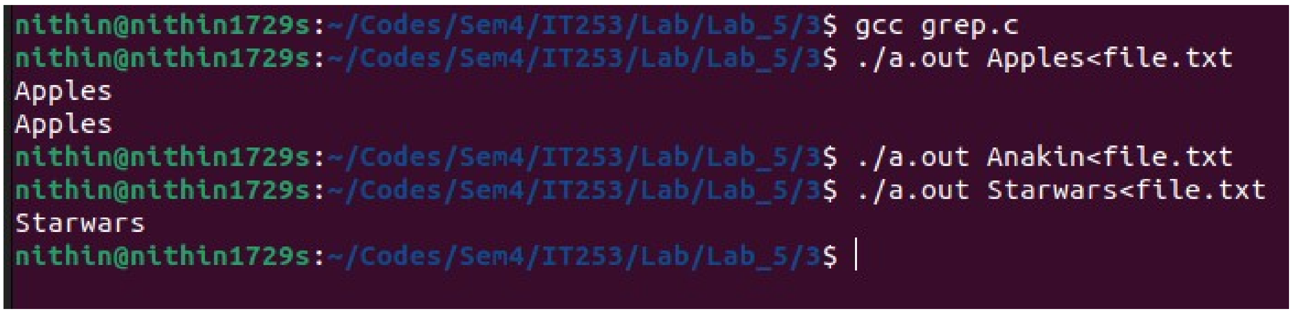


```c
#include <stdio.h>
#include <string.h>
#define MAXLINE 10000

int get_line(char *line, int max);
int main(int argc, char *argv[])

{

    char line[MAXLINE];
    int found = 0;

    if (argc != 2)
        printf("Usage: find pattern\n");

    else
        while (get_line(line, MAXLINE) > 0)
            if (strstr(line, argv[1]) != NULL)
            {
                printf("%s", line);
                found++;
            }
    return found;
}

int get_line(char s[], int lim)

{
    int c, i;

    for (i = 0; i < lim - 1 && (c = getchar()) != EOF && c != '\n'; ++i)
        s[i] = c;
    if (c == '\n')
    {
        s[i] = c;
        ++i;
    }
    s[i] = '\0';
    return i;
}
```

Output



```
Apples
Oranges
Pine
Skywalker
Apples
Starwars

```

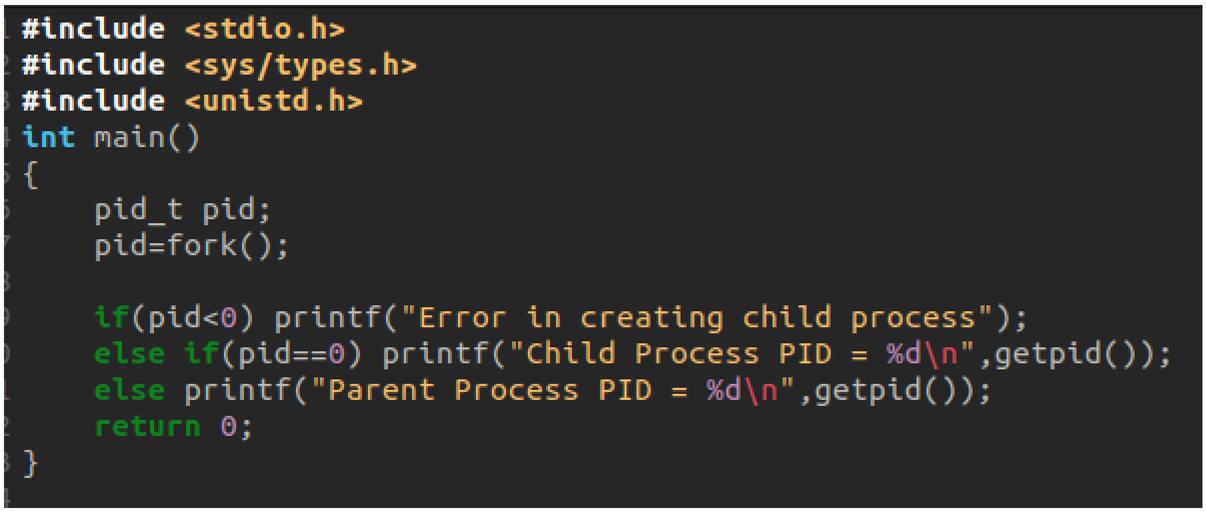


```
nithin@nithin1729s:~/Codes/Sem4/IT253/Lab/Lab_5/3$ gcc grep.c
nithin@nithin1729s:~/Codes/Sem4/IT253/Lab/Lab_5/3$ ./a.out Apples<file.txt
Apples
Apples
nithin@nithin1729s:~/Codes/Sem4/IT253/Lab/Lab_5/3$ ./a.out Anakin<file.txt
nithin@nithin1729s:~/Codes/Sem4/IT253/Lab/Lab_5/3$ ./a.out Starwars<file.txt
Starwars
nithin@nithin1729s:~/Codes/Sem4/IT253/Lab/Lab_5/3$ 
```

Note: ‘<’ means take the file to the right and make it the stdin for the program to the left.

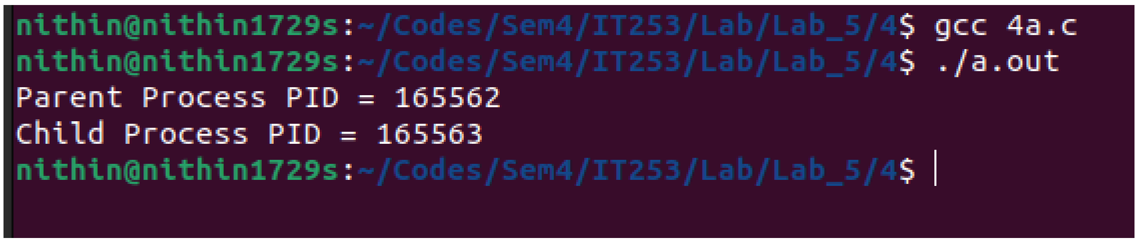
Q4. Write a program: a) To create parent & child process and print their id.

Code



```c
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
    pid_t pid;
    pid=fork();

    if(pid<0) printf("Error in creating child process");
    else if(pid==0) printf("Child Process PID = %d\n",getpid());
    else printf("Parent Process PID = %d\n",getpid());
    return 0;
}
```

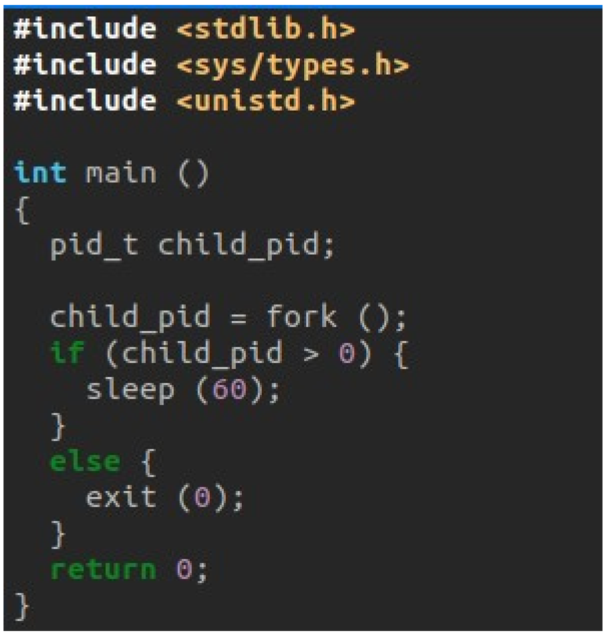
Output



```
nithin@nithin1729s:~/Codes/Sem4/IT253/Lab/Lab_5/4$ gcc 4a.c
nithin@nithin1729s:~/Codes/Sem4/IT253/Lab/Lab_5/4$ ./a.out
Parent Process PID = 165562
Child Process PID = 165563
nithin@nithin1729s:~/Codes/Sem4/IT253/Lab/Lab_5/4$
```

Q4. Write a program: b) To create a zombie process.

Code

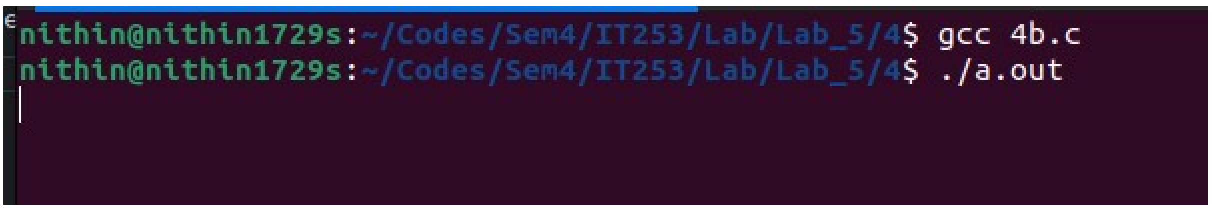


```c
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>

int main ()
{
  pid_t child_pid;

  child_pid = fork ();
  if (child_pid > 0) {
    sleep (60);
  }
  else {
    exit (0);
  }
  return 0;
}
```

Note: A zombie or a "defunct process" in Linux is a process that has been completed, but its entry still remains in the process table due to lack of correspondence between the parent and child processes.
Usually, a parent process keeps a check on the status of its child processes through the wait() function. When the child process has finished, the wait function signals the parent to completely exit the process from the memory.
However, if the parent fails to call the wait function for any of its children, the child process remains alive in the system as a dead or zombie process.
These zombie processes might accumulate, in large numbers, on your system and affect its performance.
The zombie process created through this code will run for 60 seconds. You can increase the time duration by

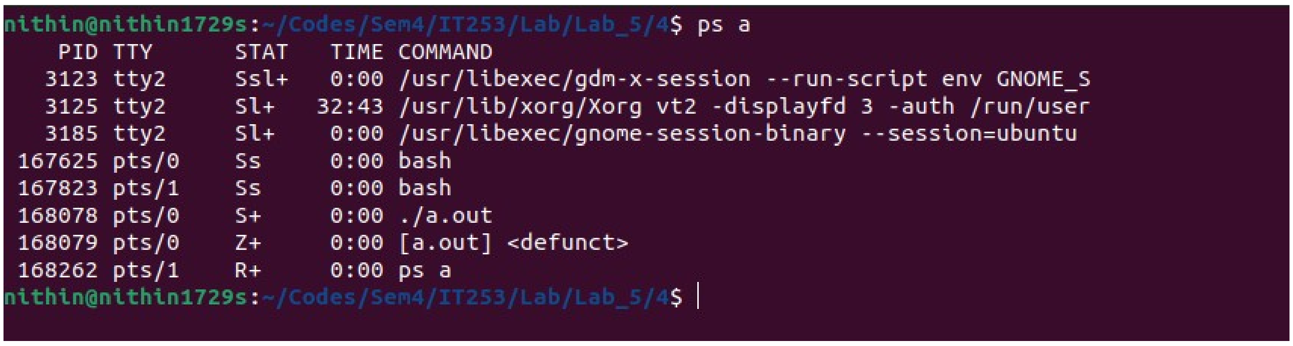
specifying a time(in seconds) in the sleep() function.

Output



```
nithin@nithin1729s:~/Codes/Sem4/IT253/Lab/Lab_5/4$ gcc 4b.c
nithin@nithin1729s:~/Codes/Sem4/IT253/Lab/Lab_5/4$ ./a.out
```

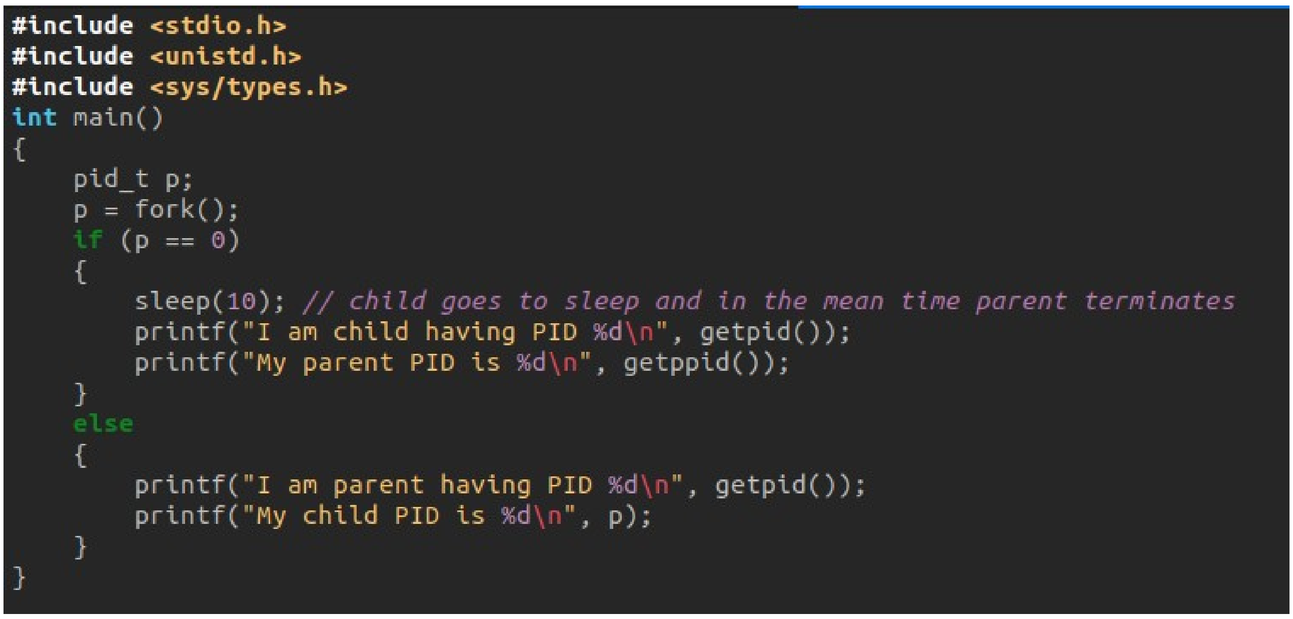
The ps command will now also show this defunct process, open a new terminal and use the below command to check the defunct process



```
nithin@nithin1729s:~/Codes/Sem4/IT253/Lab/Lab_5/4$ ps a
    PID TTY      STAT   TIME COMMAND
   3123 tty2     Ssl+   0:00 /usr/libexec/gdm-x-session --run-script env GNOME_S
   3125 tty2     Sl+   32:43 /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /run/user
   3185 tty2     Sl+    0:00 /usr/libexec/gnome-session-binary --session=ubuntu
 167625 pts/0    Ss     0:00 bash
 167823 pts/1    Ss     0:00 bash
 168078 pts/0    S+     0:00 ./a.out
 168079 pts/0    Z+     0:00 [a.out] <defunct>
 168262 pts/1    R+     0:00 ps a
nithin@nithin1729s:~/Codes/Sem4/IT253/Lab/Lab_5/4$
```

Q4. Write a program: c) To create an orphan process.

Code

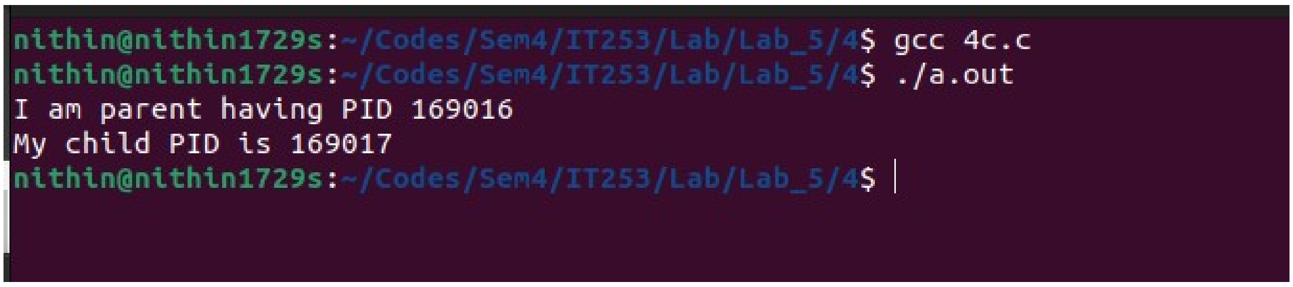


```c
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
int main()
{
    pid_t p;
    p = fork();
    if (p == 0)
    {
        sleep(10); // child goes to sleep and in the mean time parent terminates
        printf("I am child having PID %d\n", getpid());
        printf("My parent PID is %d\n", getppid());
    }
    else
    {
        printf("I am parent having PID %d\n", getpid());
        printf("My child PID is %d\n", p);
    }
}
```

Note:

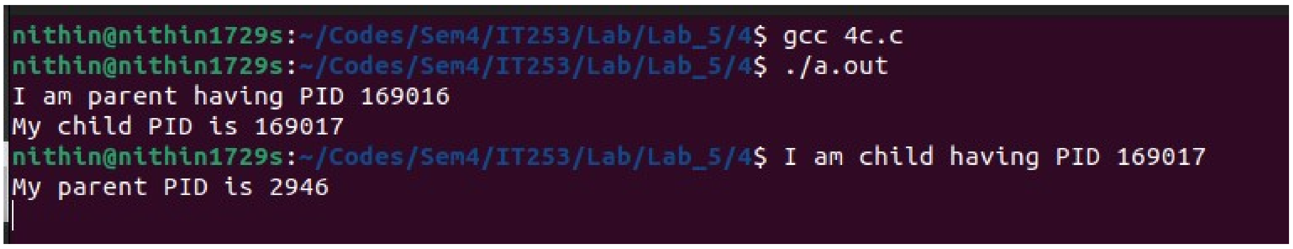
In this code, we add sleep(10) in the child section. This line of code makes the child process go to sleep for 10 seconds and the parent starts executing. Since, parent process has just two lines to print, which it does well within 10 seconds and it terminates. After 10 seconds when the child process wakes up, its parent has already terminated and hence the child becomes an orphan process. The child process will be made the child of another process so we can see in the ouput screenshots that the parent id of child changes.

Output



```
nithin@nithin1729s:~/Codes/Sem4/IT253/Lab/Lab_5/4$ gcc 4c.c
nithin@nithin1729s:~/Codes/Sem4/IT253/Lab/Lab_5/4$ ./a.out
I am parent having PID 169016
My child PID is 169017
nithin@nithin1729s:~/Codes/Sem4/IT253/Lab/Lab_5/4$ 
```

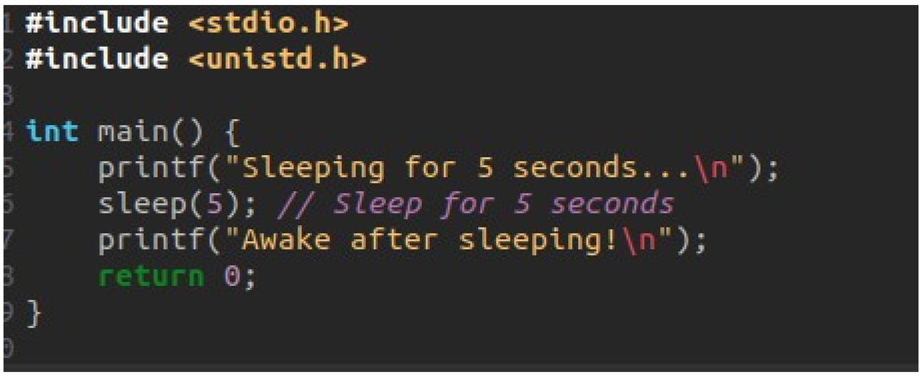
After 10 seconds,



```
nithin@nithin1729s:~/Codes/Sem4/IT253/Lab/Lab_5/4$ gcc 4c.c
nithin@nithin1729s:~/Codes/Sem4/IT253/Lab/Lab_5/4$ ./a.out
I am parent having PID 169016
My child PID is 169017
nithin@nithin1729s:~/Codes/Sem4/IT253/Lab/Lab_5/4$ I am child having PID 169017
My parent PID is 2946
```

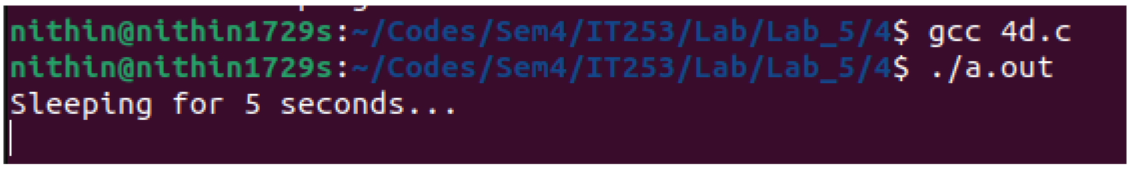
Q4. Write a program: d) To make the process to sleep for a few seconds.

Code



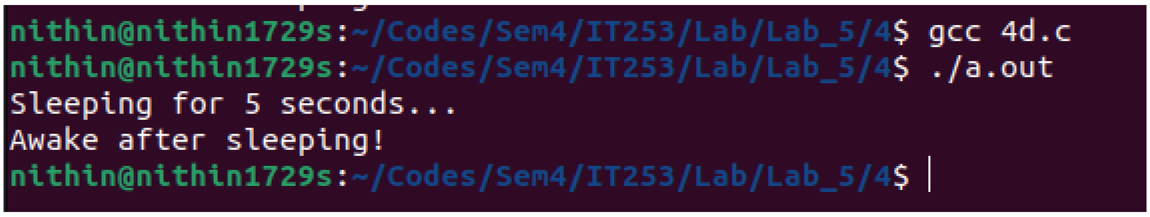
```c
#include <stdio.h>
#include <unistd.h>

int main() {
    printf("Sleeping for 5 seconds...\n");
    sleep(5); // Sleep for 5 seconds
    printf("Awake after sleeping!\n");
    return 0;
}
```

Output



```
nithin@nithin1729s:~/Codes/Sem4/IT253/Lab/Lab_5/4$ gcc 4d.c
nithin@nithin1729s:~/Codes/Sem4/IT253/Lab/Lab_5/4$ ./a.out
Sleeping for 5 seconds...
```

After 5 seconds,



```
nithin@nithin1729s:~/Codes/Sem4/IT253/Lab/Lab_5/4$ gcc 4d.c
nithin@nithin1729s:~/Codes/Sem4/IT253/Lab/Lab_5/4$ ./a.out
Sleeping for 5 seconds...
Awake after sleeping!
nithin@nithin1729s:~/Codes/Sem4/IT253/Lab/Lab_5/4$
```