Name - Khushveer

College Roll no – 24570031

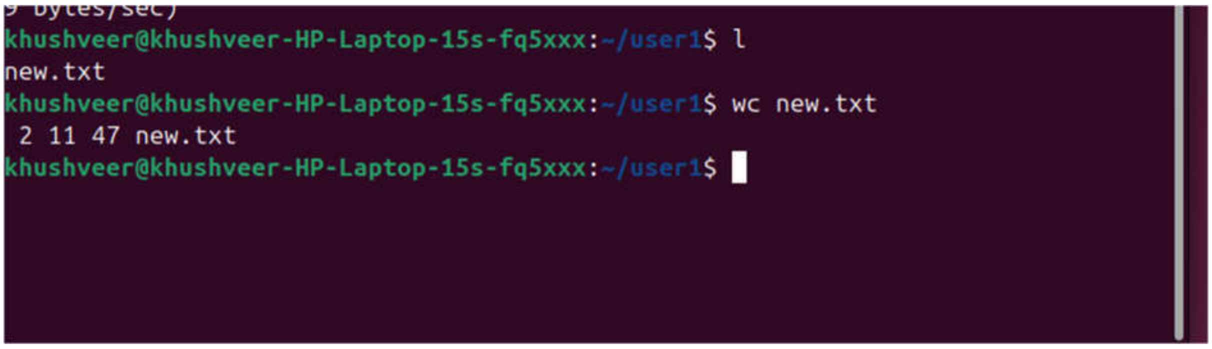
Semester – 3rd

Course - B. Sc (H) Computer Science

Q-1 :- Execute various LINUX commands for:

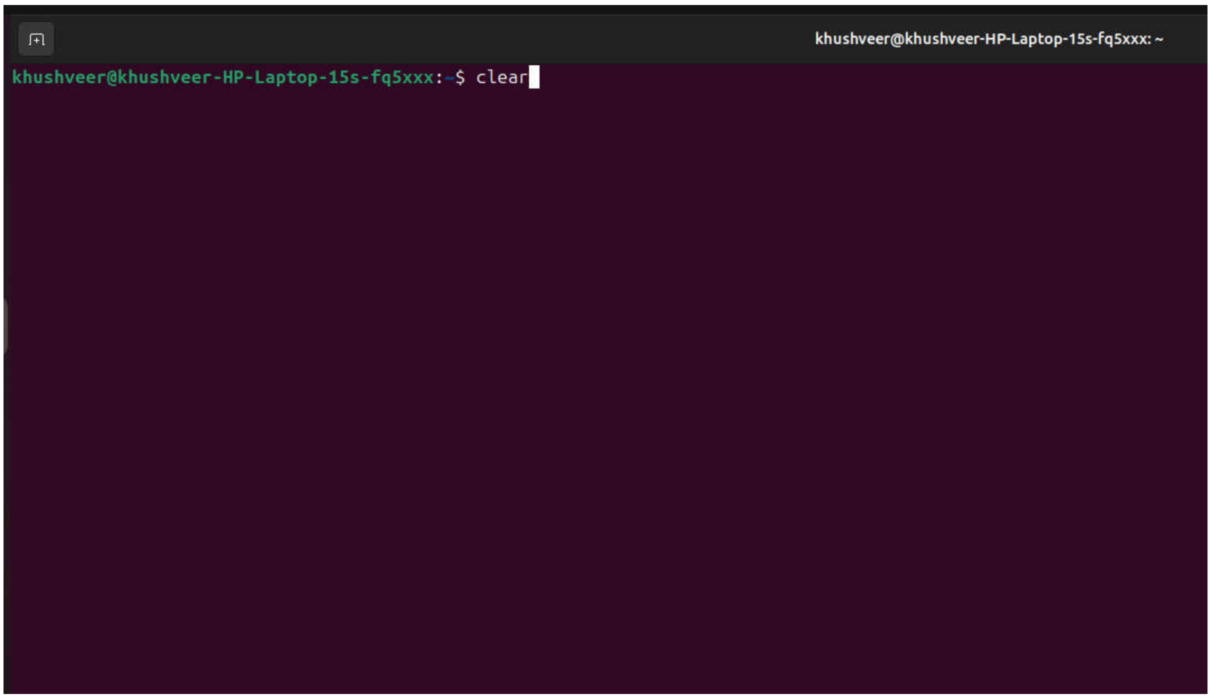
(i) information Maintenance: wc, clear, cal, who, date, pwd.

<u>Wc(word count)</u>

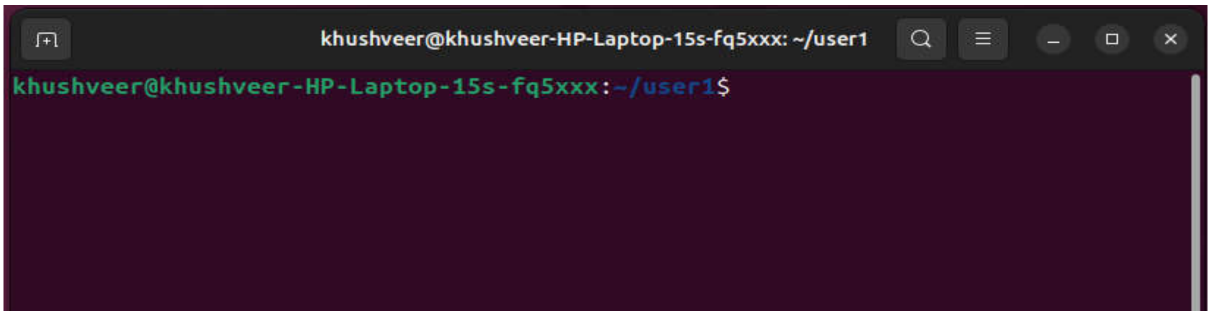


```
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$ l
new.txt
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$ wc new.txt
 2 11 47 new.txt
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$ 
```

Clear

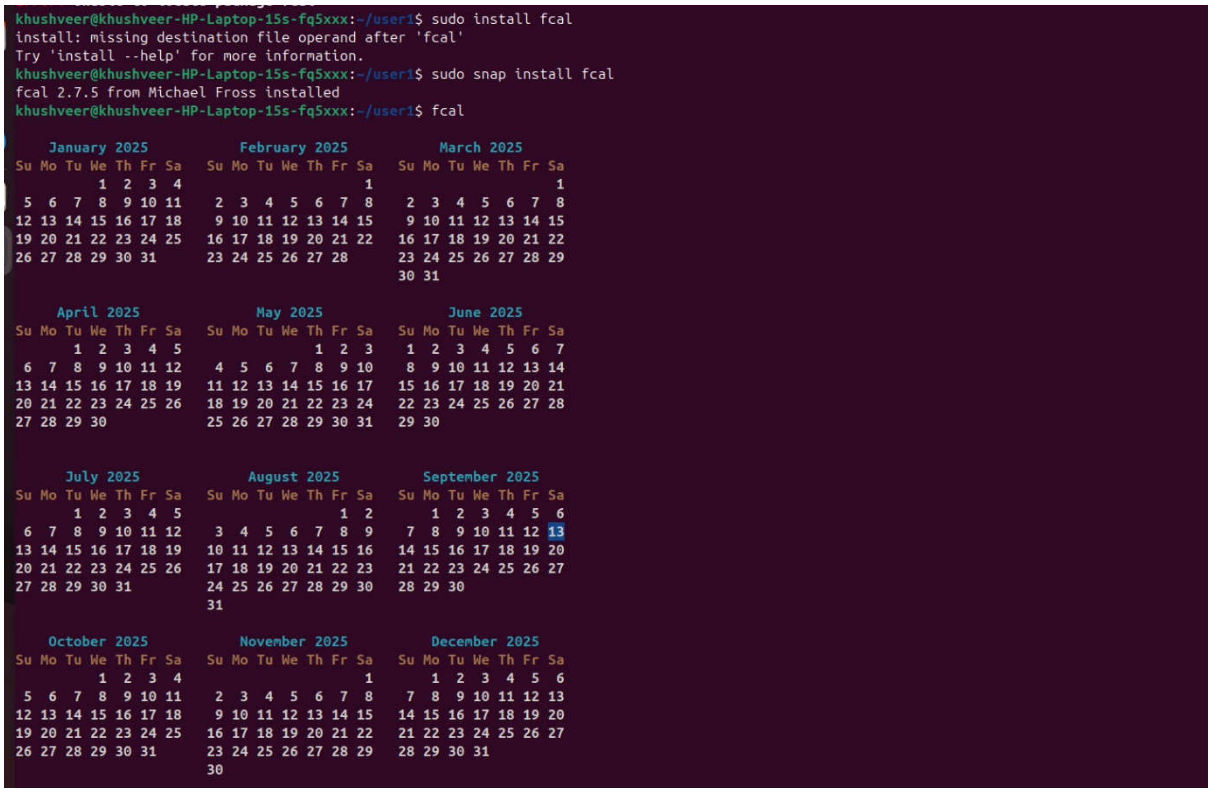

khushveer@khushveer-HP-Laptop-15s-fq5xxx: ~

```
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~$ clear
```


khushveer@khushveer-HP-Laptop-15s-fq5xxx: ~/user1

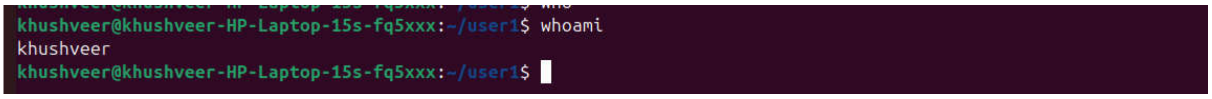
```
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$
```

Cal or fcal(old classic calender or snap-basedd, colorful,modern calender)



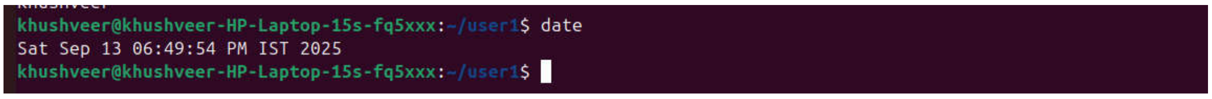
```
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$ sudo install fcal
install: missing destination file operand after 'fcal'
Try 'install --help' for more information.
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$ sudo snap install fcal
fcal 2.7.5 from Michael Fross installed
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$ fcal

    January 2025           February 2025             March 2025
Su Mo Tu We Th Fr Sa    Su Mo Tu We Th Fr Sa    Su Mo Tu We Th Fr Sa
          1  2  3  4                       1                       1
 5  6  7  8  9 10 11     2  3  4  5  6  7  8     2  3  4  5  6  7  8
12 13 14 15 16 17 18     9 10 11 12 13 14 15     9 10 11 12 13 14 15
19 20 21 22 23 24 25    16 17 18 19 20 21 22    16 17 18 19 20 21 22
26 27 28 29 30 31       23 24 25 26 27 28       23 24 25 26 27 28 29
                                                30 31

     April 2025               May 2025               June 2025
Su Mo Tu We Th Fr Sa    Su Mo Tu We Th Fr Sa    Su Mo Tu We Th Fr Sa
       1  2  3  4  5                 1  2  3     1  2  3  4  5  6  7
 6  7  8  9 10 11 12     4  5  6  7  8  9 10     8  9 10 11 12 13 14
13 14 15 16 17 18 19    11 12 13 14 15 16 17    15 16 17 18 19 20 21
20 21 22 23 24 25 26    18 19 20 21 22 23 24    22 23 24 25 26 27 28
27 28 29 30             25 26 27 28 29 30 31    29 30

      July 2025             August 2025            September 2025
Su Mo Tu We Th Fr Sa    Su Mo Tu We Th Fr Sa    Su Mo Tu We Th Fr Sa
       1  2  3  4  5                    1  2        1  2  3  4  5  6
 6  7  8  9 10 11 12     3  4  5  6  7  8  9     7  8  9 10 11 12 13
13 14 15 16 17 18 19    10 11 12 13 14 15 16    14 15 16 17 18 19 20
20 21 22 23 24 25 26    17 18 19 20 21 22 23    21 22 23 24 25 26 27
27 28 29 30 31          24 25 26 27 28 29 30    28 29 30
                        31

    October 2025           November 2025           December 2025
Su Mo Tu We Th Fr Sa    Su Mo Tu We Th Fr Sa    Su Mo Tu We Th Fr Sa
          1  2  3  4                       1        1  2  3  4  5  6
 5  6  7  8  9 10 11     2  3  4  5  6  7  8     7  8  9 10 11 12 13
12 13 14 15 16 17 18     9 10 11 12 13 14 15    14 15 16 17 18 19 20
19 20 21 22 23 24 25    16 17 18 19 20 21 22    21 22 23 24 25 26 27
26 27 28 29 30 31       23 24 25 26 27 28 29    28 29 30 31
                        30
```

Who(current user)



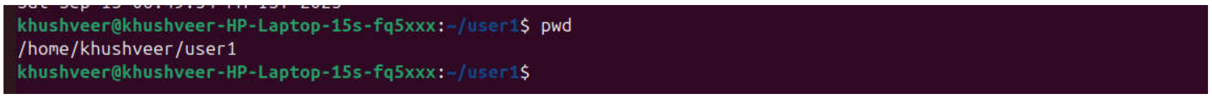
```
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$ whoami
khushveer
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$
```

Date



```
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$ date
Sat Sep 13 06:49:54 PM IST 2025
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$
```

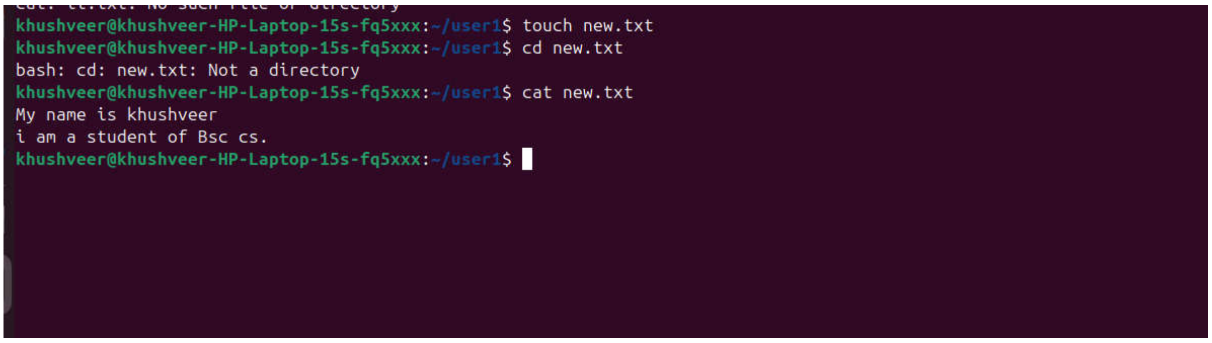
Pwd(print the current directory)



```
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$ pwd
/home/khushveer/user1
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$
```

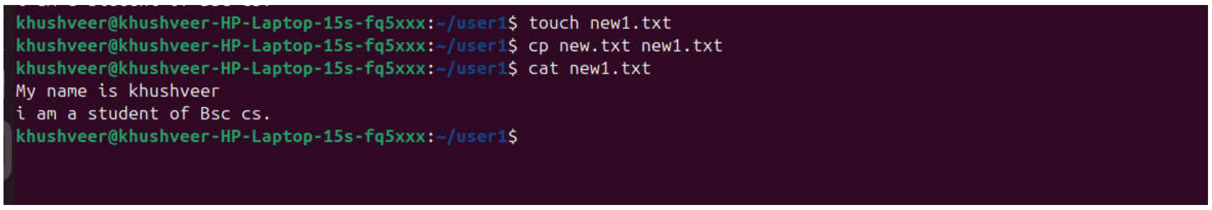
(ii) File Management: cat, cp, rm, mv, cmp, comm, diff, find, grep, awk.

Cat(show content of file)



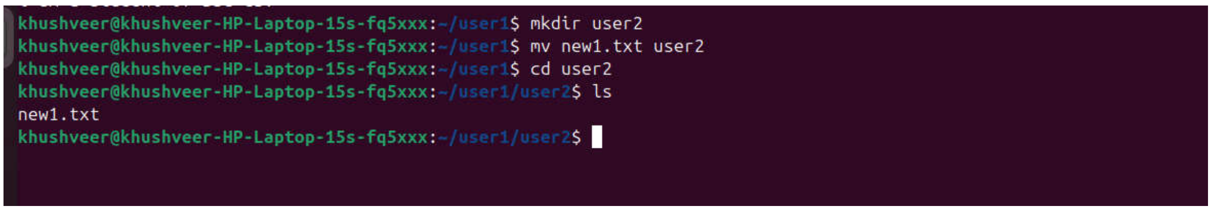
```
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$ touch new.txt
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$ cd new.txt
bash: cd: new.txt: Not a directory
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$ cat new.txt
My name is khushveer
i am a student of Bsc cs.
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$ 
```

Cp(copy files)



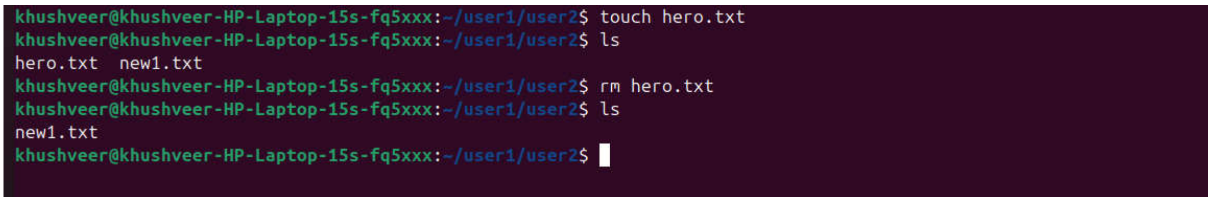
```
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$ touch new1.txt
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$ cp new.txt new1.txt
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$ cat new1.txt
My name is khushveer
i am a student of Bsc cs.
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$
```

Mv(move file)

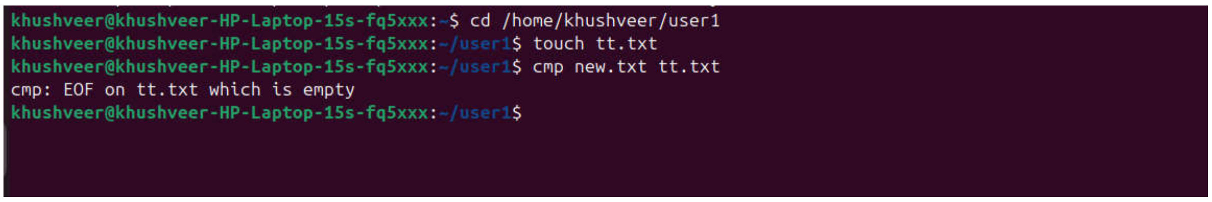


```
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$ mkdir user2
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$ mv new1.txt user2
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$ cd user2
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1/user2$ ls
new1.txt
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1/user2$ 
```

Rm(delete file)

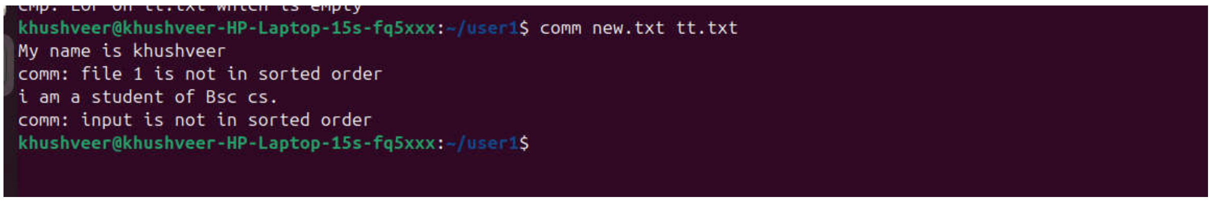


```
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1/user2$ touch hero.txt
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1/user2$ ls
hero.txt  new1.txt
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1/user2$ rm hero.txt
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1/user2$ ls
new1.txt
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1/user2$ 
```



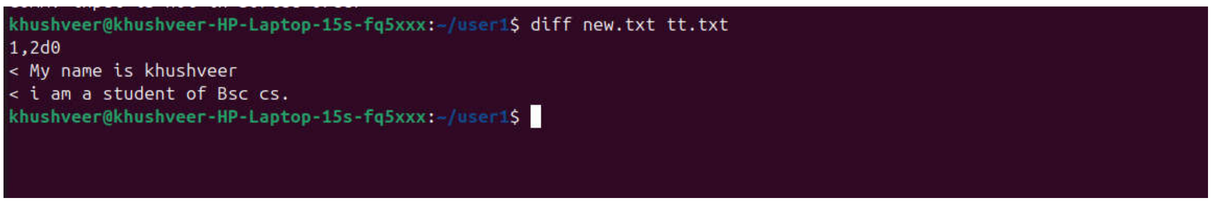
```
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~$ cd /home/khushveer/user1
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$ touch tt.txt
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$ cmp new.txt tt.txt
cmp: EOF on tt.txt which is empty
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$
```

Comm(compare two sorted file line by line)



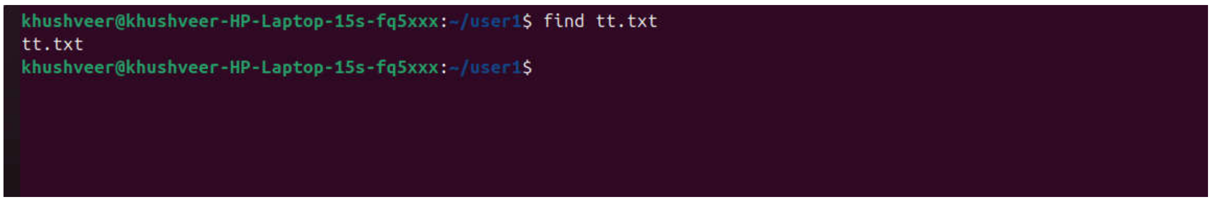
```
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$ comm new.txt tt.txt
My name is khushveer
comm: file 1 is not in sorted order
i am a student of Bsc cs.
comm: input is not in sorted order
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$
```

Diff(diffrence between two file)



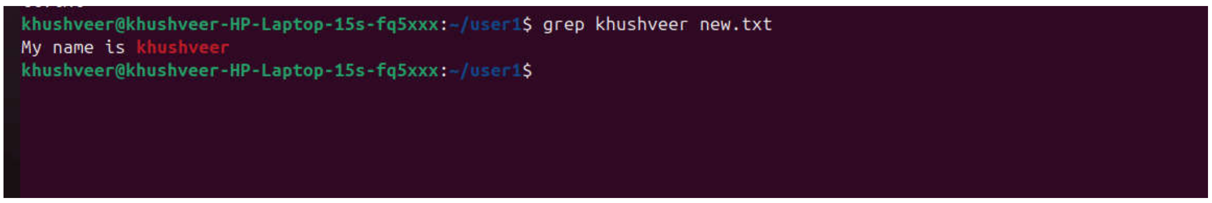
```
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$ diff new.txt tt.txt
1,2d0
< My name is khushveer
< i am a student of Bsc cs.
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$
```

Find(find file)



```
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$ find tt.txt
tt.txt
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$
```

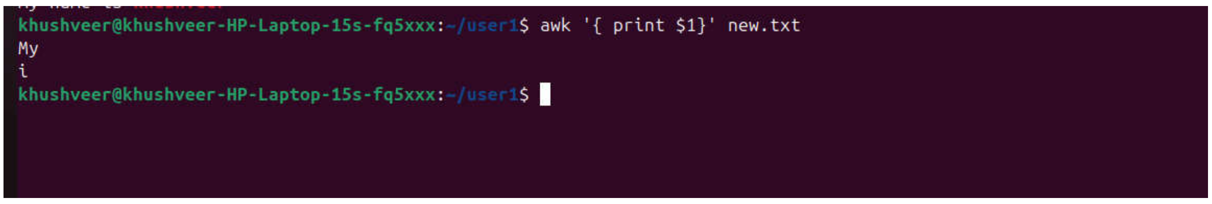
Grep(search for text patterns)



```
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$ grep khushveer new.txt
My name is khushveer
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$
```

Awk(extract,format and proccess texts)

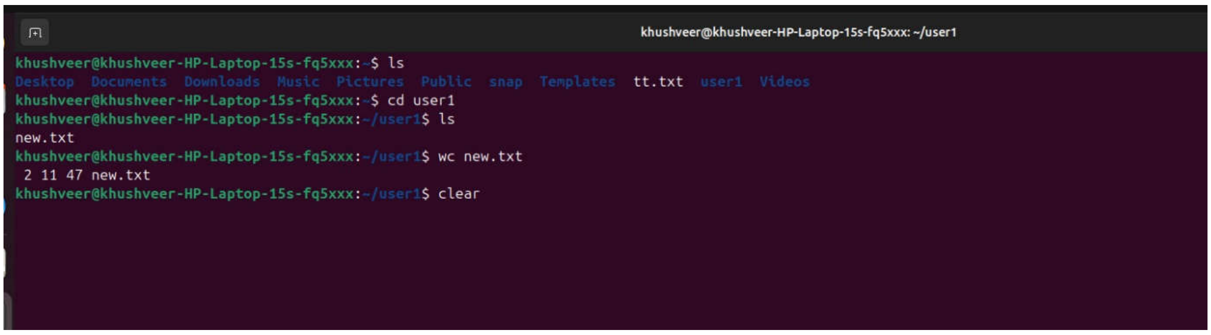
awk is a programming language for text processing. It works line by line and splits each line into fields (columns), usually separated by spaces



```
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$ awk '{ print $1}' new.txt
My
i
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$
```

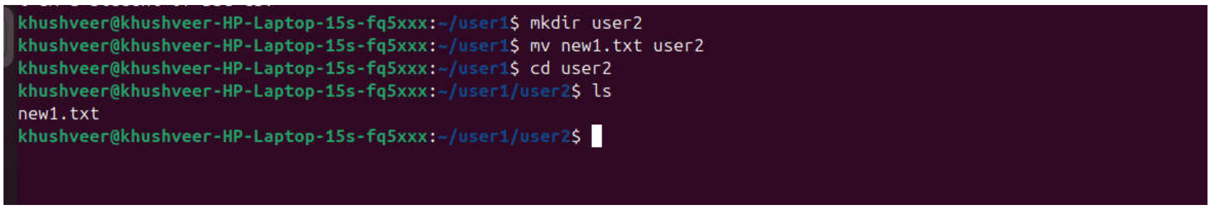
(Iii). Directory Management: cd, mkdir, rmdir, ls

Cd (change directory)



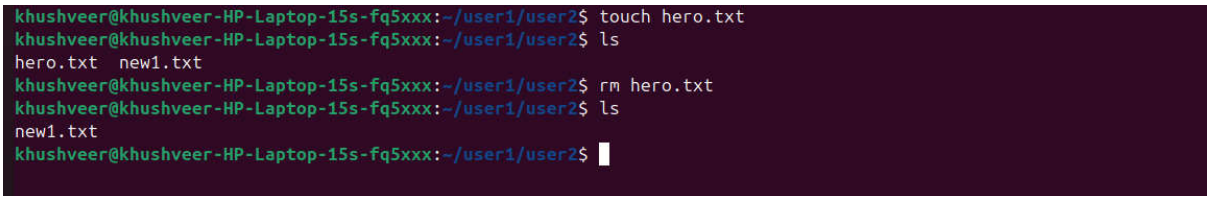
```
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  snap  Templates  tt.txt  user1  Videos
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~$ cd user1
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$ ls
new.txt
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$ wc new.txt
 2 11 47 new.txt
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$ clear
```

Mkdir(make directory)



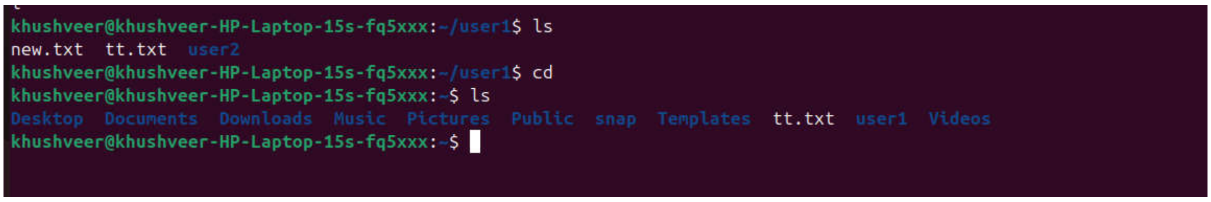
```
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$ mkdir user2
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$ mv new1.txt user2
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$ cd user2
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1/user2$ ls
new1.txt
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1/user2$ 
```

Rmdir(remove directory)



```
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1/user2$ touch hero.txt
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1/user2$ ls
hero.txt  new1.txt
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1/user2$ rm hero.txt
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1/user2$ ls
new1.txt
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1/user2$ 
```

Ls(list directories/files)



```
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$ ls
new.txt  tt.txt  user2
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$ cd
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  snap  Templates  tt.txt  user1  Videos
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~$ 
```

Q – 2 - Execute various LINUX commands for

i. Communication: Input-output redirection, Pipe

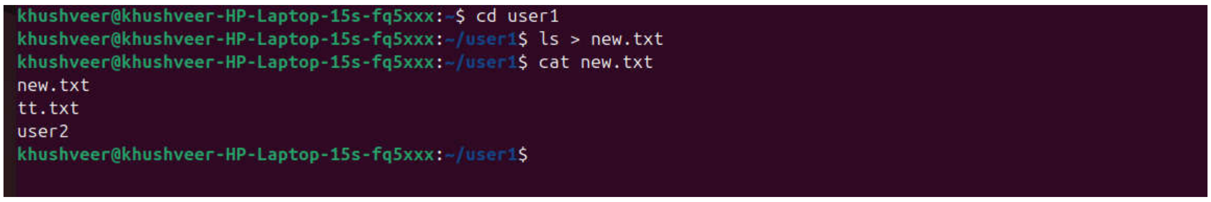
1. Input/Output Redirection

Every command in Linux works with **three standard streams**:

- **stdin (0)** → standard input (keyboard by default)
- **stdout (1)** → standard output (screen by default)

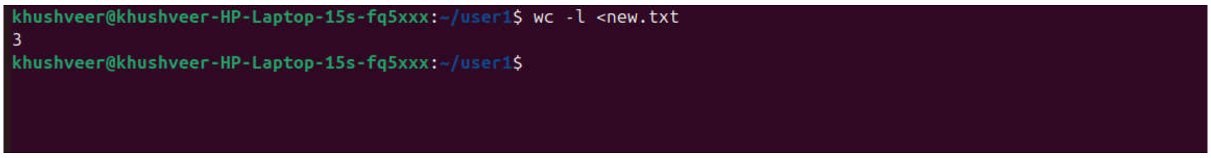
- **stderr (2)** → standard error (error messages)

-->Output redirection



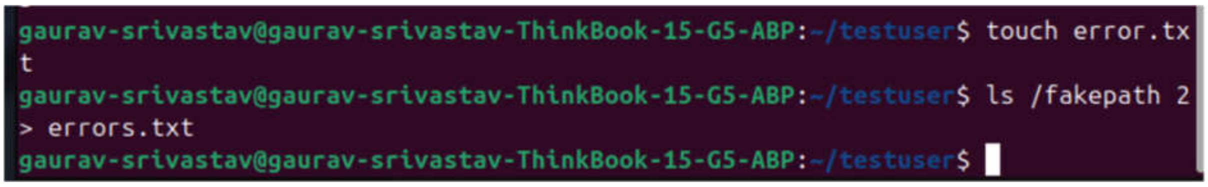
```
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~$ cd user1
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$ ls > new.txt
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$ cat new.txt
new.txt
tt.txt
user2
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$
```

--> input redirection



```
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$ wc -l <new.txt
3
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$
```

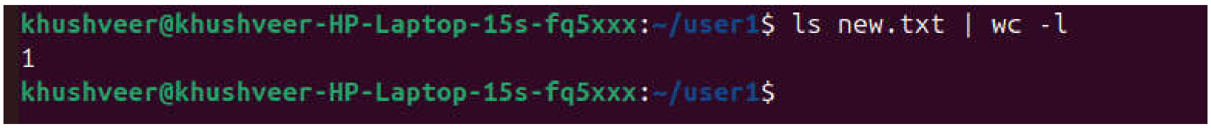
--> error redirection



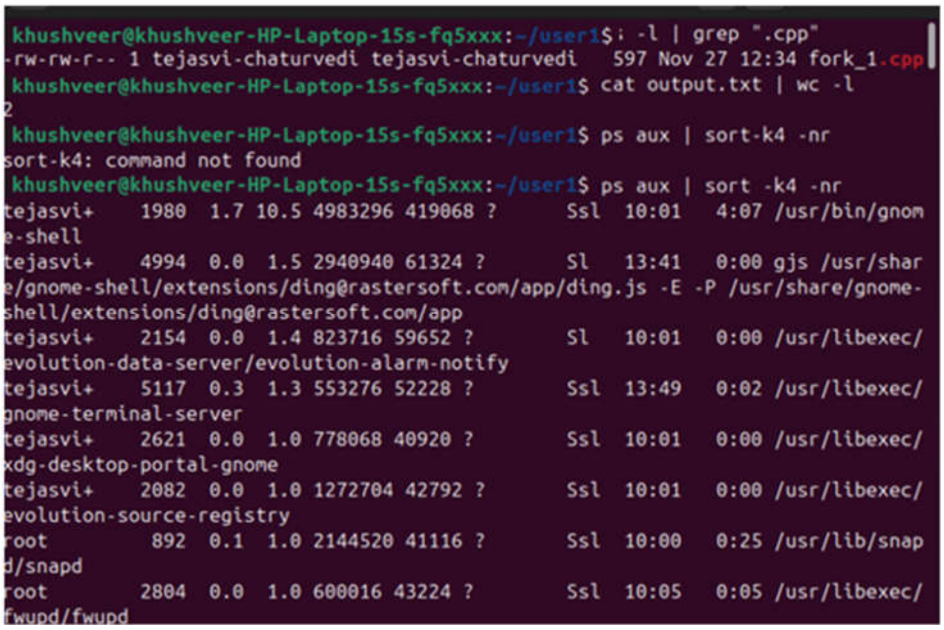
```
gaurav-srivastav@gaurav-srivastav-ThinkBook-15-G5-ABP:~/testuser$ touch error.tx
t
gaurav-srivastav@gaurav-srivastav-ThinkBook-15-G5-ABP:~/testuser$ ls /fakepath 2
> errors.txt
gaurav-srivastav@gaurav-srivastav-ThinkBook-15-G5-ABP:~/testuser$
```

Pipes (|)

A **pipe** connects the **stdout** of one command to the **stdin** of another.
This is **command-to-command communication**.

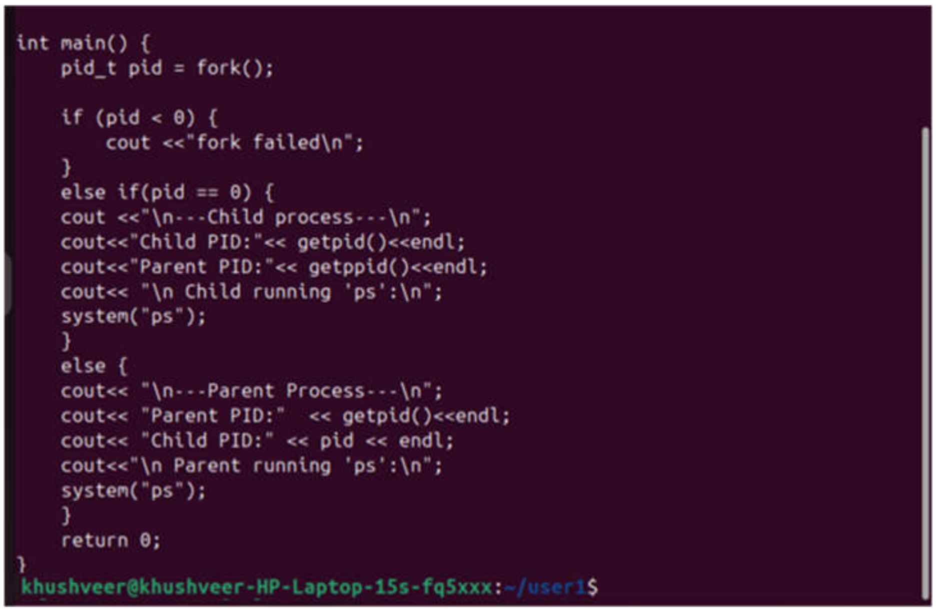


```
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$ ls new.txt | wc -l
1
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$
```

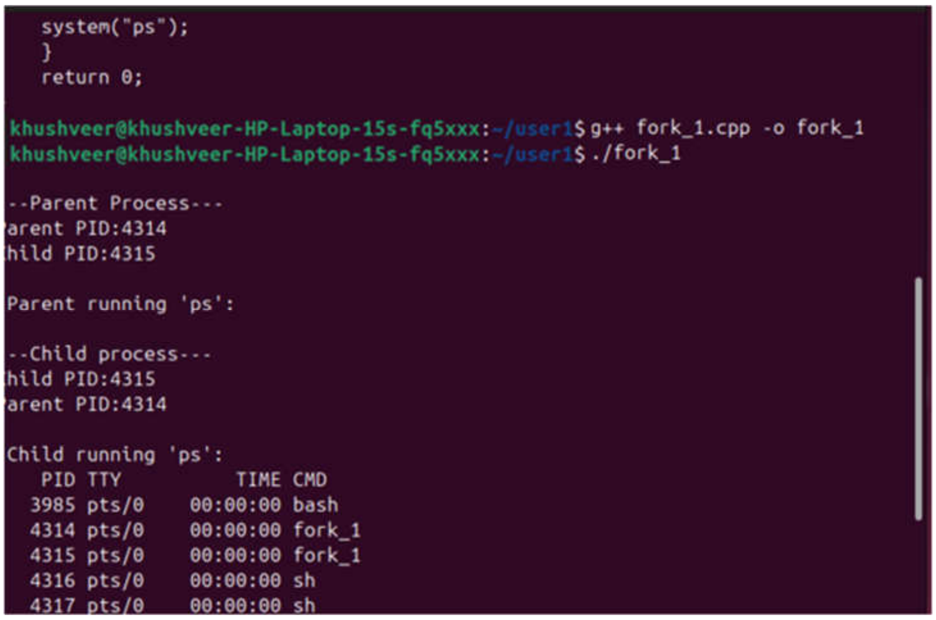


```
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$ls -l | grep ".cpp"
-rw-rw-r-- 1 tejasvi-chaturvedi tejasvi-chaturvedi   597 Nov 27 12:34 fork_1.cpp
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$ cat output.txt | wc -l
2
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$ ps aux | sort-k4 -nr
sort-k4: command not found
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$ ps aux | sort -k4 -nr
tejasvi+    1980  1.7 10.5 4983296 419068 ?      Ssl  10:01   4:07 /usr/bin/gnom
e-shell
tejasvi+    4994  0.0  1.5 2940940 61324 ?       Sl   13:41   0:00 gjs /usr/shar
e/gnome-shell/extensions/ding@rastersoft.com/app/ding.js -E -P /usr/share/gnome-
shell/extensions/ding@rastersoft.com/app
tejasvi+    2154  0.0  1.4 823716 59652 ?        Sl   10:01   0:00 /usr/libexec/
evolution-data-server/evolution-alarm-notify
tejasvi+    5117  0.3  1.3 553276 52228 ?        Ssl  13:49   0:02 /usr/libexec/
gnome-terminal-server
tejasvi+    2621  0.0  1.0 778068 40920 ?        Ssl  10:01   0:00 /usr/libexec/
xdg-desktop-portal-gnome
tejasvi+    2082  0.0  1.0 1272704 42792 ?       Ssl  10:01   0:00 /usr/libexec/
evolution-source-registry
root         892  0.1  1.0 2144520 41116 ?       Ssl  10:00   0:25 /usr/lib/snap
d/snapd
root        2804  0.0  1.0 600016 43224 ?        Ssl  10:05   0:05 /usr/libexec/
fwupd/fwupd
```

ii. Process Control: fork, getpid, ps, kill, sleep

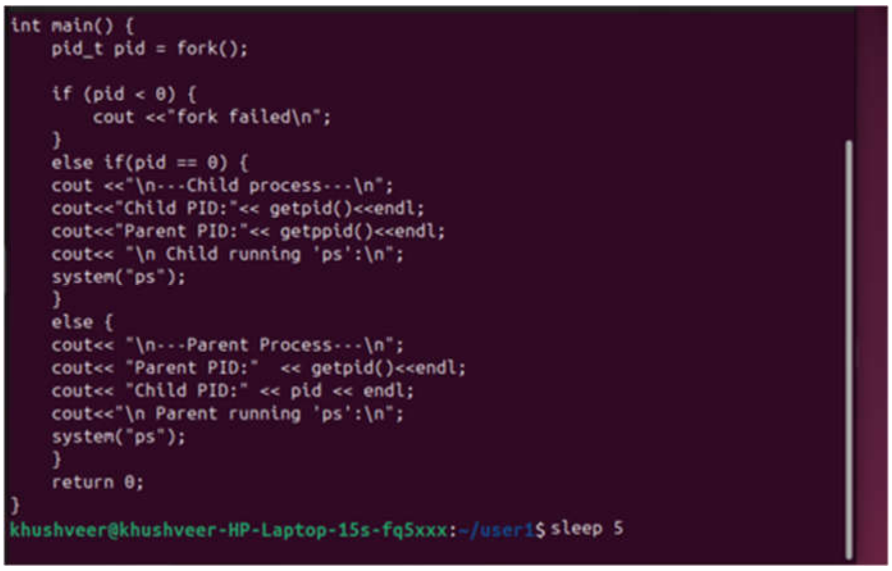


```
int main() {
    pid_t pid = fork();

    if (pid < 0) {
        cout <<"fork failed\n";
    }
    else if(pid == 0) {
    cout <<"\n---Child process---\n";
    cout<<"Child PID:"<< getpid()<<endl;
    cout<<"Parent PID:"<< getppid()<<endl;
    cout<< "\n Child running 'ps':\n";
    system("ps");
    }
    else {
    cout<< "\n---Parent Process---\n";
    cout<< "Parent PID:"  << getpid()<<endl;
    cout<< "Child PID:" << pid << endl;
    cout<<"\n Parent running 'ps':\n";
    system("ps");
    }
    return 0;
}
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$
```



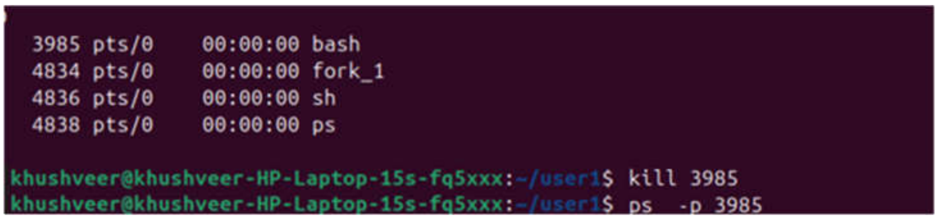
```
   system("ps");
   }
   return 0;

khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$ g++ fork_1.cpp -o fork_1
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$ ./fork_1

--Parent Process---
arent PID:4314
hild PID:4315

Parent running 'ps':

--Child process---
hild PID:4315
arent PID:4314

Child running 'ps':
   PID TTY          TIME CMD
  3985 pts/0    00:00:00 bash
  4314 pts/0    00:00:00 fork_1
  4315 pts/0    00:00:00 fork_1
  4316 pts/0    00:00:00 sh
  4317 pts/0    00:00:00 sh
```

Sleep



```
int main() {
    pid_t pid = fork();

    if (pid < 0) {
        cout <<"fork failed\n";
    }
    else if(pid == 0) {
    cout <<"\n---Child process---\n";
    cout<<"Child PID:"<< getpid()<<endl;
    cout<<"Parent PID:"<< getppid()<<endl;
    cout<< "\n Child running 'ps':\n";
    system("ps");
    }
    else {
    cout<< "\n---Parent Process---\n";
    cout<< "Parent PID:"  << getpid()<<endl;
    cout<< "Child PID:" << pid << endl;
    cout<<"\n Parent running 'ps':\n";
    system("ps");
    }
    return 0;
}
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$ sleep 5
```

Kill

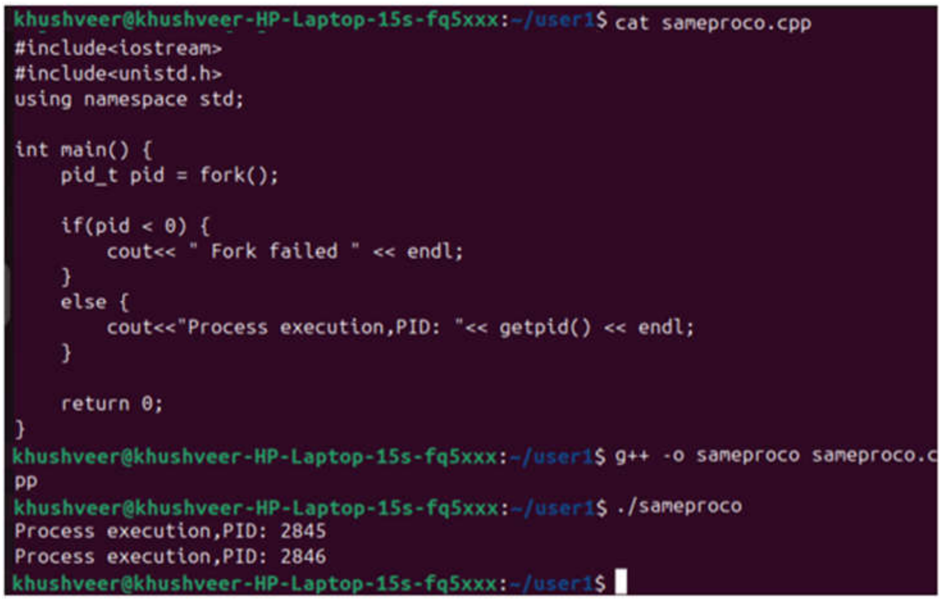


```
  3985 pts/0    00:00:00 bash
  4834 pts/0    00:00:00 fork_1
  4836 pts/0    00:00:00 sh
  4838 pts/0    00:00:00 ps

khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$ kill 3985
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$ ps  -p 3985
```

ii. Management: chmod, chown, chgrp

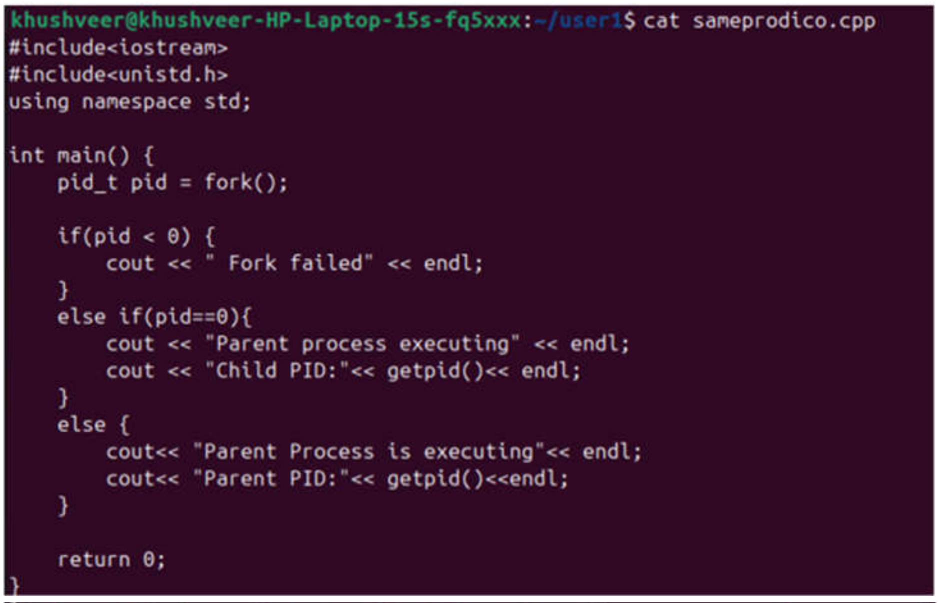
3. Write a programme (using fork() and/or exec() commands) where parent and child execute:

i. same program, same code.

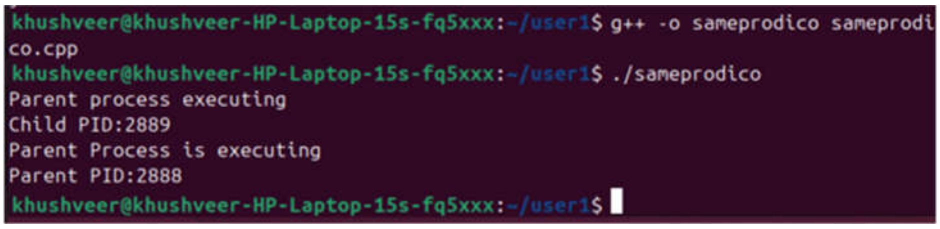


```
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$ cat sameproco.cpp
#include<iostream>
#include<unistd.h>
using namespace std;

int main() {
    pid_t pid = fork();

    if(pid < 0) {
        cout<< " Fork failed " << endl;
    }
    else {
        cout<<"Process execution,PID: "<< getpid() << endl;
    }

    return 0;
}
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$ g++ -o sameproco sameproco.c
pp
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$ ./sameproco
Process execution,PID: 2845
Process execution,PID: 2846
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$ 
```

iii. same program, different code.

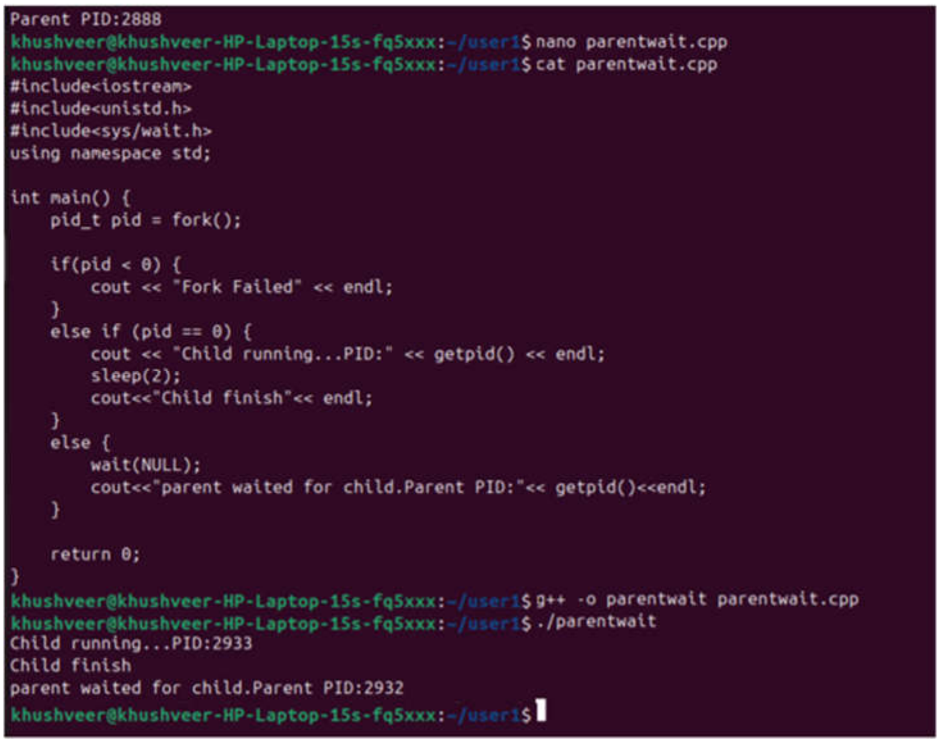


```
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$ cat sameprodico.cpp
#include<iostream>
#include<unistd.h>
using namespace std;

int main() {
    pid_t pid = fork();

    if(pid < 0) {
        cout << " Fork failed" << endl;
    }
    else if(pid==0){
        cout << "Parent process executing" << endl;
        cout << "Child PID:"<< getpid()<< endl;
    }
    else {
        cout<< "Parent Process is executing"<< endl;
        cout<< "Parent PID:"<< getpid()<<endl;
    }

    return 0;
}
```



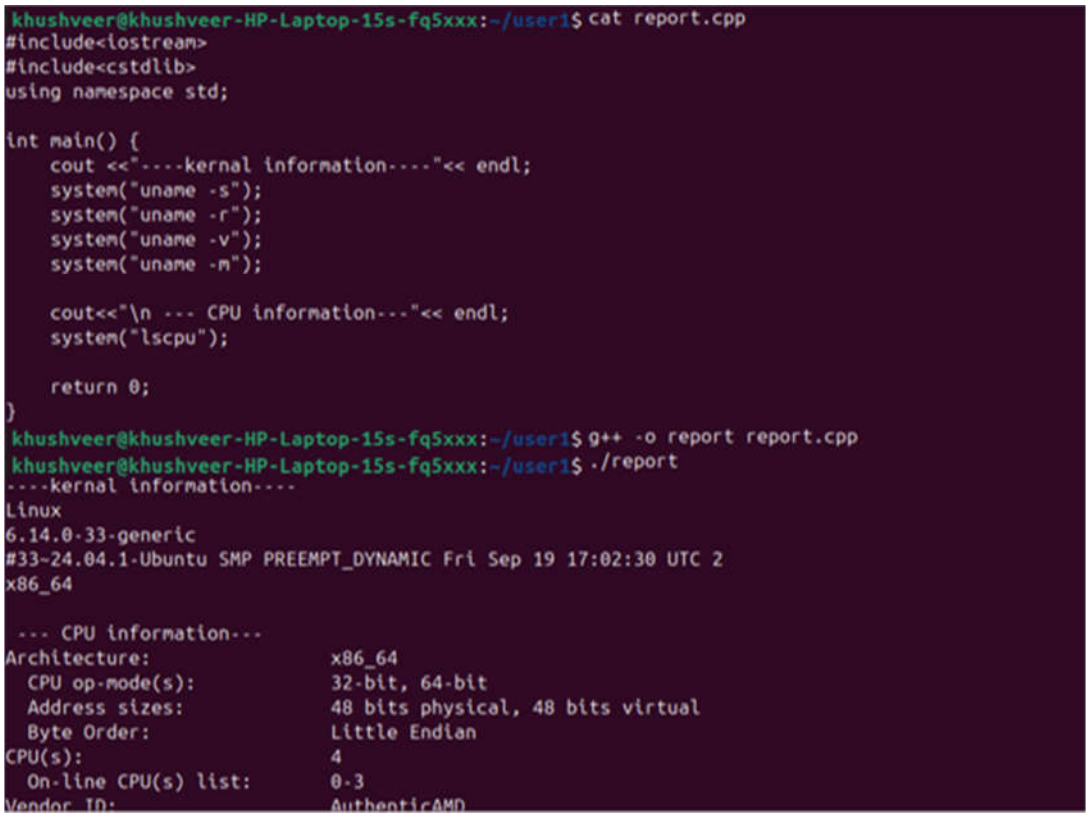
```
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$ g++ -o sameprodico sameprodi
co.cpp
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$ ./sameprodico
Parent process executing
Child PID:2889
Parent Process is executing
Parent PID:2888
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$
```

iii. Before terminating, the parent waits for the child to finish its task.



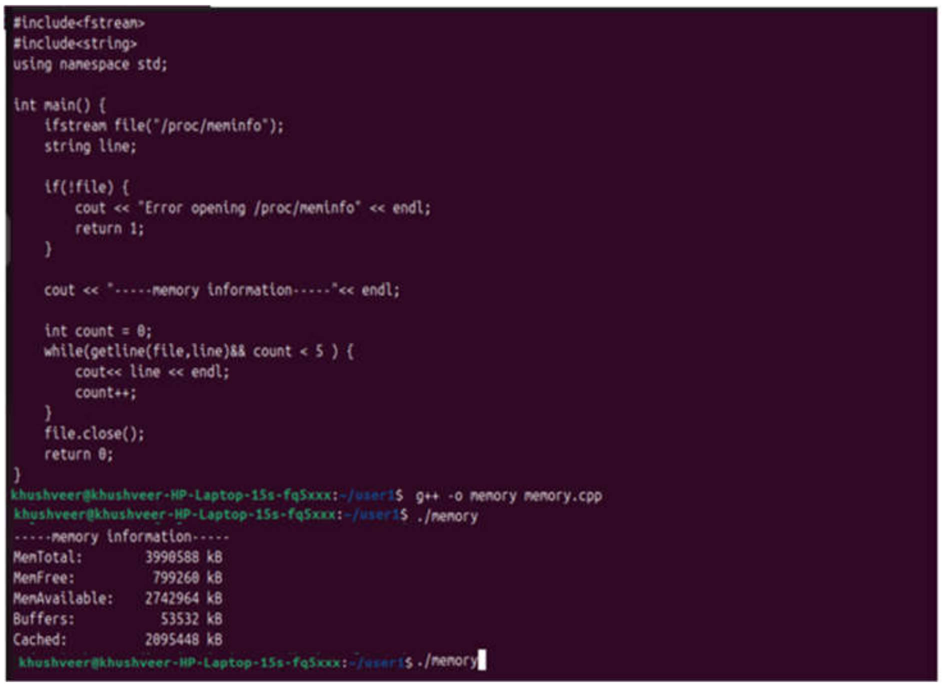
```
Parent PID:2888
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$ nano parentwait.cpp
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$ cat parentwait.cpp
#include<iostream>
#include<unistd.h>
#include<sys/wait.h>
using namespace std;

int main() {
    pid_t pid = fork();

    if(pid < 0) {
        cout << "Fork Failed" << endl;
    }
    else if (pid == 0) {
        cout << "Child running...PID:" << getpid() << endl;
        sleep(2);
        cout<<"Child finish"<< endl;
    }
    else {
        wait(NULL);
        cout<<"parent waited for child.Parent PID:"<< getpid()<<endl;
    }

    return 0;
}
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$ g++ -o parentwait parentwait.cpp
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$ ./parentwait
Child running...PID:2933
Child finish
parent waited for child.Parent PID:2932
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$
```

4. Write a program to report behaviour of Linux kernel including kernel version, CPU type and model. (CPU information)



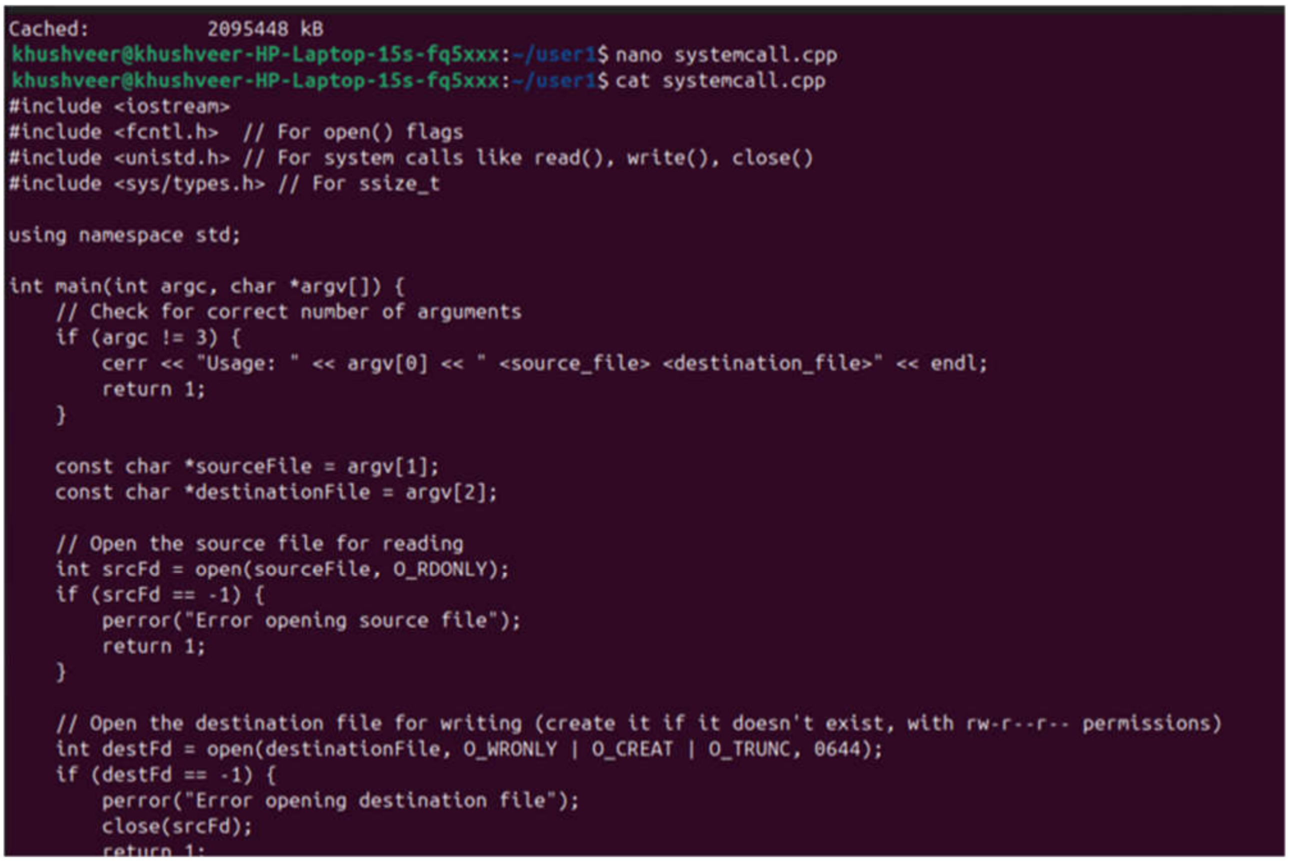
```
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$ cat report.cpp
#include<iostream>
#include<cstdlib>
using namespace std;

int main() {
    cout <<"----kernal information----"<< endl;
    system("uname -s");
    system("uname -r");
    system("uname -v");
    system("uname -m");

    cout<<"\n --- CPU information---"<< endl;
    system("lscpu");

    return 0;
}
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$ g++ -o report report.cpp
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$ ./report
----kernal information----
Linux
6.14.0-33-generic
#33~24.04.1-Ubuntu SMP PREEMPT_DYNAMIC Fri Sep 19 17:02:30 UTC 2
x86_64

 --- CPU information---
Architecture:             x86_64
  CPU op-mode(s):         32-bit, 64-bit
  Address sizes:          48 bits physical, 48 bits virtual
  Byte Order:             Little Endian
CPU(s):                   4
  On-line CPU(s) list:    0-3
Vendor ID:                AuthenticAMD
```

5. Write a program to report behaviour of Linux kernel including configured memory, amount of free and used memory. (Memory information)

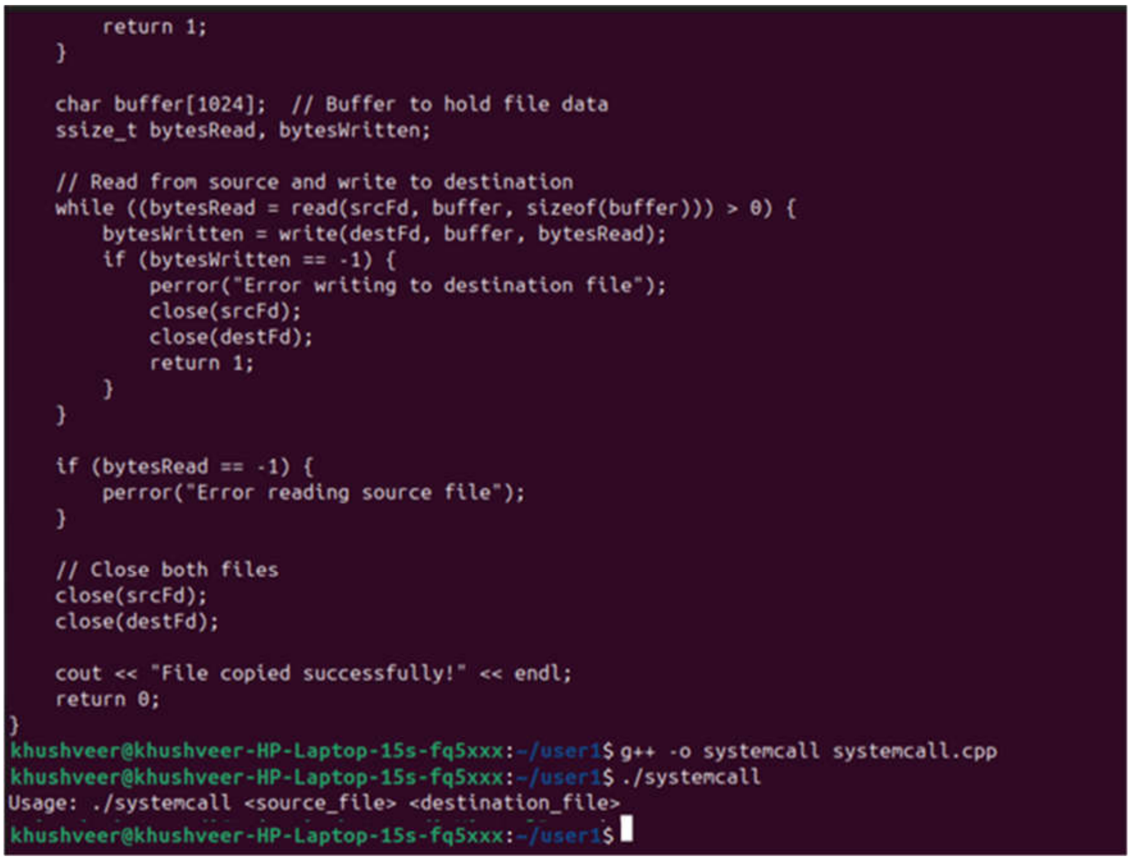


```
#include<fstream>
#include<string>
using namespace std;

int main() {
    ifstream file("/proc/meminfo");
    string line;

    if(!file) {
        cout << "Error opening /proc/meminfo" << endl;
        return 1;
    }

    cout << "-----memory information-----"<< endl;

    int count = 0;
    while(getline(file,line)&& count < 5 ) {
        cout<< line << endl;
        count++;
    }
    file.close();
    return 0;
}
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$  g++ -o memory memory.cpp
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$ ./memory
-----memory information-----
MemTotal:        3990588 kB
MemFree:          799260 kB
MemAvailable:    2742964 kB
Buffers:           53532 kB
Cached:          2895448 kB
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$ ./memory
```

6. write a program to copy files using system calls.

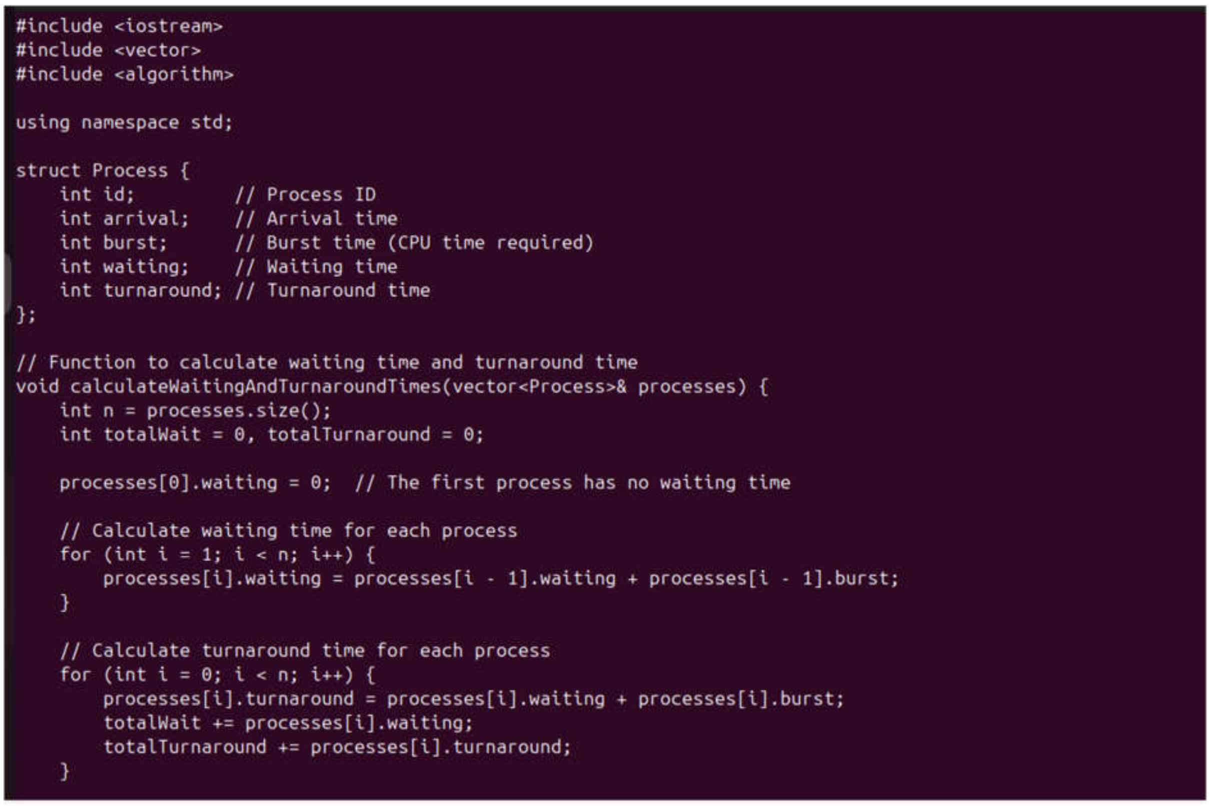


```
Cached:          2095448 kB
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$ nano systemcall.cpp
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$ cat systemcall.cpp
#include <iostream>
#include <fcntl.h>  // For open() flags
#include <unistd.h> // For system calls like read(), write(), close()
#include <sys/types.h> // For ssize_t

using namespace std;

int main(int argc, char *argv[]) {
    // Check for correct number of arguments
    if (argc != 3) {
        cerr << "Usage: " << argv[0] << " <source_file> <destination_file>" << endl;
        return 1;
    }

    const char *sourceFile = argv[1];
    const char *destinationFile = argv[2];

    // Open the source file for reading
    int srcFd = open(sourceFile, O_RDONLY);
    if (srcFd == -1) {
        perror("Error opening source file");
        return 1;
    }

    // Open the destination file for writing (create it if it doesn't exist, with rw-r--r-- permissions)
    int destFd = open(destinationFile, O_WRONLY | O_CREAT | O_TRUNC, 0644);
    if (destFd == -1) {
        perror("Error opening destination file");
        close(srcFd);
```

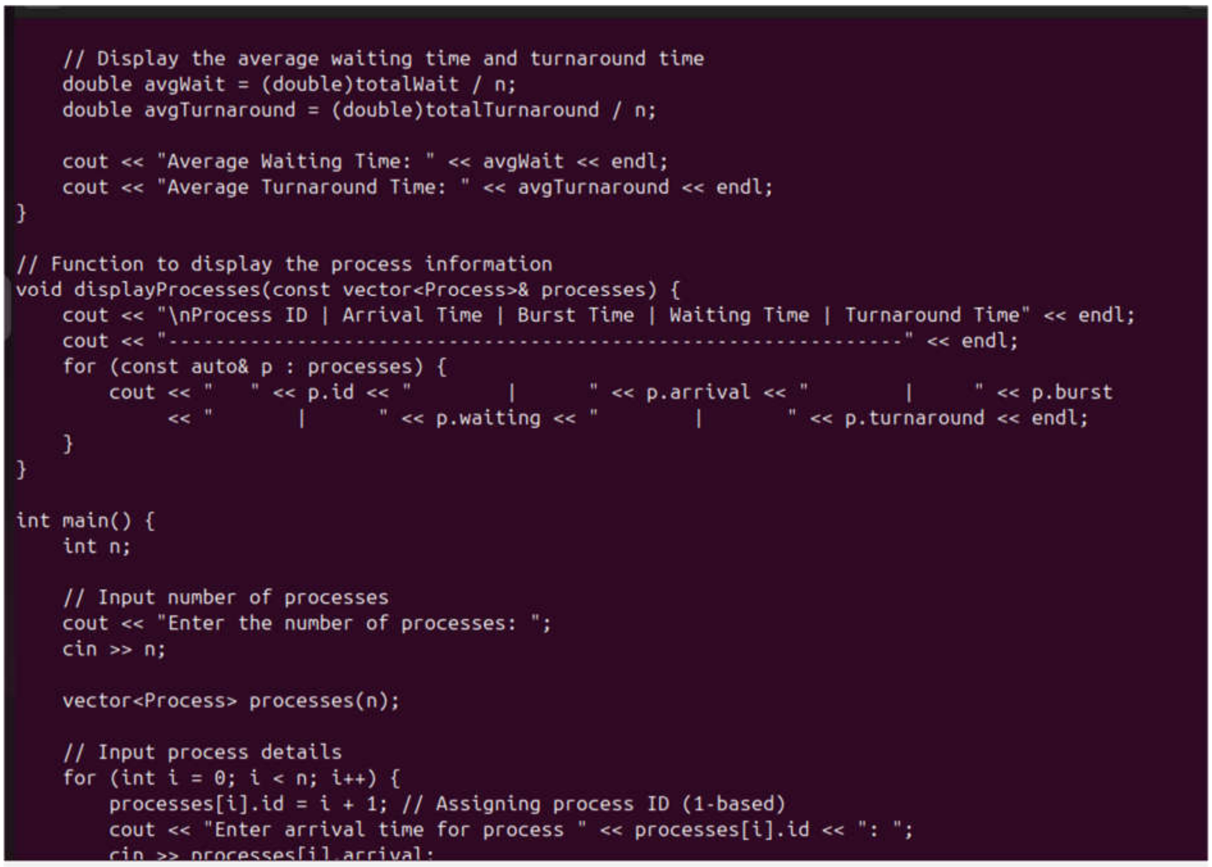


```
        return 1;
    }

    char buffer[1024];  // Buffer to hold file data
    ssize_t bytesRead, bytesWritten;

    // Read from source and write to destination
    while ((bytesRead = read(srcFd, buffer, sizeof(buffer))) > 0) {
        bytesWritten = write(destFd, buffer, bytesRead);
        if (bytesWritten == -1) {
            perror("Error writing to destination file");
            close(srcFd);
            close(destFd);
            return 1;
        }
    }

    if (bytesRead == -1) {
        perror("Error reading source file");
    }

    // Close both files
    close(srcFd);
    close(destFd);

    cout << "File copied successfully!" << endl;
    return 0;
}
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$ g++ -o systemcall systemcall.cpp
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$ ./systemcall
Usage: ./systemcall <source_file> <destination_file>
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$
```

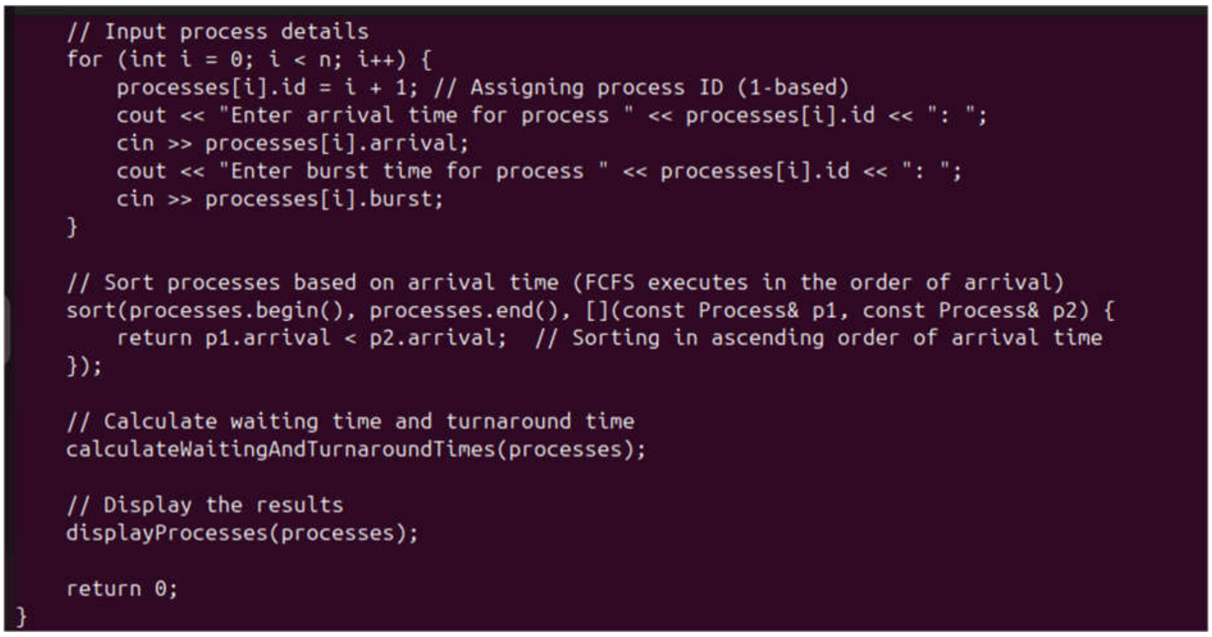
7. Write a program to implement first-come, first-served FCFS scheduling algorithm in CPP code.



```cpp
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

struct Process {
    int id;          // Process ID
    int arrival;     // Arrival time
    int burst;       // Burst time (CPU time required)
    int waiting;     // Waiting time
    int turnaround; // Turnaround time
};

// Function to calculate waiting time and turnaround time
void calculateWaitingAndTurnaroundTimes(vector<Process>& processes) {
    int n = processes.size();
    int totalWait = 0, totalTurnaround = 0;

    processes[0].waiting = 0;  // The first process has no waiting time

    // Calculate waiting time for each process
    for (int i = 1; i < n; i++) {
        processes[i].waiting = processes[i - 1].waiting + processes[i - 1].burst;
    }

    // Calculate turnaround time for each process
    for (int i = 0; i < n; i++) {
        processes[i].turnaround = processes[i].waiting + processes[i].burst;
        totalWait += processes[i].waiting;
        totalTurnaround += processes[i].turnaround;
    }
```

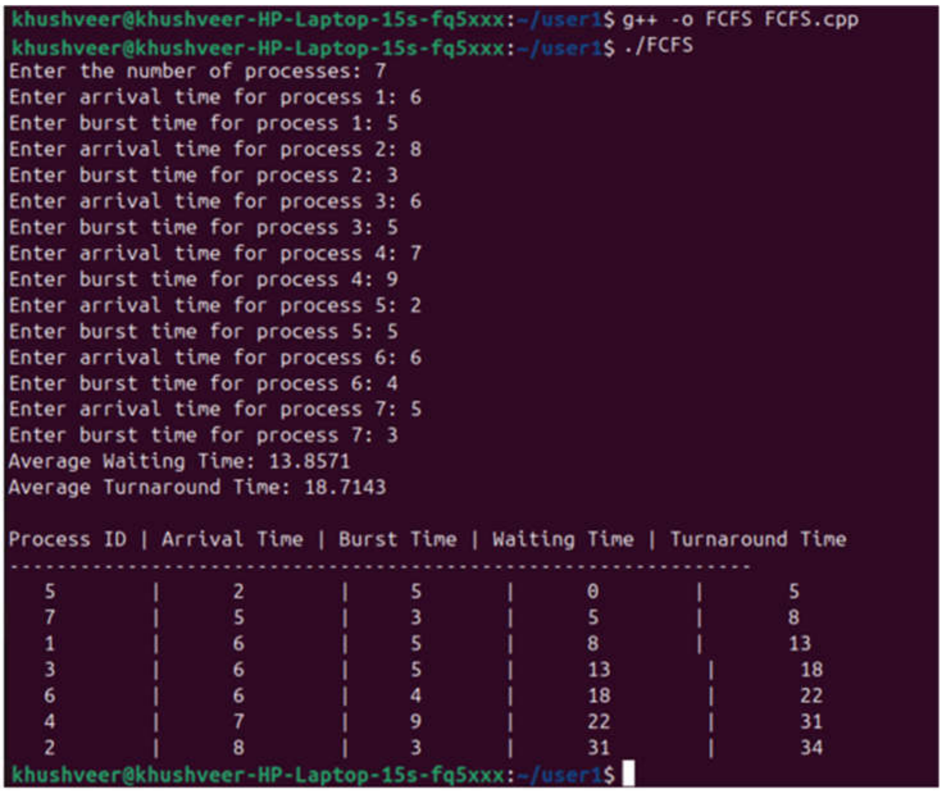


```cpp

    // Display the average waiting time and turnaround time
    double avgWait = (double)totalWait / n;
    double avgTurnaround = (double)totalTurnaround / n;

    cout << "Average Waiting Time: " << avgWait << endl;
    cout << "Average Turnaround Time: " << avgTurnaround << endl;
}

// Function to display the process information
void displayProcesses(const vector<Process>& processes) {
    cout << "\nProcess ID | Arrival Time | Burst Time | Waiting Time | Turnaround Time" << endl;
    cout << "---------------------------------------------------------------" << endl;
    for (const auto& p : processes) {
        cout << "    " << p.id << "          |       " << p.arrival << "          |      " << p.burst
             << "        |       " << p.waiting << "          |        " << p.turnaround << endl;
    }
}

int main() {
    int n;

    // Input number of processes
    cout << "Enter the number of processes: ";
    cin >> n;

    vector<Process> processes(n);

    // Input process details
    for (int i = 0; i < n; i++) {
        processes[i].id = i + 1; // Assigning process ID (1-based)
        cout << "Enter arrival time for process " << processes[i].id << ": ";
        cin >> processes[i].arrival;
```



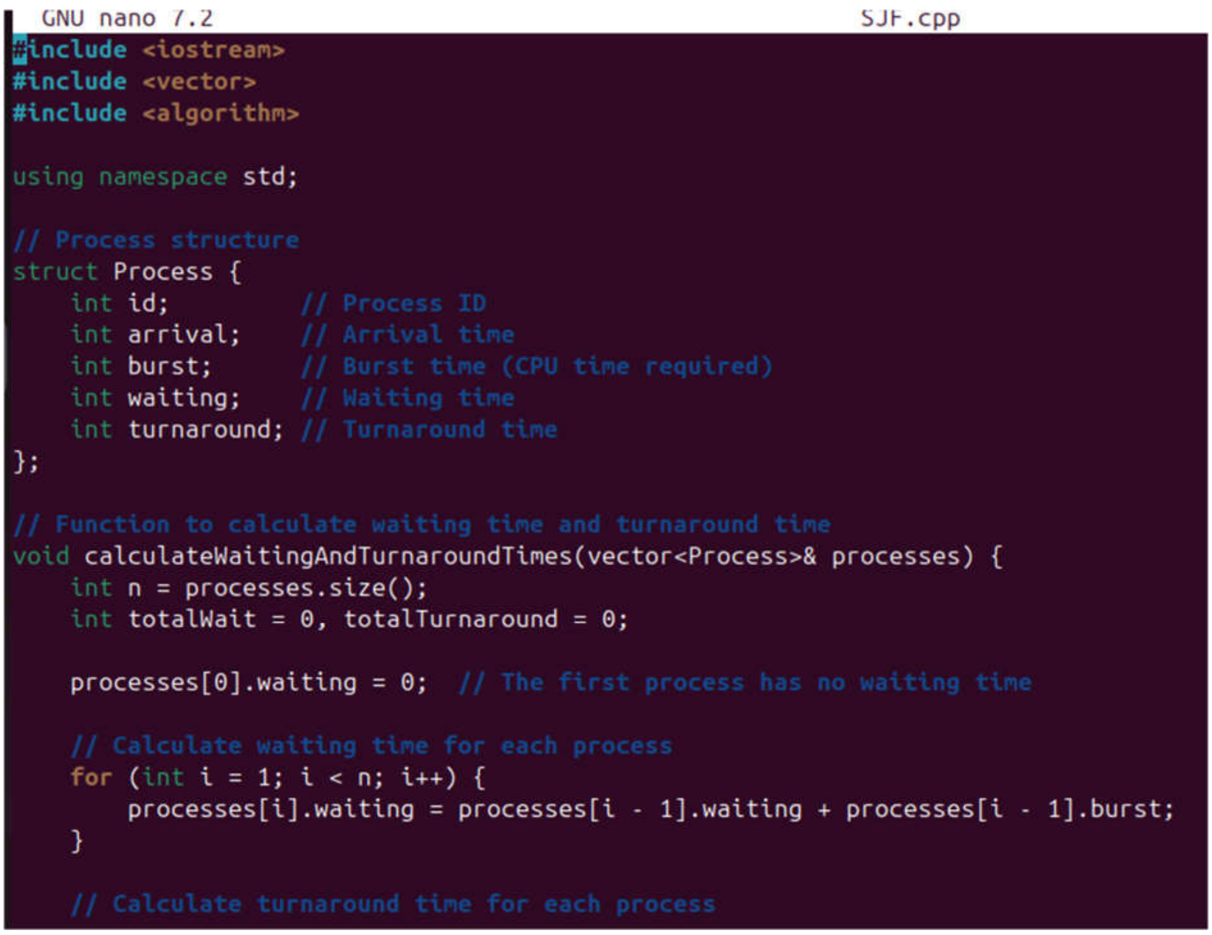
```cpp
    // Input process details
    for (int i = 0; i < n; i++) {
        processes[i].id = i + 1; // Assigning process ID (1-based)
        cout << "Enter arrival time for process " << processes[i].id << ": ";
        cin >> processes[i].arrival;
        cout << "Enter burst time for process " << processes[i].id << ": ";
        cin >> processes[i].burst;
    }

    // Sort processes based on arrival time (FCFS executes in the order of arrival)
    sort(processes.begin(), processes.end(), [](const Process& p1, const Process& p2) {
        return p1.arrival < p2.arrival;  // Sorting in ascending order of arrival time
    });

    // Calculate waiting time and turnaround time
    calculateWaitingAndTurnaroundTimes(processes);

    // Display the results
    displayProcesses(processes);

    return 0;
}
```

Output –

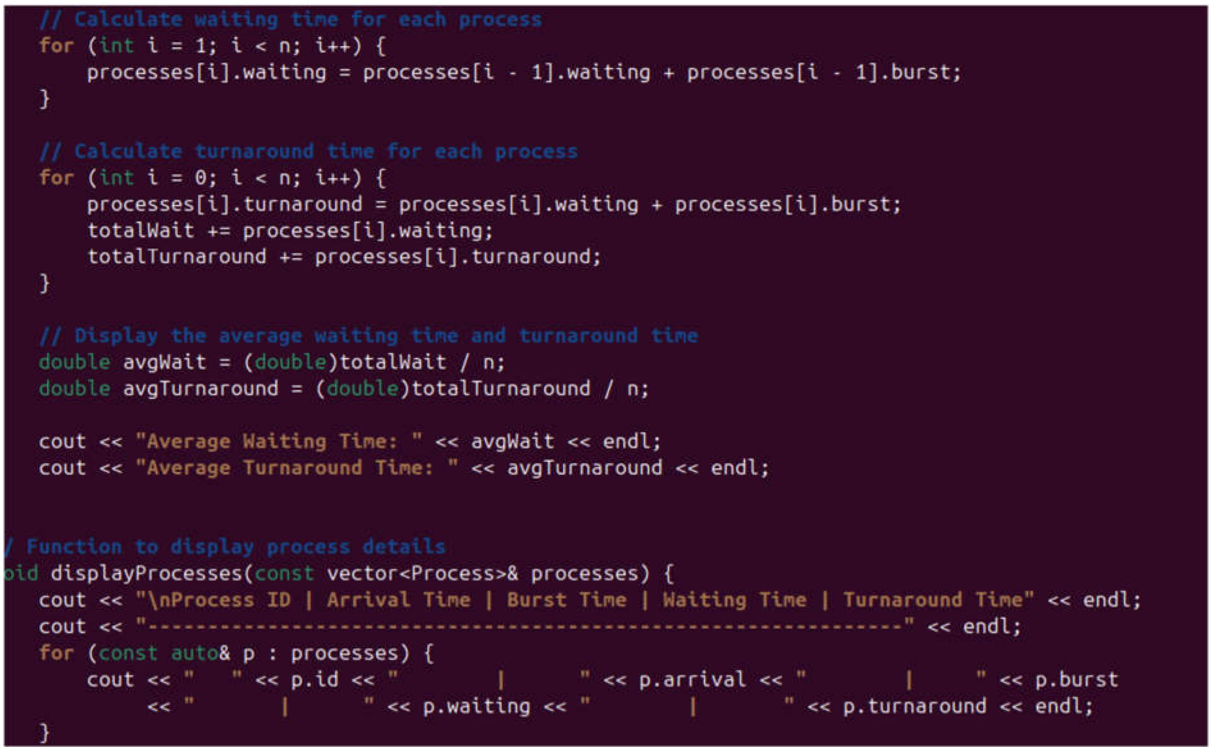


```
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$ g++ -o FCFS FCFS.cpp
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$ ./FCFS
Enter the number of processes: 7
Enter arrival time for process 1: 6
Enter burst time for process 1: 5
Enter arrival time for process 2: 8
Enter burst time for process 2: 3
Enter arrival time for process 3: 6
Enter burst time for process 3: 5
Enter arrival time for process 4: 7
Enter burst time for process 4: 9
Enter arrival time for process 5: 2
Enter burst time for process 5: 5
Enter arrival time for process 6: 6
Enter burst time for process 6: 4
Enter arrival time for process 7: 5
Enter burst time for process 7: 3
Average Waiting Time: 13.8571
Average Turnaround Time: 18.7143

Process ID | Arrival Time | Burst Time | Waiting Time | Turnaround Time
-----------------------------------------------------------------
   5       |      2       |     5      |      0       |      5
   7       |      5       |     3      |      5       |      8
   1       |      6       |     5      |      8       |      13
   3       |      6       |     5      |      13       |      18
   6       |      6       |     4      |      18       |      22
   4       |      7       |     9      |      22       |      31
   2       |      8       |     3      |      31       |      34
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$
```

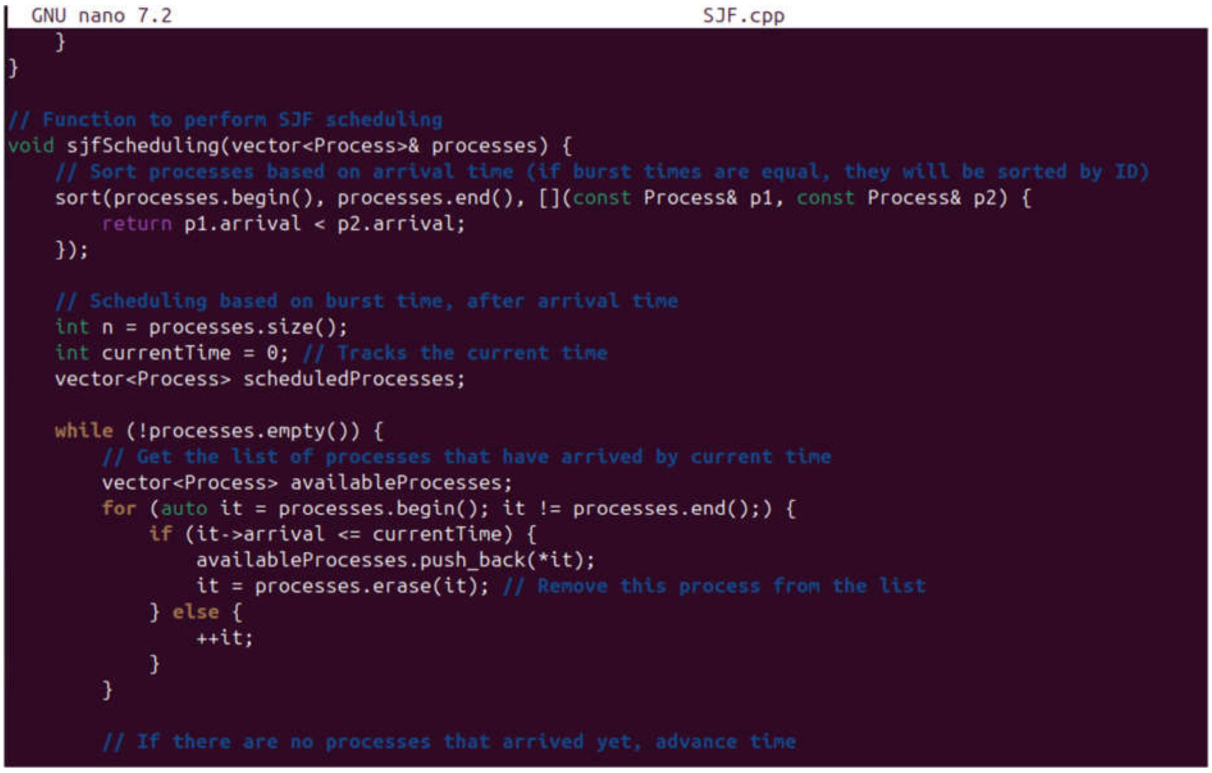
8. Write a program to implement Shortest Job First SJF Scheduling Algorithm.



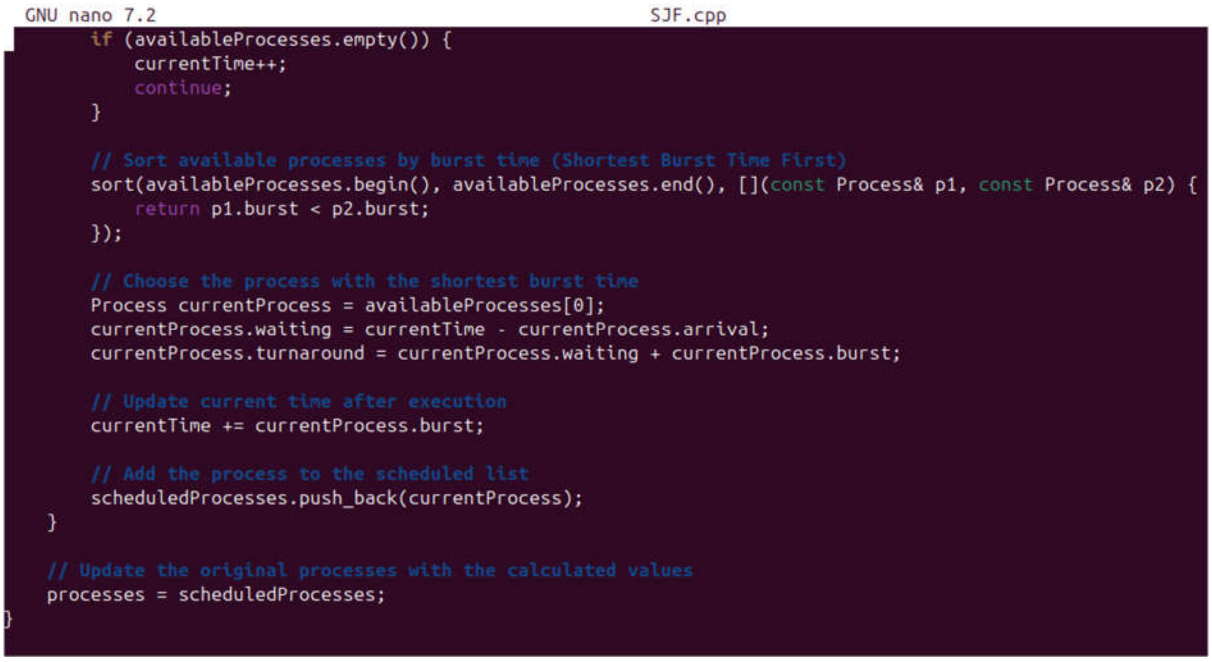
```
  GNU nano 7.2                                          SJF.cpp
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

// Process structure
struct Process {
    int id;          // Process ID
    int arrival;     // Arrival time
    int burst;       // Burst time (CPU time required)
    int waiting;     // Waiting time
    int turnaround;  // Turnaround time
};

// Function to calculate waiting time and turnaround time
void calculateWaitingAndTurnaroundTimes(vector<Process>& processes) {
    int n = processes.size();
    int totalWait = 0, totalTurnaround = 0;

    processes[0].waiting = 0;  // The first process has no waiting time

    // Calculate waiting time for each process
    for (int i = 1; i < n; i++) {
        processes[i].waiting = processes[i - 1].waiting + processes[i - 1].burst;
    }

    // Calculate turnaround time for each process
```



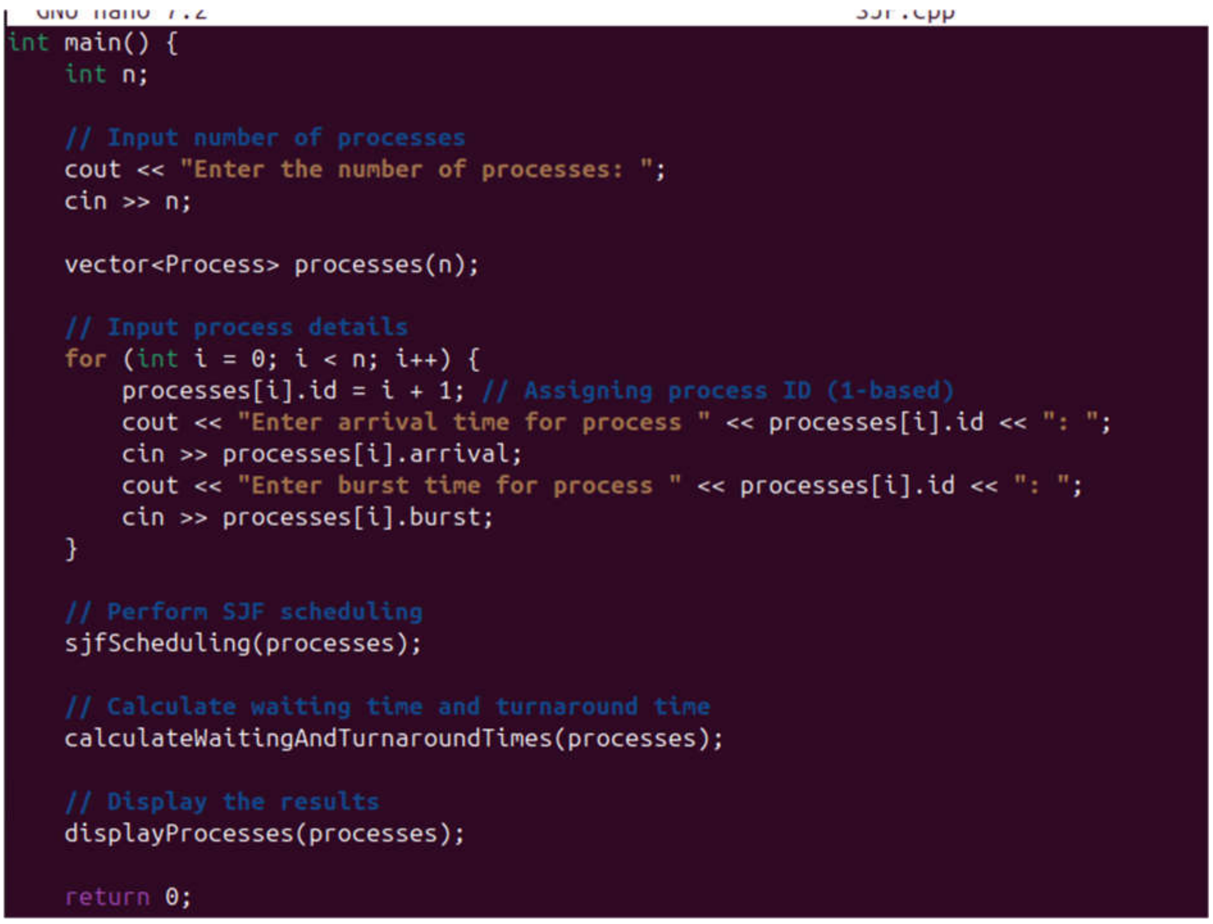
```
   // Calculate waiting time for each process
   for (int i = 1; i < n; i++) {
       processes[i].waiting = processes[i - 1].waiting + processes[i - 1].burst;
   }

   // Calculate turnaround time for each process
   for (int i = 0; i < n; i++) {
       processes[i].turnaround = processes[i].waiting + processes[i].burst;
       totalWait += processes[i].waiting;
       totalTurnaround += processes[i].turnaround;
   }

   // Display the average waiting time and turnaround time
   double avgWait = (double)totalWait / n;
   double avgTurnaround = (double)totalTurnaround / n;

   cout << "Average Waiting Time: " << avgWait << endl;
   cout << "Average Turnaround Time: " << avgTurnaround << endl;


/ Function to display process details
oid displayProcesses(const vector<Process>& processes) {
   cout << "\nProcess ID | Arrival Time | Burst Time | Waiting Time | Turnaround Time" << endl;
   cout << "-----------------------------------------------------------------" << endl;
   for (const auto& p : processes) {
       cout << "    " << p.id << "          |      " << p.arrival << "         |     " << p.burst
            << "       |      " << p.waiting << "          |        " << p.turnaround << endl;
   }
```



```
  GNU nano 7.2                                        SJF.cpp
    }
}

// Function to perform SJF scheduling
void sjfScheduling(vector<Process>& processes) {
    // Sort processes based on arrival time (if burst times are equal, they will be sorted by ID)
    sort(processes.begin(), processes.end(), [](const Process& p1, const Process& p2) {
        return p1.arrival < p2.arrival;
    });

    // Scheduling based on burst time, after arrival time
    int n = processes.size();
    int currentTime = 0; // Tracks the current time
    vector<Process> scheduledProcesses;

    while (!processes.empty()) {
        // Get the list of processes that have arrived by current time
        vector<Process> availableProcesses;
        for (auto it = processes.begin(); it != processes.end();) {
            if (it->arrival <= currentTime) {
                availableProcesses.push_back(*it);
                it = processes.erase(it); // Remove this process from the list
            } else {
                ++it;
            }
        }

        // If there are no processes that arrived yet, advance time
```

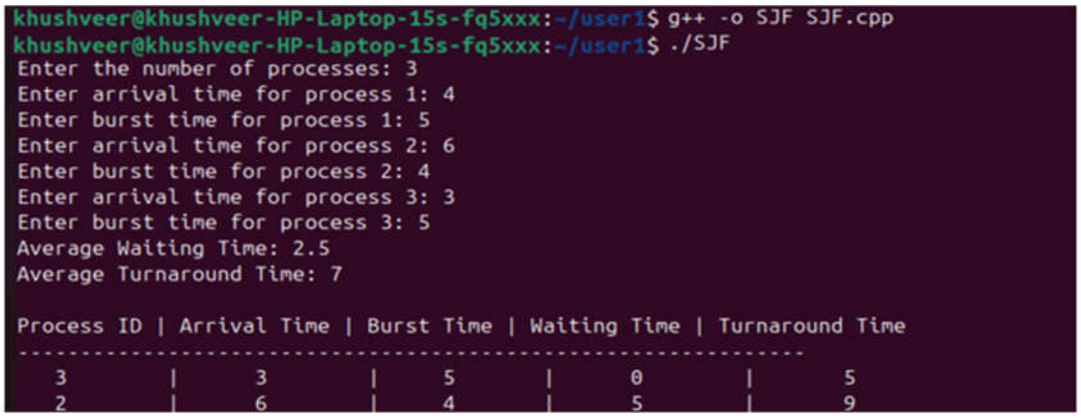


```
  GNU nano 7.2                                        SJF.cpp
        if (availableProcesses.empty()) {
            currentTime++;
            continue;
        }

        // Sort available processes by burst time (Shortest Burst Time First)
        sort(availableProcesses.begin(), availableProcesses.end(), [](const Process& p1, const Process& p2) {
            return p1.burst < p2.burst;
        });

        // Choose the process with the shortest burst time
        Process currentProcess = availableProcesses[0];
        currentProcess.waiting = currentTime - currentProcess.arrival;
        currentProcess.turnaround = currentProcess.waiting + currentProcess.burst;

        // Update current time after execution
        currentTime += currentProcess.burst;

        // Add the process to the scheduled list
        scheduledProcesses.push_back(currentProcess);
    }

    // Update the original processes with the calculated values
    processes = scheduledProcesses;
}
```



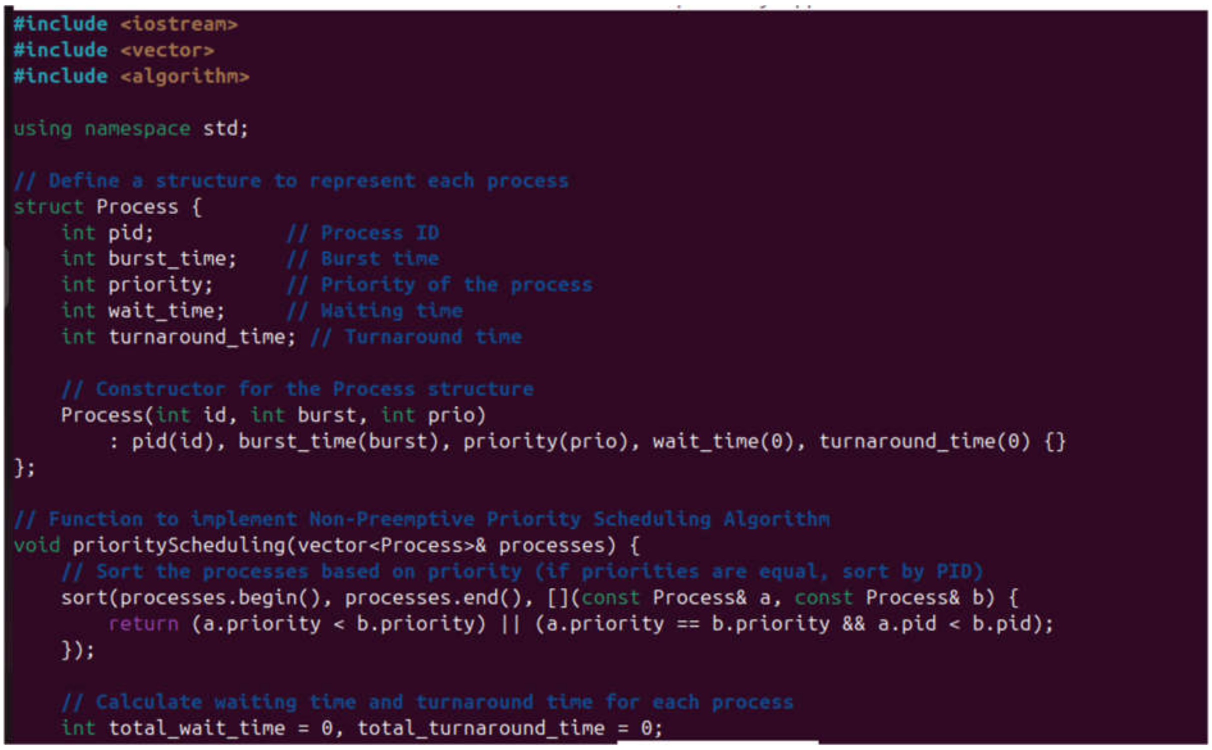
```
  GNU nano 7.2                                         SJF.cpp
int main() {
    int n;

    // Input number of processes
    cout << "Enter the number of processes: ";
    cin >> n;

    vector<Process> processes(n);

    // Input process details
    for (int i = 0; i < n; i++) {
        processes[i].id = i + 1; // Assigning process ID (1-based)
        cout << "Enter arrival time for process " << processes[i].id << ": ";
        cin >> processes[i].arrival;
        cout << "Enter burst time for process " << processes[i].id << ": ";
        cin >> processes[i].burst;
    }

    // Perform SJF scheduling
    sjfScheduling(processes);

    // Calculate waiting time and turnaround time
    calculateWaitingAndTurnaroundTimes(processes);

    // Display the results
    displayProcesses(processes);

    return 0;
```

Output –

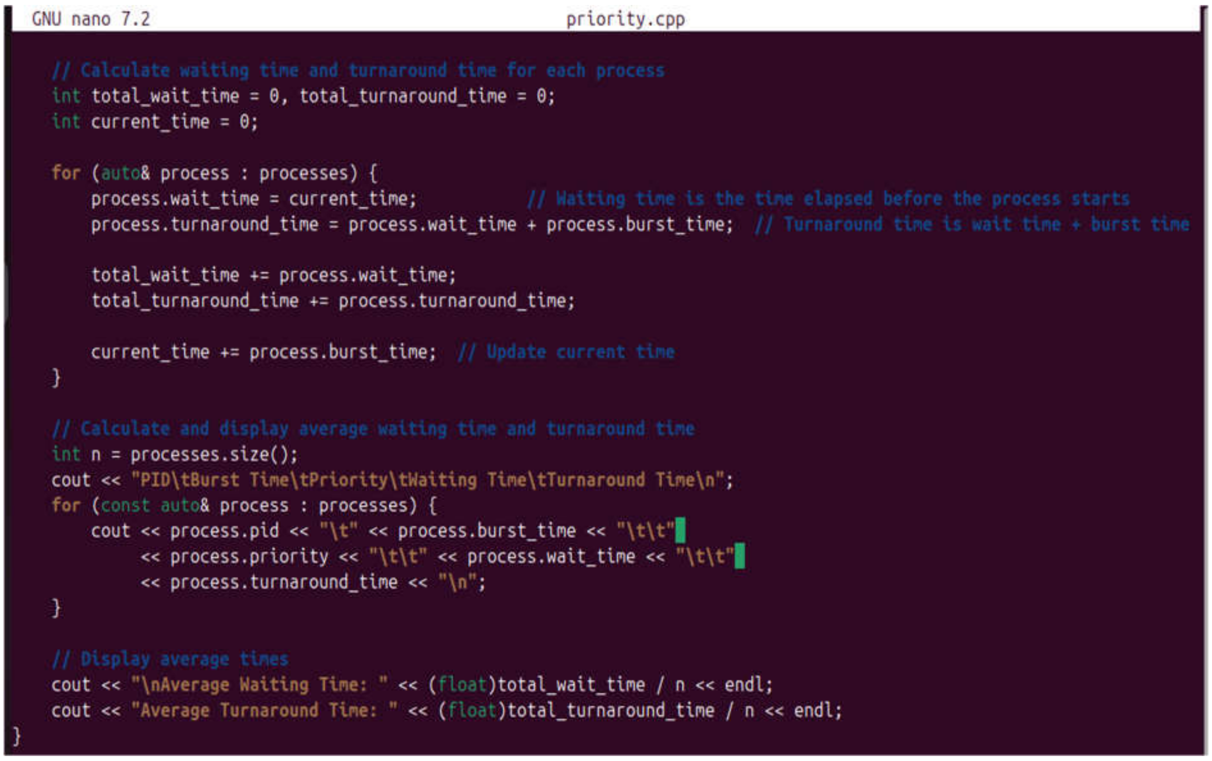


```
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$ g++ -o SJF SJF.cpp
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$ ./SJF
Enter the number of processes: 3
Enter arrival time for process 1: 4
Enter burst time for process 1: 5
Enter arrival time for process 2: 6
Enter burst time for process 2: 4
Enter arrival time for process 3: 3
Enter burst time for process 3: 5
Average Waiting Time: 2.5
Average Turnaround Time: 7

Process ID | Arrival Time | Burst Time | Waiting Time | Turnaround Time
-------------------------------------------------------------------
    3      |      3       |     5      |      0       |       5
    2      |      6       |     4      |      5       |       9
```

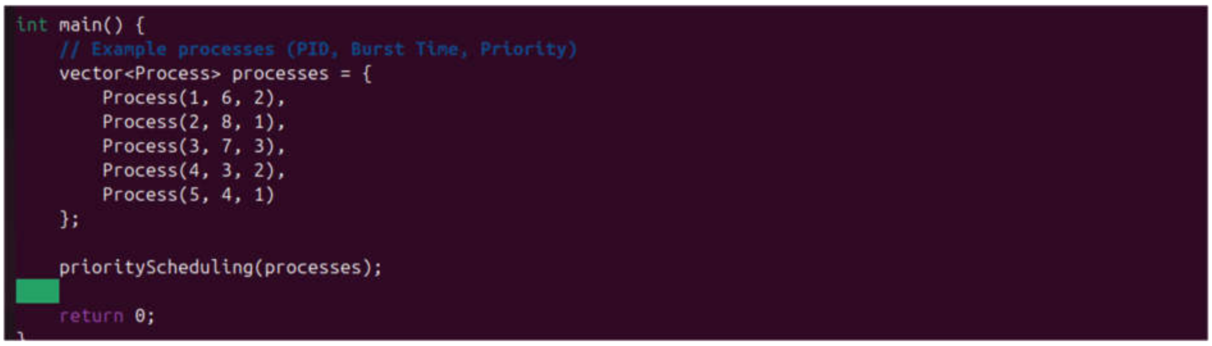
9. Write a program to implement non-primitive priority-based scheduling algorithm.



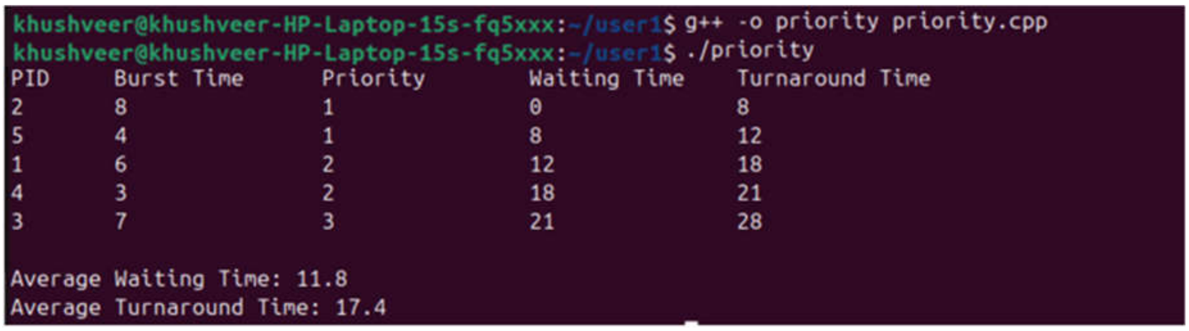
```cpp
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

// Define a structure to represent each process
struct Process {
    int pid;            // Process ID
    int burst_time;     // Burst time
    int priority;       // Priority of the process
    int wait_time;      // Waiting time
    int turnaround_time; // Turnaround time

    // Constructor for the Process structure
    Process(int id, int burst, int prio)
        : pid(id), burst_time(burst), priority(prio), wait_time(0), turnaround_time(0) {}
};

// Function to implement Non-Preemptive Priority Scheduling Algorithm
void priorityScheduling(vector<Process>& processes) {
    // Sort the processes based on priority (if priorities are equal, sort by PID)
    sort(processes.begin(), processes.end(), [](const Process& a, const Process& b) {
        return (a.priority < b.priority) || (a.priority == b.priority && a.pid < b.pid);
    });

    // Calculate waiting time and turnaround time for each process
    int total_wait_time = 0, total_turnaround_time = 0;
```



```cpp
 GNU nano 7.2                                priority.cpp

    // Calculate waiting time and turnaround time for each process
    int total_wait_time = 0, total_turnaround_time = 0;
    int current_time = 0;

    for (auto& process : processes) {
        process.wait_time = current_time;           // Waiting time is the time elapsed before the process starts
        process.turnaround_time = process.wait_time + process.burst_time;  // Turnaround time is wait time + burst time

        total_wait_time += process.wait_time;
        total_turnaround_time += process.turnaround_time;

        current_time += process.burst_time;  // Update current time
    }

    // Calculate and display average waiting time and turnaround time
    int n = processes.size();
    cout << "PID\tBurst Time\tPriority\tWaiting Time\tTurnaround Time\n";
    for (const auto& process : processes) {
        cout << process.pid << "\t" << process.burst_time << "\t\t"
             << process.priority << "\t\t" << process.wait_time << "\t\t"
             << process.turnaround_time << "\n";
    }

    // Display average times
    cout << "\nAverage Waiting Time: " << (float)total_wait_time / n << endl;
    cout << "Average Turnaround Time: " << (float)total_turnaround_time / n << endl;
}
```

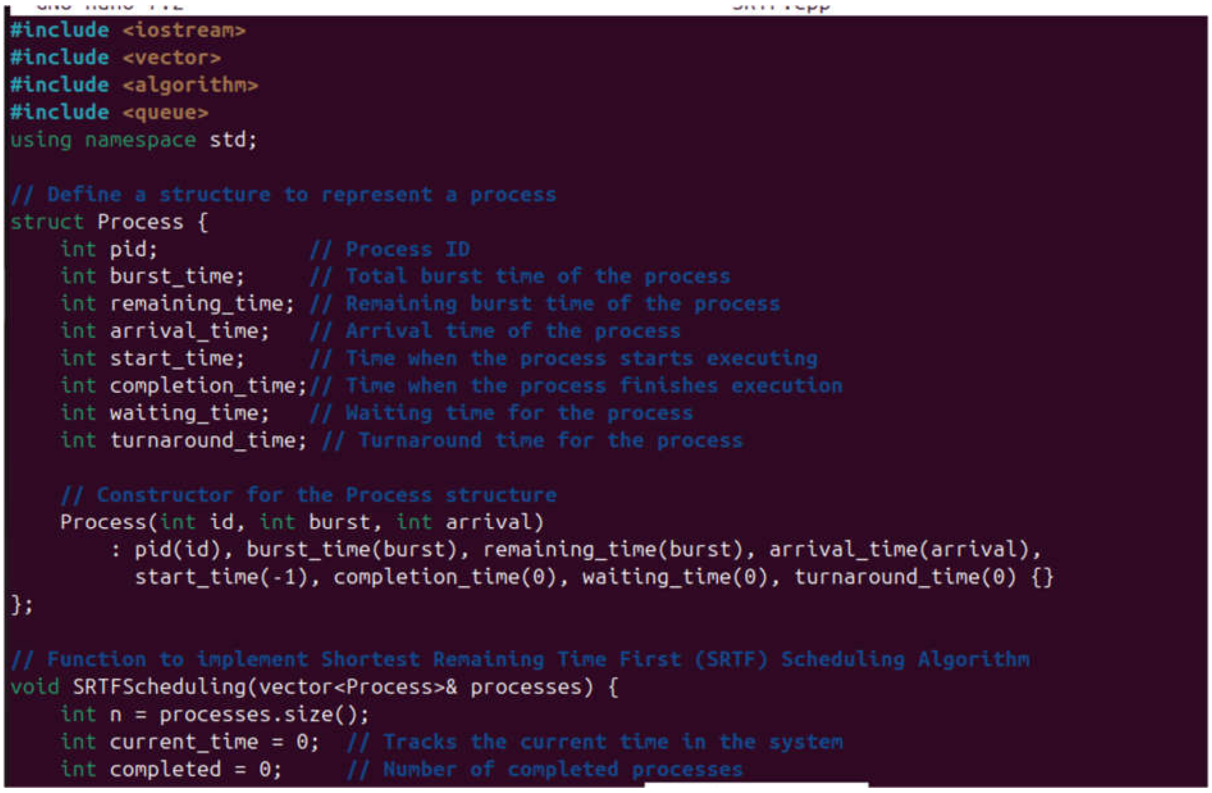


```
int main() {
    // Example processes (PID, Burst Time, Priority)
    vector<Process> processes = {
        Process(1, 6, 2),
        Process(2, 8, 1),
        Process(3, 7, 3),
        Process(4, 3, 2),
        Process(5, 4, 1)
    };

    priorityScheduling(processes);

    return 0;
```

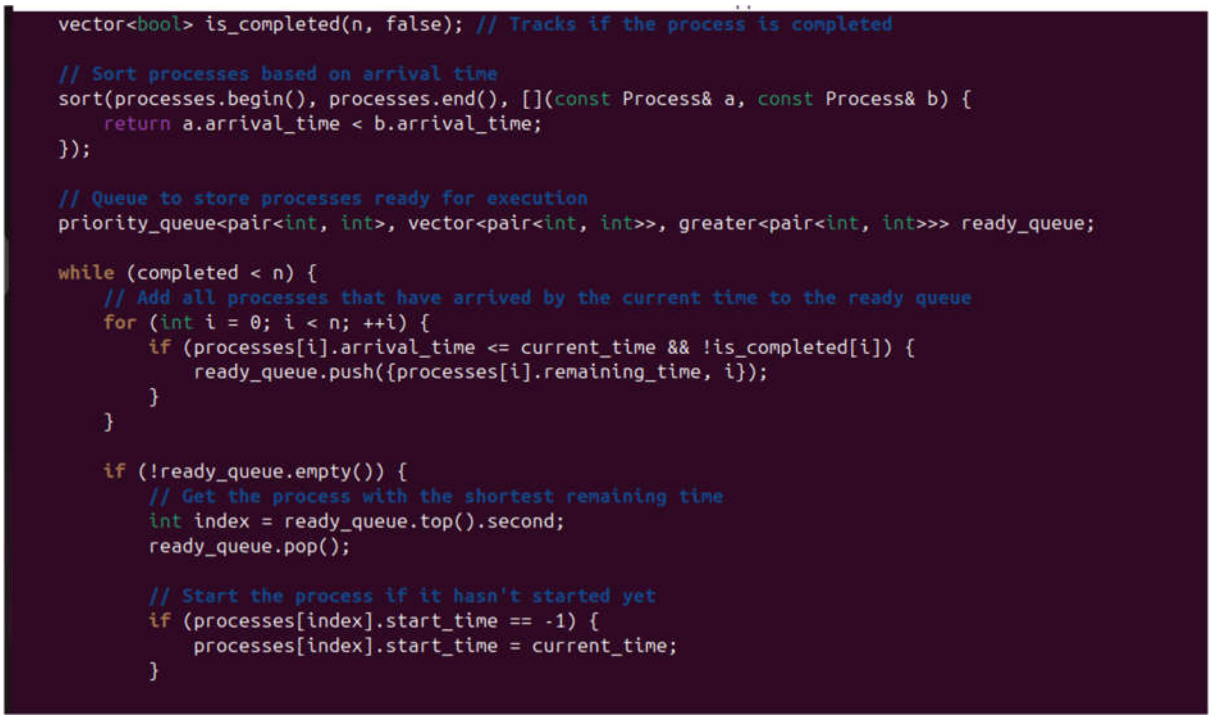


```
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$ g++ -o priority priority.cpp
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$ ./priority
PID     Burst Time      Priority        Waiting Time    Turnaround Time
2       8               1               0               8
5       4               1               8               12
1       6               2               12              18
4       3               2               18              21
3       7               3               21              28

Average Waiting Time: 11.8
Average Turnaround Time: 17.4
```

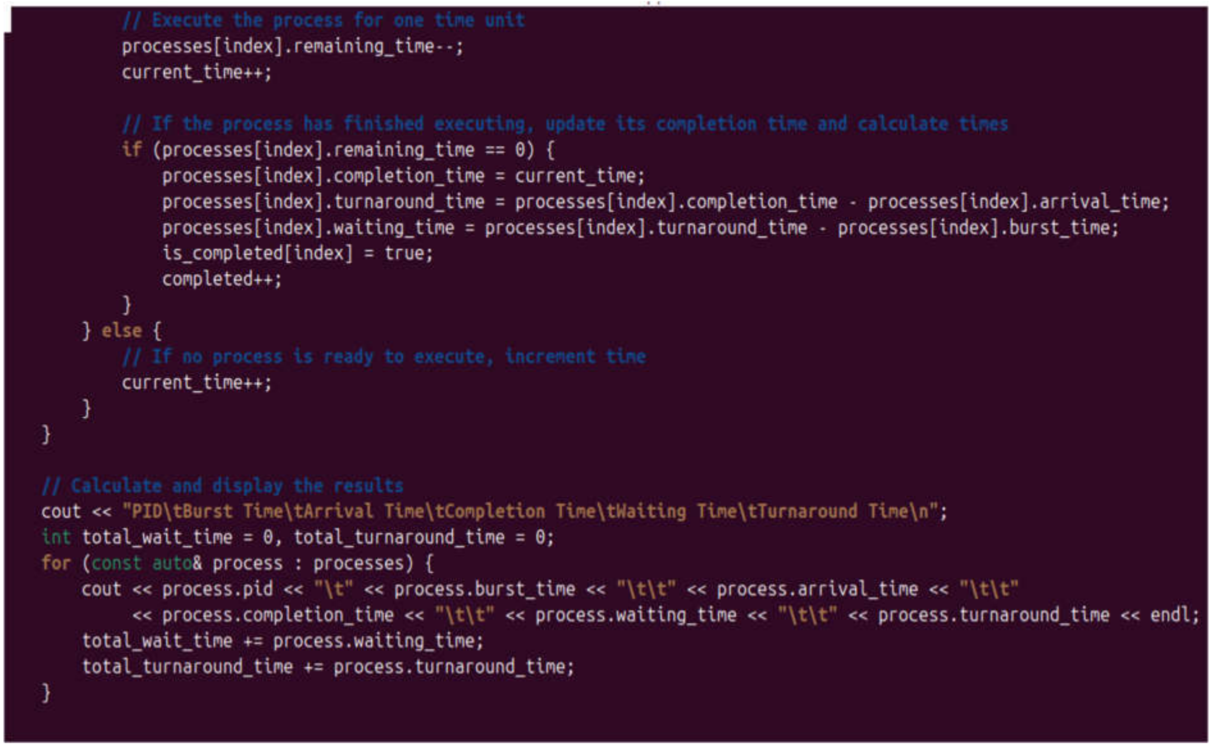
10. Write a program to implement SRTF scheduling algorithm.



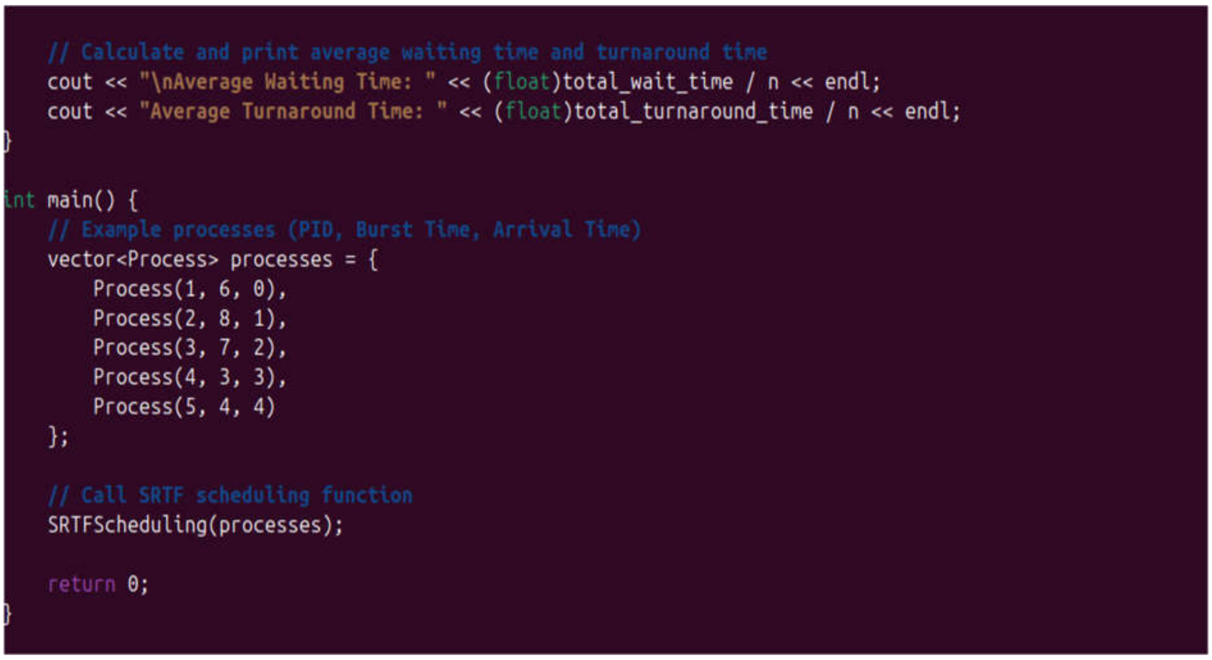
```
#include <iostream>
#include <vector>
#include <algorithm>
#include <queue>
using namespace std;

// Define a structure to represent a process
struct Process {
    int pid;            // Process ID
    int burst_time;     // Total burst time of the process
    int remaining_time; // Remaining burst time of the process
    int arrival_time;   // Arrival time of the process
    int start_time;     // Time when the process starts executing
    int completion_time;// Time when the process finishes execution
    int waiting_time;   // Waiting time for the process
    int turnaround_time; // Turnaround time for the process

    // Constructor for the Process structure
    Process(int id, int burst, int arrival)
        : pid(id), burst_time(burst), remaining_time(burst), arrival_time(arrival),
          start_time(-1), completion_time(0), waiting_time(0), turnaround_time(0) {}
};

// Function to implement Shortest Remaining Time First (SRTF) Scheduling Algorithm
void SRTFScheduling(vector<Process>& processes) {
    int n = processes.size();
    int current_time = 0;  // Tracks the current time in the system
    int completed = 0;     // Number of completed processes
```



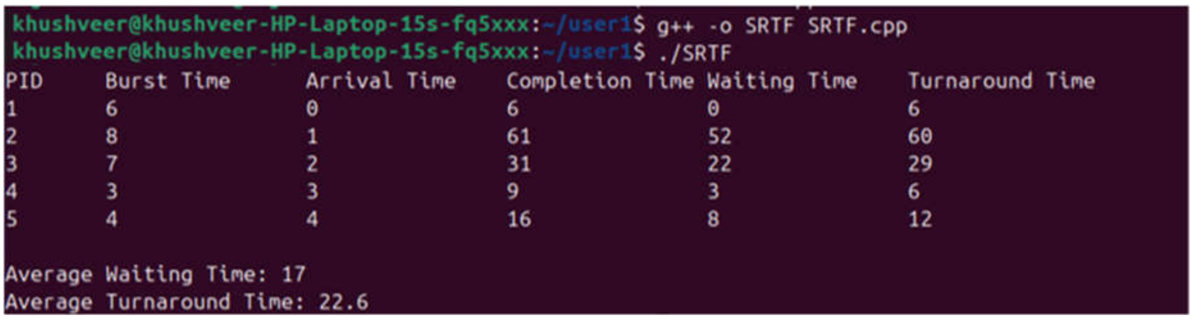
```cpp
    vector<bool> is_completed(n, false); // Tracks if the process is completed

    // Sort processes based on arrival time
    sort(processes.begin(), processes.end(), [](const Process& a, const Process& b) {
        return a.arrival_time < b.arrival_time;
    });

    // Queue to store processes ready for execution
    priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> ready_queue;

    while (completed < n) {
        // Add all processes that have arrived by the current time to the ready queue
        for (int i = 0; i < n; ++i) {
            if (processes[i].arrival_time <= current_time && !is_completed[i]) {
                ready_queue.push({processes[i].remaining_time, i});
            }
        }

        if (!ready_queue.empty()) {
            // Get the process with the shortest remaining time
            int index = ready_queue.top().second;
            ready_queue.pop();

            // Start the process if it hasn't started yet
            if (processes[index].start_time == -1) {
                processes[index].start_time = current_time;
            }
```



```cpp
            // Execute the process for one time unit
            processes[index].remaining_time--;
            current_time++;

            // If the process has finished executing, update its completion time and calculate times
            if (processes[index].remaining_time == 0) {
                processes[index].completion_time = current_time;
                processes[index].turnaround_time = processes[index].completion_time - processes[index].arrival_time;
                processes[index].waiting_time = processes[index].turnaround_time - processes[index].burst_time;
                is_completed[index] = true;
                completed++;
            }
        } else {
            // If no process is ready to execute, increment time
            current_time++;
        }
    }

    // Calculate and display the results
    cout << "PID\tBurst Time\tArrival Time\tCompletion Time\tWaiting Time\tTurnaround Time\n";
    int total_wait_time = 0, total_turnaround_time = 0;
    for (const auto& process : processes) {
        cout << process.pid << "\t" << process.burst_time << "\t\t" << process.arrival_time << "\t\t"
             << process.completion_time << "\t\t" << process.waiting_time << "\t\t" << process.turnaround_time << endl;
        total_wait_time += process.waiting_time;
        total_turnaround_time += process.turnaround_time;
    }
```

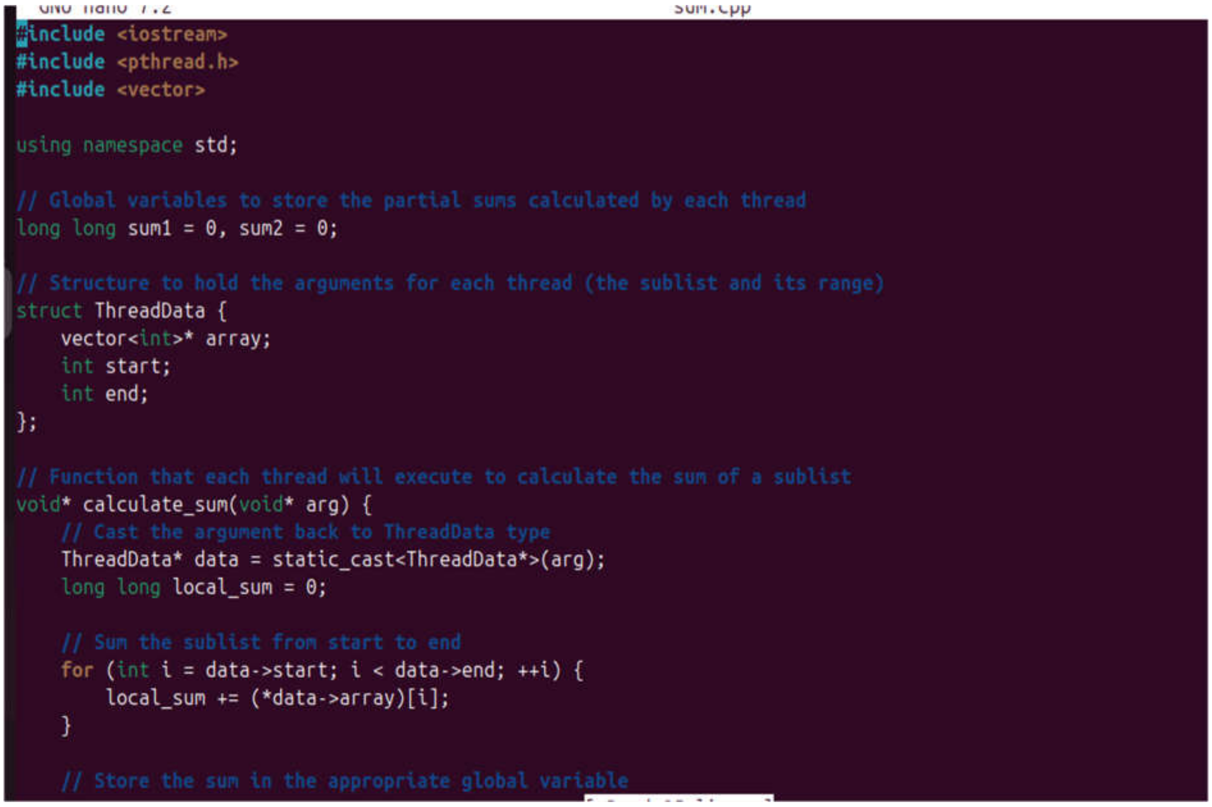


```
    // Calculate and print average waiting time and turnaround time
    cout << "\nAverage Waiting Time: " << (float)total_wait_time / n << endl;
    cout << "Average Turnaround Time: " << (float)total_turnaround_time / n << endl;
}

int main() {
    // Example processes (PID, Burst Time, Arrival Time)
    vector<Process> processes = {
        Process(1, 6, 0),
        Process(2, 8, 1),
        Process(3, 7, 2),
        Process(4, 3, 3),
        Process(5, 4, 4)
    };

    // Call SRTF scheduling function
    SRTFScheduling(processes);

    return 0;
}
```

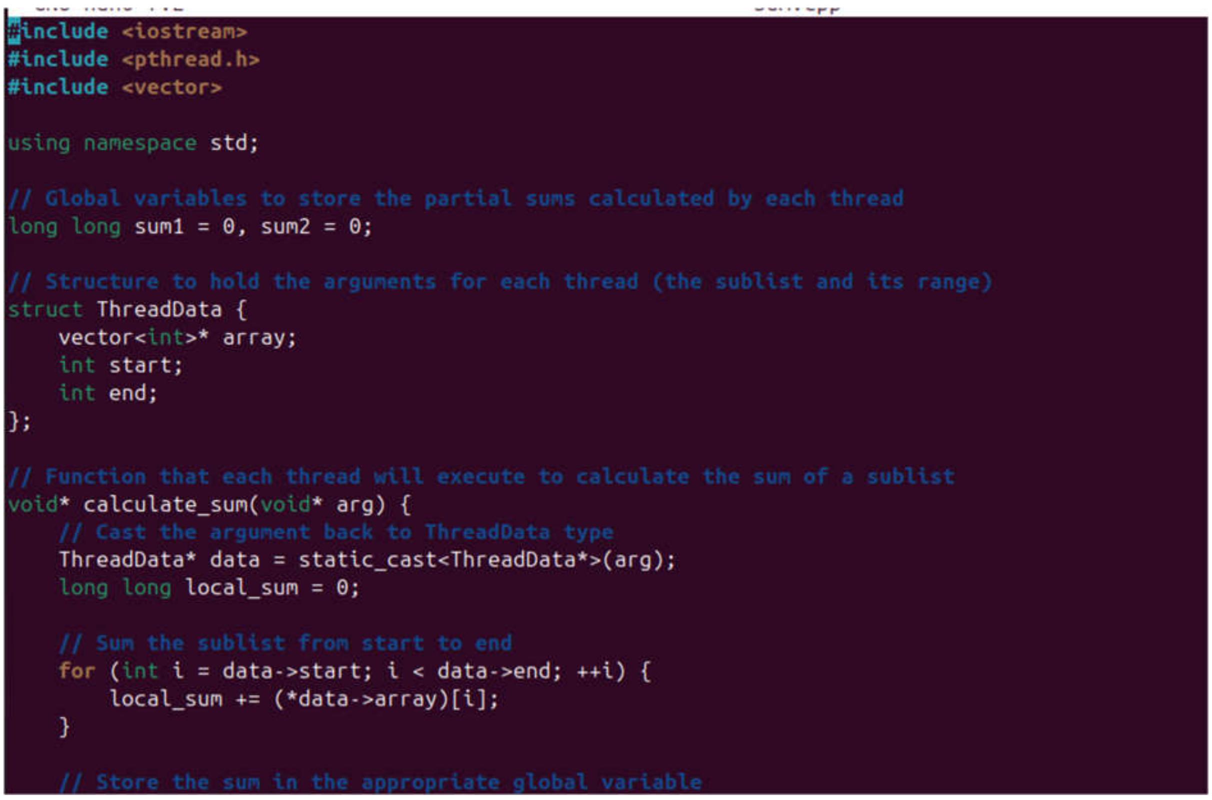


```
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$ g++ -o SRTF SRTF.cpp
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$ ./SRTF
PID     Burst Time      Arrival Time    Completion Time Waiting Time    Turnaround Time
1       6               0               6               0               6
2       8               1               61              52              60
3       7               2               31              22              29
4       3               3               9               3               6
5       4               4               16              8               12

Average Waiting Time: 17
Average Turnaround Time: 22.6
```

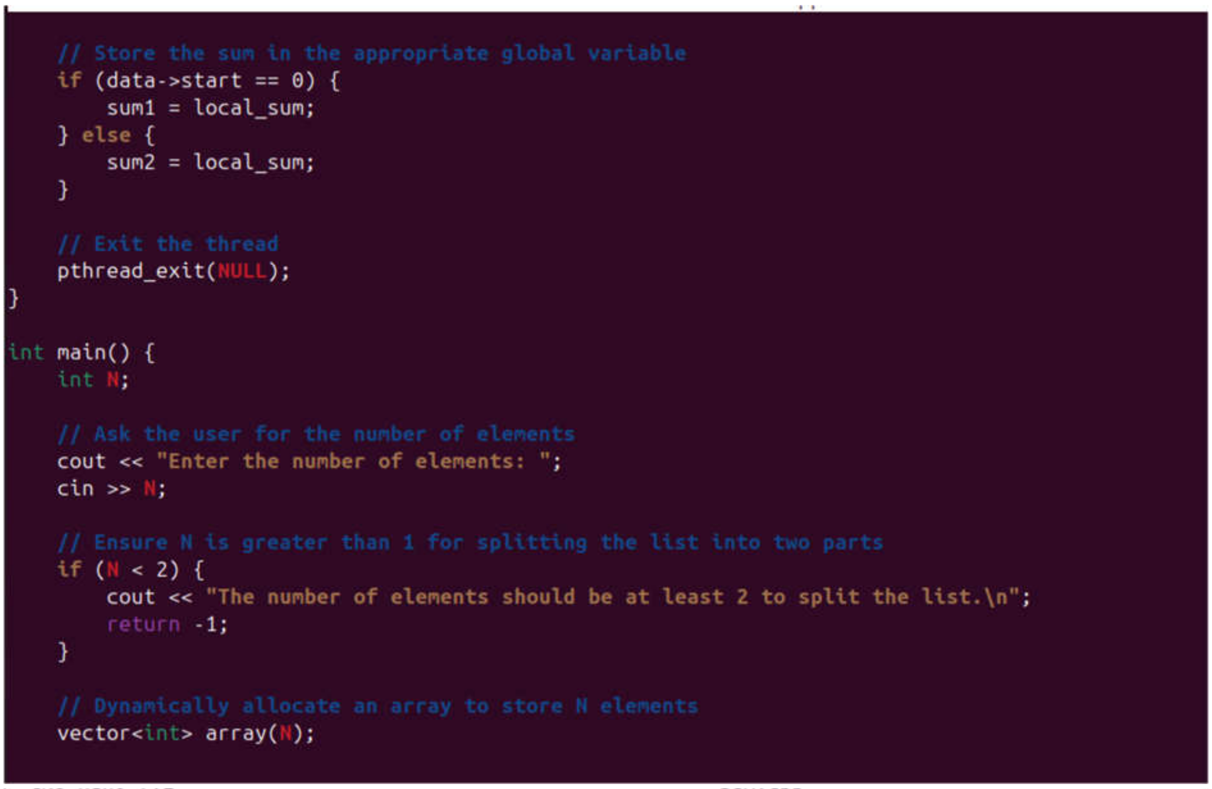
11. Write a program to calculate sum of n numbers using pthreads. A list of n numbers is divided into two smaller lists of equal size. Two separate threads are used to run the sublist.



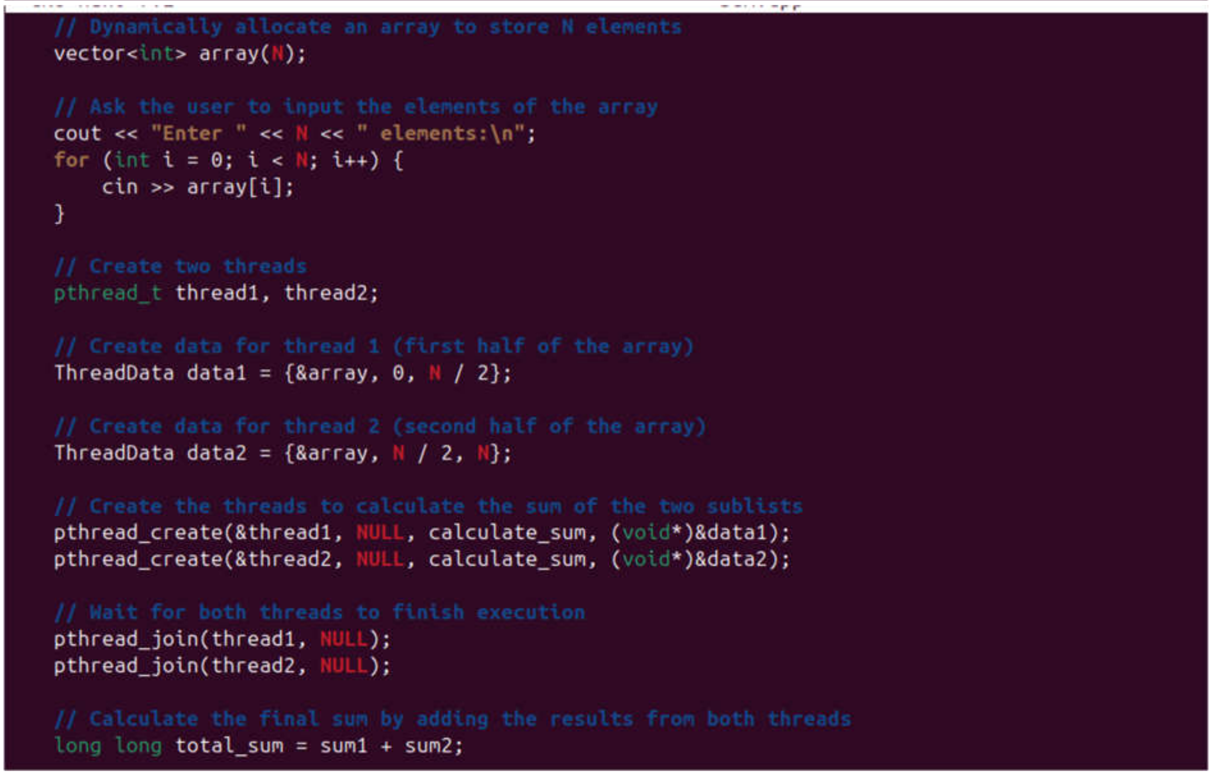
```cpp
#include <iostream>
#include <pthread.h>
#include <vector>

using namespace std;

// Global variables to store the partial sums calculated by each thread
long long sum1 = 0, sum2 = 0;

// Structure to hold the arguments for each thread (the sublist and its range)
struct ThreadData {
    vector<int>* array;
    int start;
    int end;
};

// Function that each thread will execute to calculate the sum of a sublist
void* calculate_sum(void* arg) {
    // Cast the argument back to ThreadData type
    ThreadData* data = static_cast<ThreadData*>(arg);
    long long local_sum = 0;

    // Sum the sublist from start to end
    for (int i = data->start; i < data->end; ++i) {
        local_sum += (*data->array)[i];
    }

    // Store the sum in the appropriate global variable
```



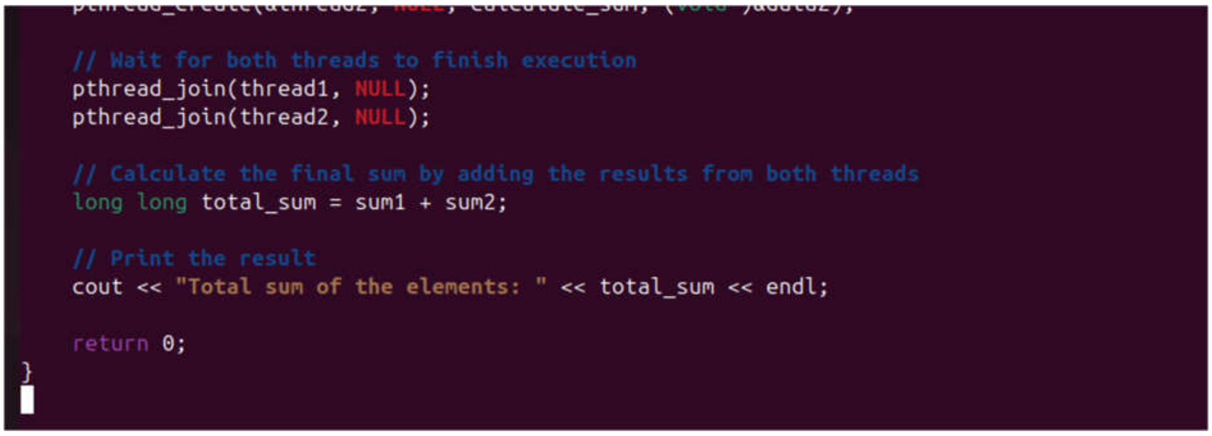
```cpp
#include <iostream>
#include <pthread.h>
#include <vector>

using namespace std;

// Global variables to store the partial sums calculated by each thread
long long sum1 = 0, sum2 = 0;

// Structure to hold the arguments for each thread (the sublist and its range)
struct ThreadData {
    vector<int>* array;
    int start;
    int end;
};

// Function that each thread will execute to calculate the sum of a sublist
void* calculate_sum(void* arg) {
    // Cast the argument back to ThreadData type
    ThreadData* data = static_cast<ThreadData*>(arg);
    long long local_sum = 0;

    // Sum the sublist from start to end
    for (int i = data->start; i < data->end; ++i) {
        local_sum += (*data->array)[i];
    }

    // Store the sum in the appropriate global variable
```



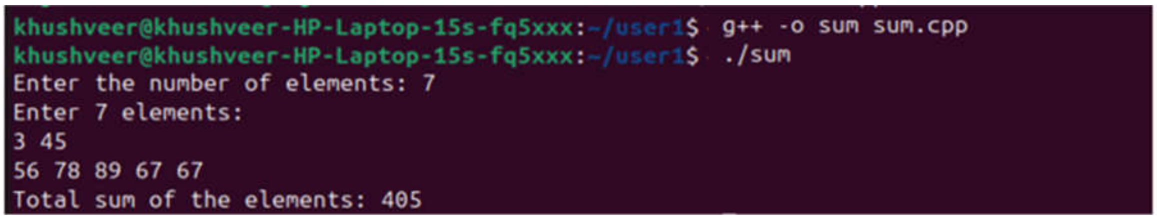
```
    // Store the sum in the appropriate global variable
    if (data->start == 0) {
        sum1 = local_sum;
    } else {
        sum2 = local_sum;
    }

    // Exit the thread
    pthread_exit(NULL);
}

int main() {
    int N;

    // Ask the user for the number of elements
    cout << "Enter the number of elements: ";
    cin >> N;

    // Ensure N is greater than 1 for splitting the list into two parts
    if (N < 2) {
        cout << "The number of elements should be at least 2 to split the list.\n";
        return -1;
    }

    // Dynamically allocate an array to store N elements
    vector<int> array(N);
```



```
    // Dynamically allocate an array to store N elements
    vector<int> array(N);

    // Ask the user to input the elements of the array
    cout << "Enter " << N << " elements:\n";
    for (int i = 0; i < N; i++) {
        cin >> array[i];
    }

    // Create two threads
    pthread_t thread1, thread2;

    // Create data for thread 1 (first half of the array)
    ThreadData data1 = {&array, 0, N / 2};

    // Create data for thread 2 (second half of the array)
    ThreadData data2 = {&array, N / 2, N};

    // Create the threads to calculate the sum of the two sublists
    pthread_create(&thread1, NULL, calculate_sum, (void*)&data1);
    pthread_create(&thread2, NULL, calculate_sum, (void*)&data2);

    // Wait for both threads to finish execution
    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);

    // Calculate the final sum by adding the results from both threads
    long long total_sum = sum1 + sum2;
```

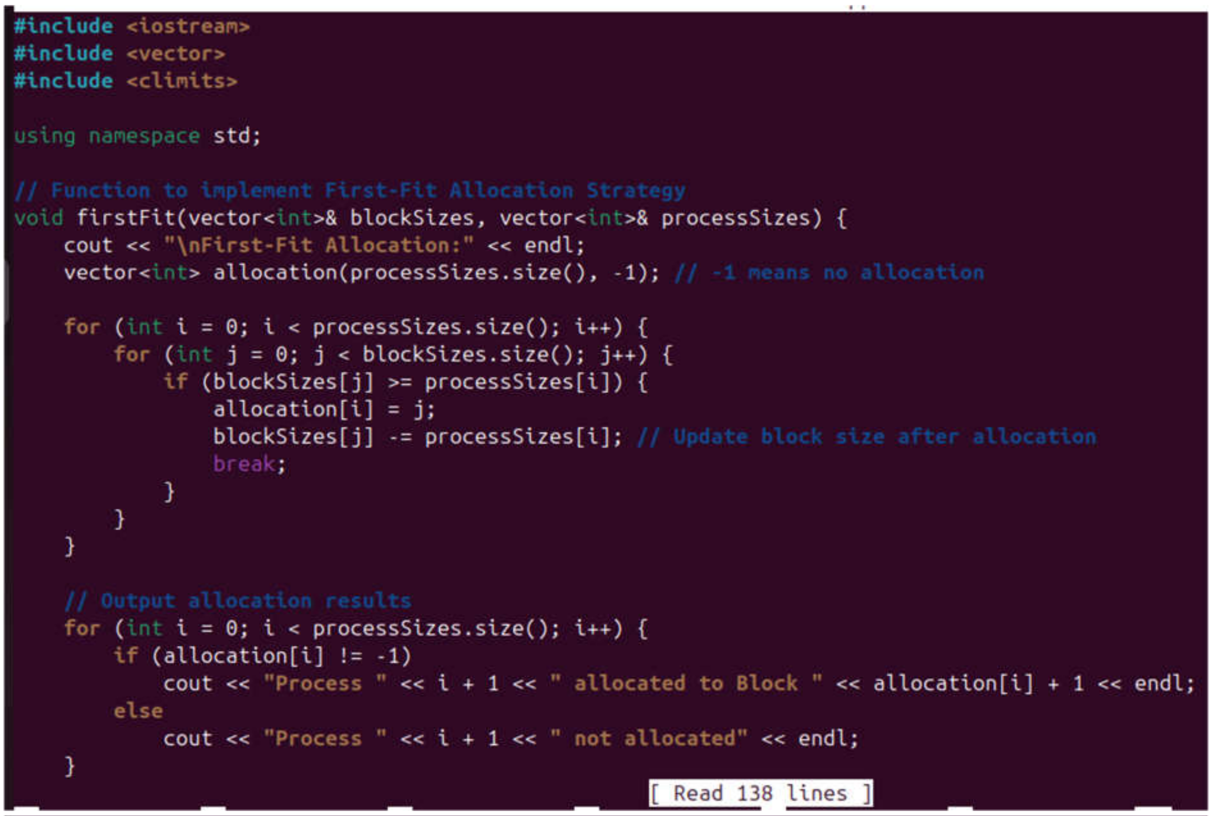


```
    pthread_create(&thread2, NULL, calculate_sum, (void*)&data2);

    // Wait for both threads to finish execution
    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);

    // Calculate the final sum by adding the results from both threads
    long long total_sum = sum1 + sum2;

    // Print the result
    cout << "Total sum of the elements: " << total_sum << endl;

    return 0;
}
```

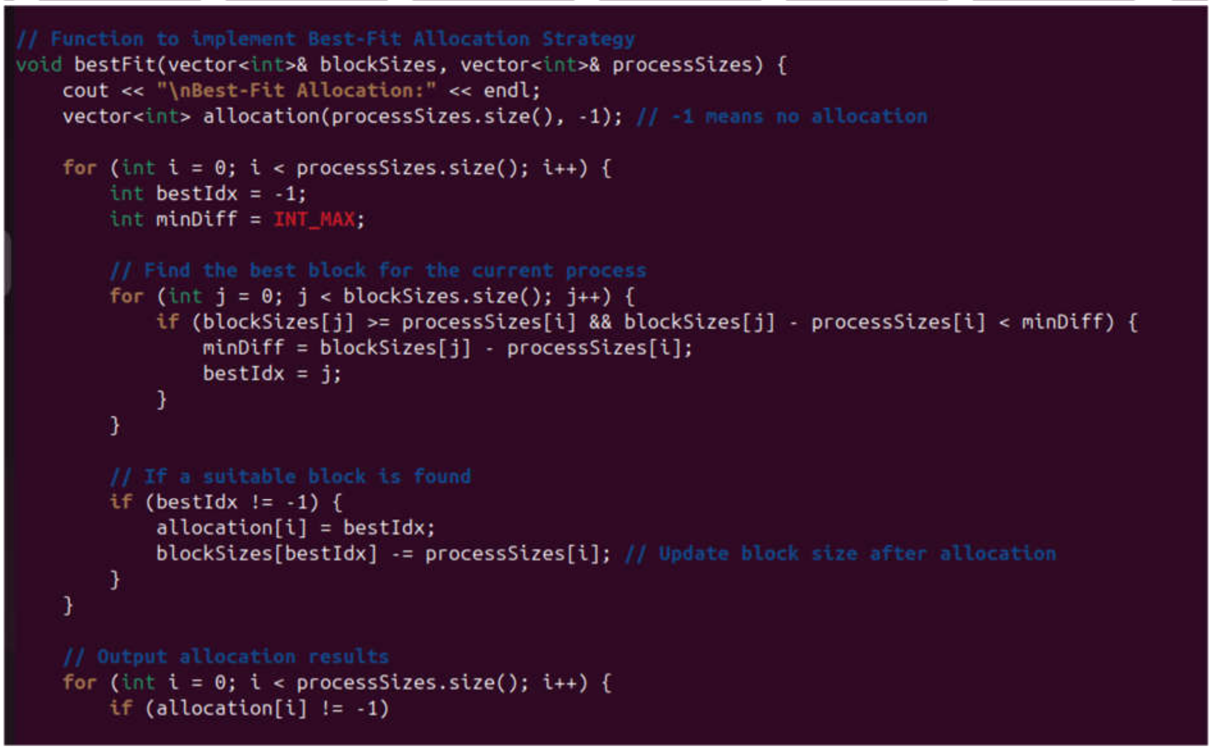


```
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$ g++ -o sum sum.cpp
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$ ./sum
Enter the number of elements: 7
Enter 7 elements:
3 45
56 78 89 67 67
Total sum of the elements: 405
```

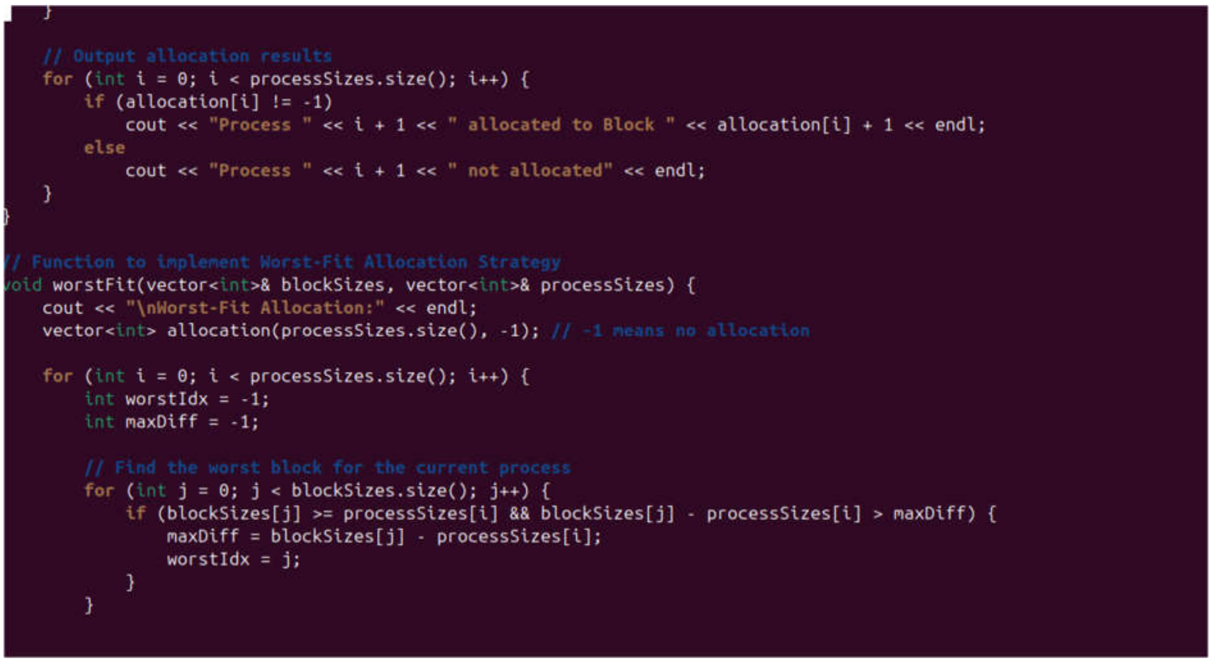
12. write a program to implement first-fit, best-fit and worst-fit allocation strategies.



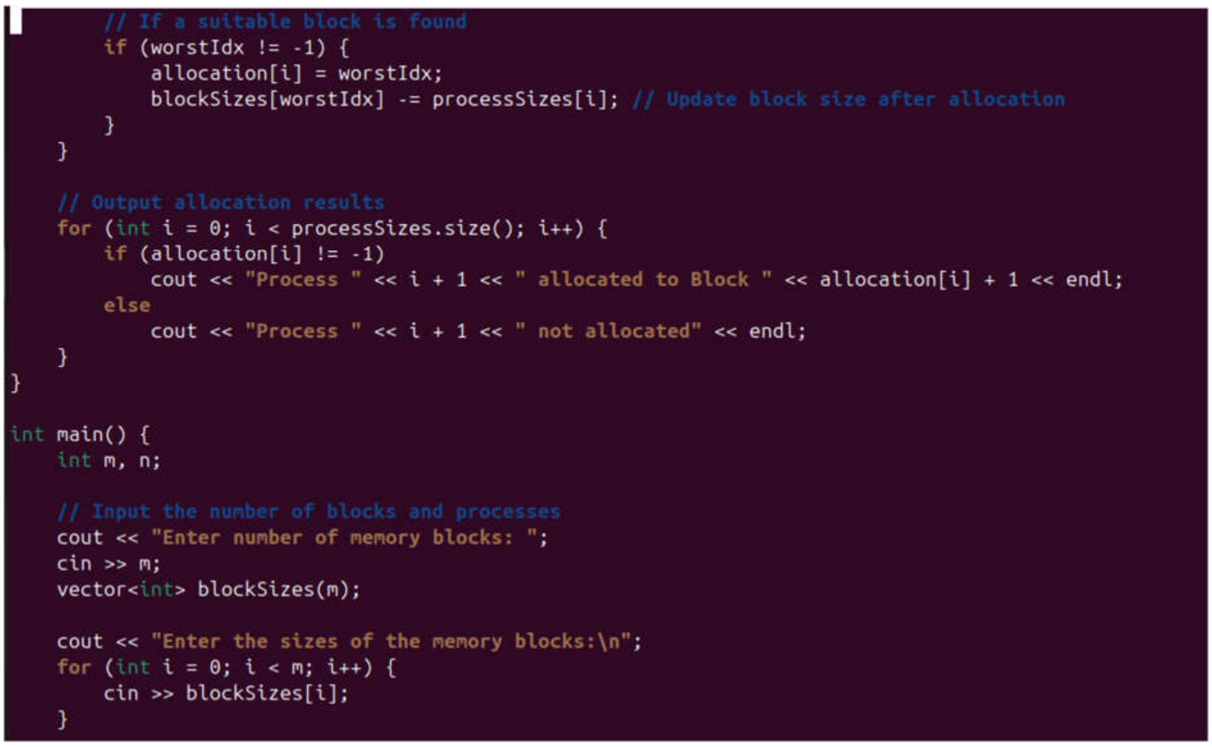
```
#include <iostream>
#include <vector>
#include <climits>

using namespace std;

// Function to implement First-Fit Allocation Strategy
void firstFit(vector<int>& blockSizes, vector<int>& processSizes) {
    cout << "\nFirst-Fit Allocation:" << endl;
    vector<int> allocation(processSizes.size(), -1); // -1 means no allocation

    for (int i = 0; i < processSizes.size(); i++) {
        for (int j = 0; j < blockSizes.size(); j++) {
            if (blockSizes[j] >= processSizes[i]) {
                allocation[i] = j;
                blockSizes[j] -= processSizes[i]; // Update block size after allocation
                break;
            }
        }
    }

    // Output allocation results
    for (int i = 0; i < processSizes.size(); i++) {
        if (allocation[i] != -1)
            cout << "Process " << i + 1 << " allocated to Block " << allocation[i] + 1 << endl;
        else
            cout << "Process " << i + 1 << " not allocated" << endl;
    }
                                                    [ Read 138 lines ]
```



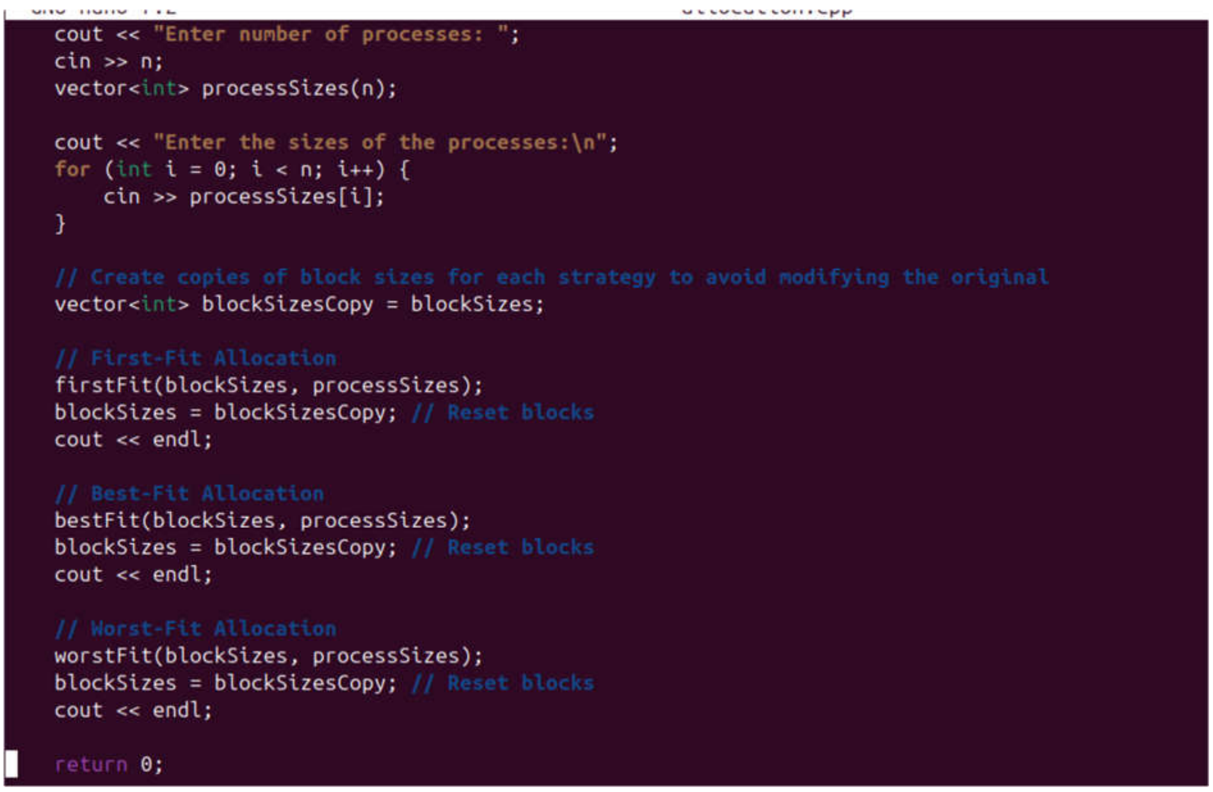
```
// Function to implement Best-Fit Allocation Strategy
void bestFit(vector<int>& blockSizes, vector<int>& processSizes) {
    cout << "\nBest-Fit Allocation:" << endl;
    vector<int> allocation(processSizes.size(), -1); // -1 means no allocation

    for (int i = 0; i < processSizes.size(); i++) {
        int bestIdx = -1;
        int minDiff = INT_MAX;

        // Find the best block for the current process
        for (int j = 0; j < blockSizes.size(); j++) {
            if (blockSizes[j] >= processSizes[i] && blockSizes[j] - processSizes[i] < minDiff) {
                minDiff = blockSizes[j] - processSizes[i];
                bestIdx = j;
            }
        }

        // If a suitable block is found
        if (bestIdx != -1) {
            allocation[i] = bestIdx;
            blockSizes[bestIdx] -= processSizes[i]; // Update block size after allocation
        }
    }

    // Output allocation results
    for (int i = 0; i < processSizes.size(); i++) {
        if (allocation[i] != -1)
```



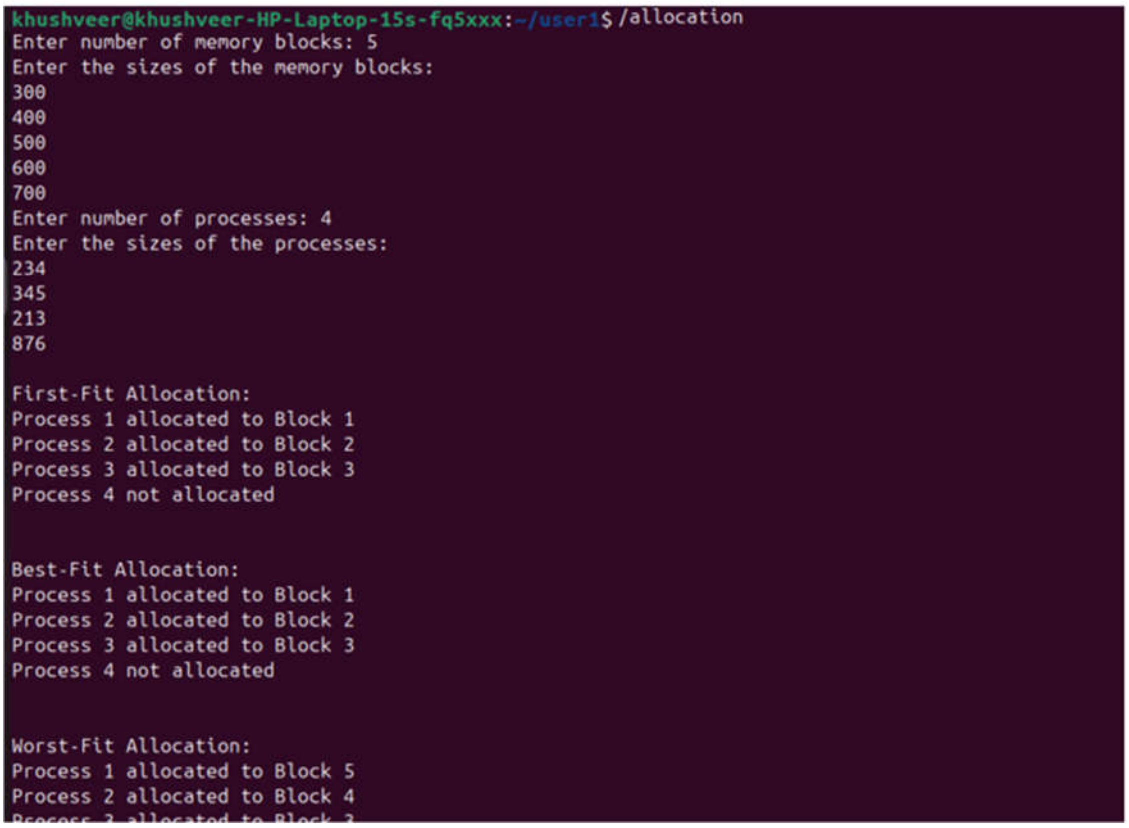
```
    }

    // Output allocation results
    for (int i = 0; i < processSizes.size(); i++) {
        if (allocation[i] != -1)
            cout << "Process " << i + 1 << " allocated to Block " << allocation[i] + 1 << endl;
        else
            cout << "Process " << i + 1 << " not allocated" << endl;
    }
}

/ Function to implement Worst-Fit Allocation Strategy
oid worstFit(vector<int>& blockSizes, vector<int>& processSizes) {
    cout << "\nWorst-Fit Allocation:" << endl;
    vector<int> allocation(processSizes.size(), -1); // -1 means no allocation

    for (int i = 0; i < processSizes.size(); i++) {
        int worstIdx = -1;
        int maxDiff = -1;

        // Find the worst block for the current process
        for (int j = 0; j < blockSizes.size(); j++) {
            if (blockSizes[j] >= processSizes[i] && blockSizes[j] - processSizes[i] > maxDiff) {
                maxDiff = blockSizes[j] - processSizes[i];
                worstIdx = j;
            }
        }
```



```
        // If a suitable block is found
        if (worstIdx != -1) {
            allocation[i] = worstIdx;
            blockSizes[worstIdx] -= processSizes[i]; // Update block size after allocation
        }
    }

    // Output allocation results
    for (int i = 0; i < processSizes.size(); i++) {
        if (allocation[i] != -1)
            cout << "Process " << i + 1 << " allocated to Block " << allocation[i] + 1 << endl;
        else
            cout << "Process " << i + 1 << " not allocated" << endl;
    }
}

int main() {
    int m, n;

    // Input the number of blocks and processes
    cout << "Enter number of memory blocks: ";
    cin >> m;
    vector<int> blockSizes(m);

    cout << "Enter the sizes of the memory blocks:\n";
    for (int i = 0; i < m; i++) {
        cin >> blockSizes[i];
    }
```



```
    cout << "Enter number of processes: ";
    cin >> n;
    vector<int> processSizes(n);

    cout << "Enter the sizes of the processes:\n";
    for (int i = 0; i < n; i++) {
        cin >> processSizes[i];
    }

    // Create copies of block sizes for each strategy to avoid modifying the original
    vector<int> blockSizesCopy = blockSizes;

    // First-Fit Allocation
    firstFit(blockSizes, processSizes);
    blockSizes = blockSizesCopy; // Reset blocks
    cout << endl;

    // Best-Fit Allocation
    bestFit(blockSizes, processSizes);
    blockSizes = blockSizesCopy; // Reset blocks
    cout << endl;

    // Worst-Fit Allocation
    worstFit(blockSizes, processSizes);
    blockSizes = blockSizesCopy; // Reset blocks
    cout << endl;

    return 0;
```



```
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1$ /allocation
Enter number of memory blocks: 5
Enter the sizes of the memory blocks:
300
400
500
600
700
Enter number of processes: 4
Enter the sizes of the processes:
234
345
213
876

First-Fit Allocation:
Process 1 allocated to Block 1
Process 2 allocated to Block 2
Process 3 allocated to Block 3
Process 4 not allocated


Best-Fit Allocation:
Process 1 allocated to Block 1
Process 2 allocated to Block 2
Process 3 allocated to Block 3
Process 4 not allocated


Worst-Fit Allocation:
Process 1 allocated to Block 5
Process 2 allocated to Block 4
```