OPERATING SYSTEM

PRACTICAL FILE

RAMANUJAN COLLEGE

UNIVERSITY OF DELHI

SUBMITTED BY :

NAME :Khushveer Kaur

ROLL NO. : 24570031

EXAMINATION ROLL NO : 24020570052

CLASS : BSC(H) Computer Science SEM-3

SUBMITTED TO:

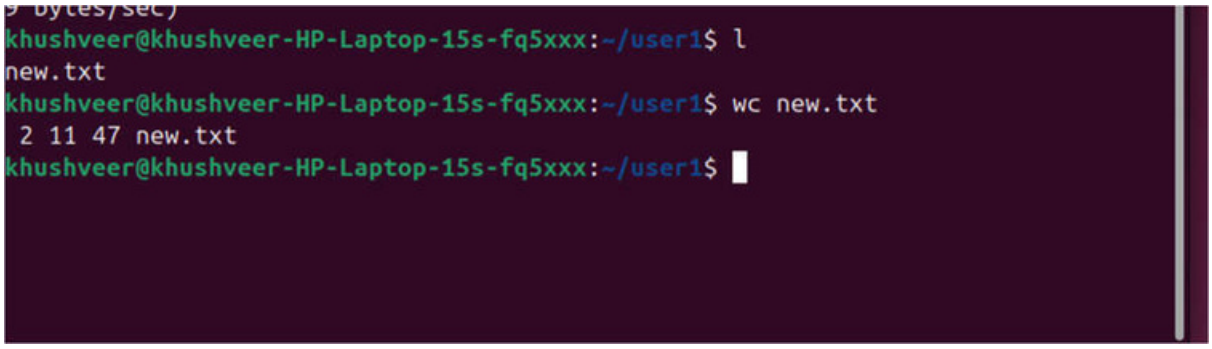
Mrs .Sheetal

ASSISTANT PROFESSOR, RAMANUJAN COLLEGE, UNIVERSITY OF DELHI, CR PARK MAIN ROAD,BLOCK H , KALKAJI NEW DELHI 110019

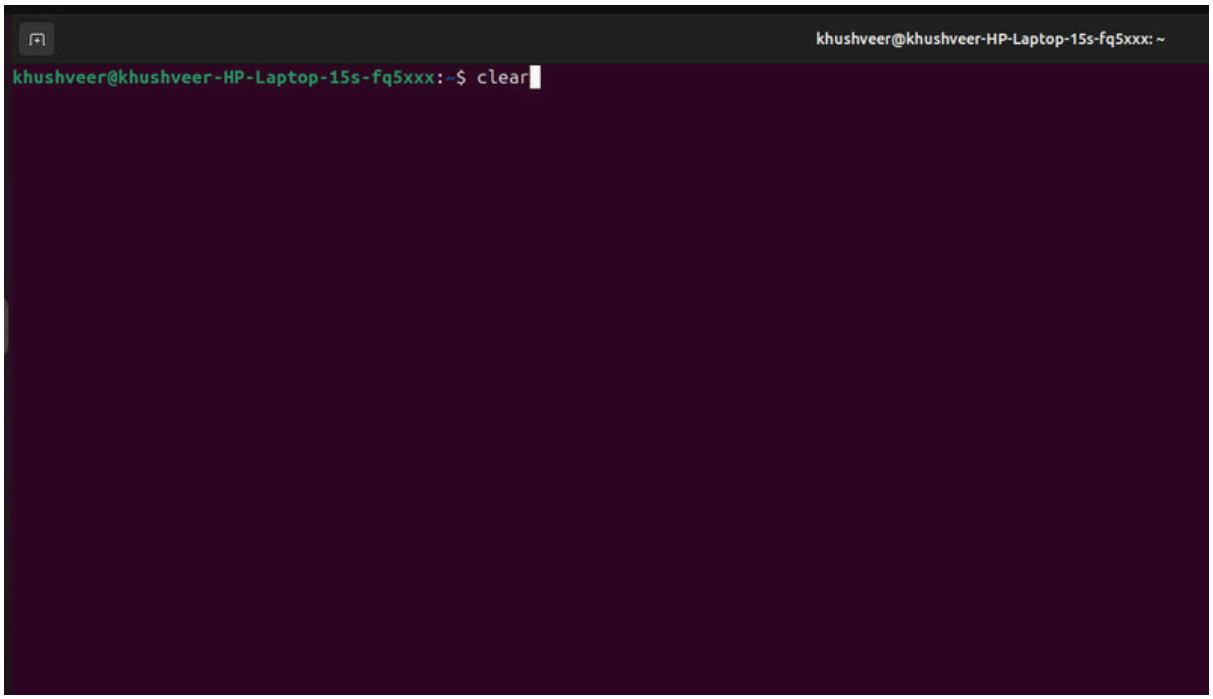
Q-1 :- Execute various LINUX commands for:

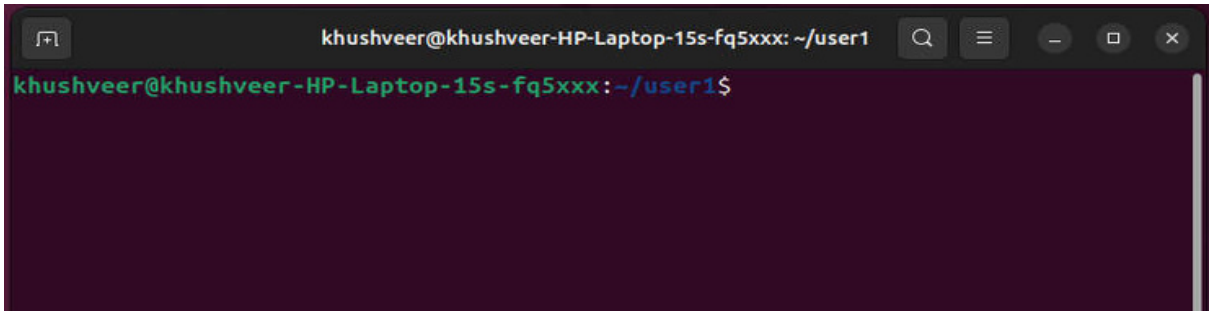
(i) informa on Maintenance: wc, clear, cal, who, date, pwd.

Wc(word count)

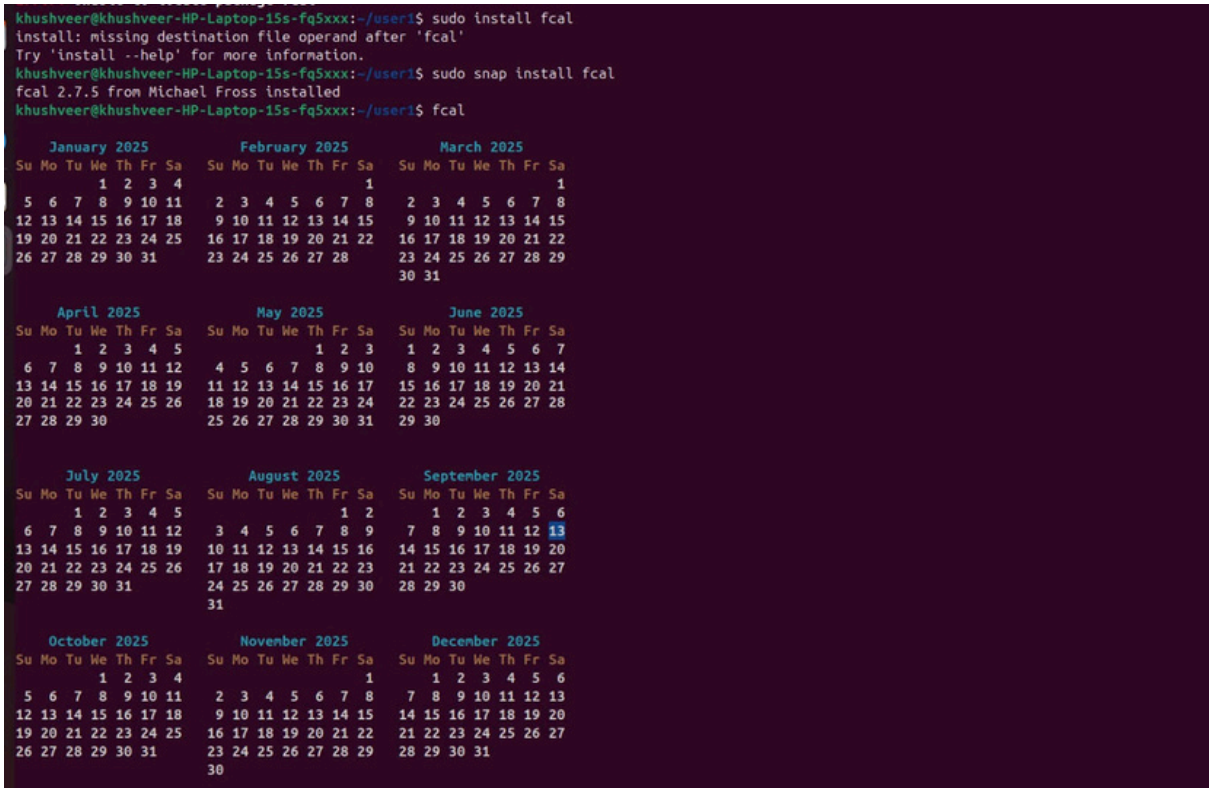


Clear

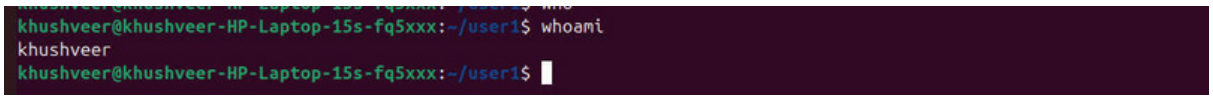

khushveer@khushveer-HP-Laptop-15s-fq5xxx: ~
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~$ clear


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)

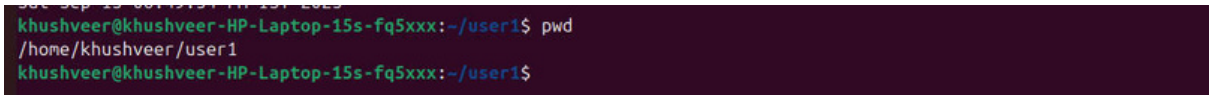


Who(current user)



Date

Pwd(print the current directory)



(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)

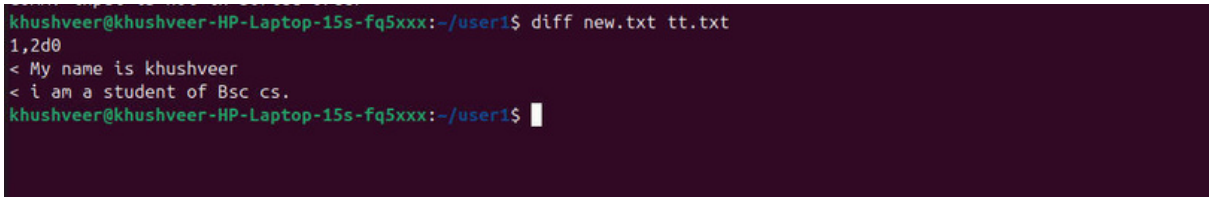
```
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1/user2$ touch hero.txt
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1/user2$ ls
hero.txt  new1.txt
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1/user2$ rm hero.txt
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1/user2$ ls
new1.txt
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/user1/user2$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

```
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$
```

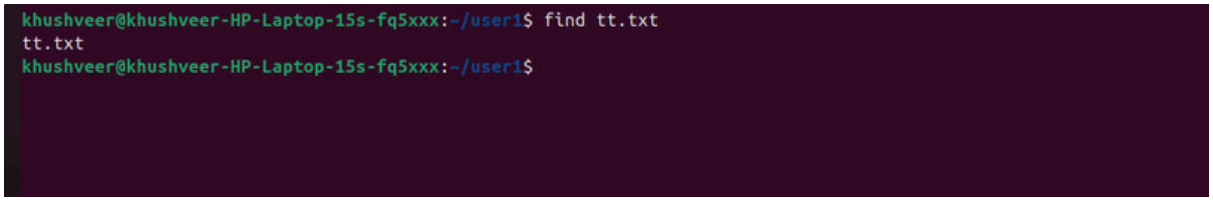
Comm(compare two sorted file line by line)

```
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)



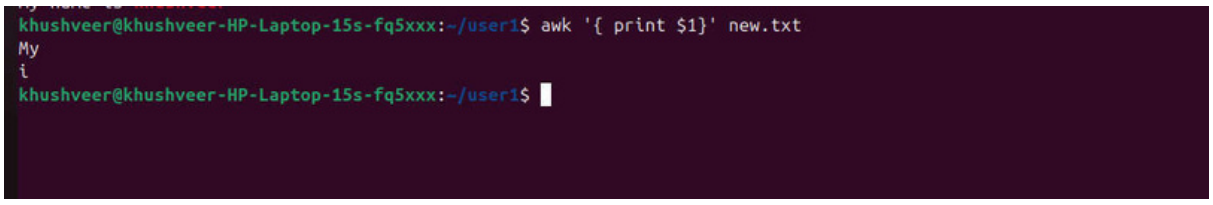
Find(find file)



Grep(search for text pa erns)

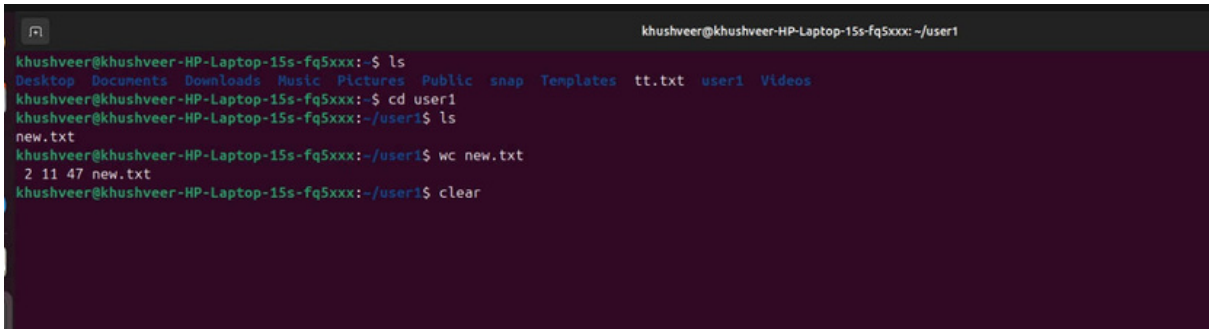
Awk(extract,format and proccess texts)

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



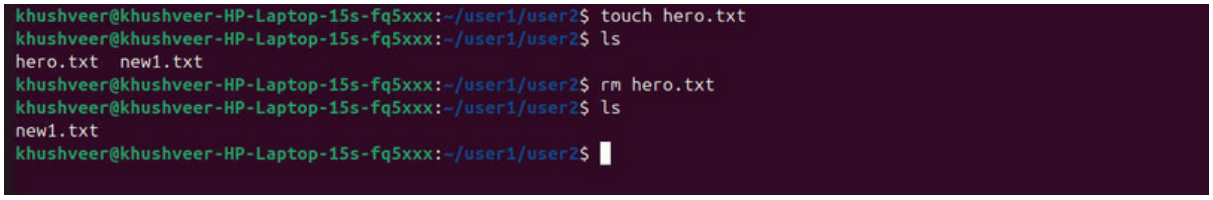
(Iii). Directory Management: cd, mkdir, rmdir, ls

Cd (change directory)



Mkdir(make directory)

Rmdir(remove directory)



Ls(list directories/files)

Q– 2 - Execute various LINUX commands for

i. Communica on: Input-output redirec on, Pipe

1. Input/Output Redirec on

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 redirec on

```
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 redirec on

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

--> error redirec on

```
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 communica on.

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

OR

Communication:

Input-output redirection

```
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/Downloads$ echo "Hello OS Lab" > file1.txt
cat file1.txt
Hello OS Lab
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/Downloads$ echo "Second line" >> file1.txt
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/Downloads$ cat file1.txt
Hello OS Lab
Second line
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/Downloads$ wc -l < file1.txt
2
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/Downloads$
```

Pipe , Protection

```
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/Downloads$ wc -l < file1.txt
2
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/Downloads$ ls | wc -l
8
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/Downloads$ ls -l file1.txt
-rw-rw-r-- 1 khushveer khushveer 25 Nov 30 17:40 file1.txt
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/Downloads$ chmod 000 file1.txt
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/Downloads$ ls -l file1.txt
---------- 1 khushveer khushveer 25 Nov 30 17:40 file1.txt
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/Downloads$ cat file1.txt
cat: file1.txt: Permission denied (os error 13)
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/Downloads$
```

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

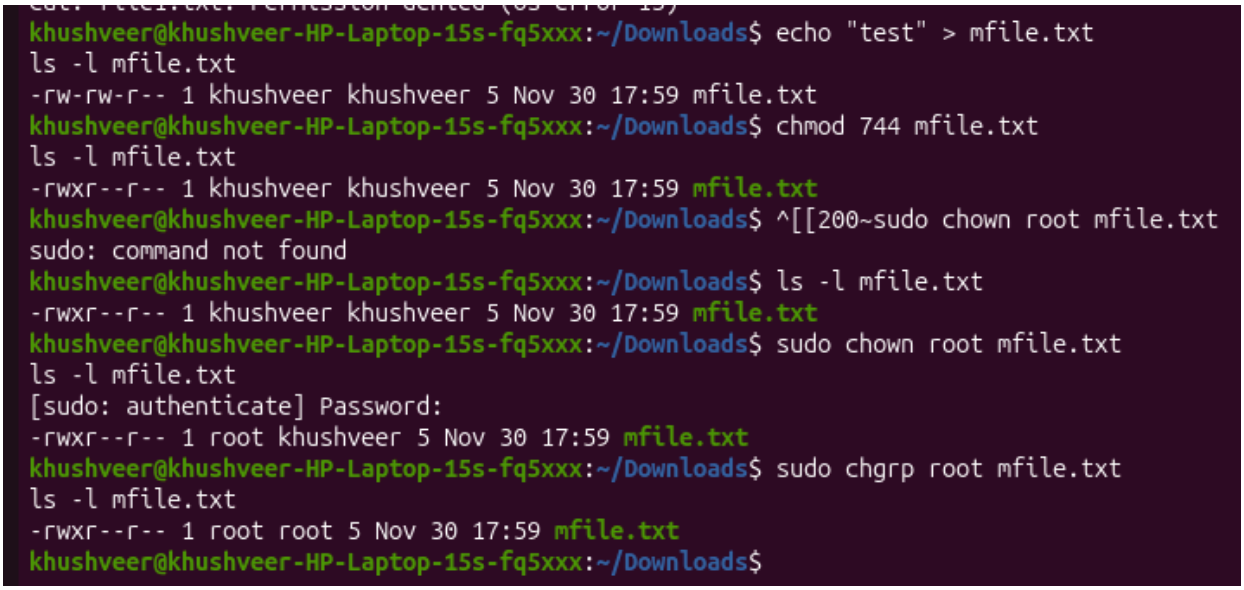
```
int main() {
    pid_t pid = fork();

    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$
```

```
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/Downloads$ g++ fork.cpp -o fork
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/Downloads$ ./fork
I am the single parent. About to fork...
Parent: Hello from the C++ PARENT process! (Child's PID: 6462)
Child: Hello from the C++ CHILD process! (PID: 6462)
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/Downloads$ sleep 1000 &
[1] 6482
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/Downloads$ sleep 5

khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/Downloads$
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/Downloads$ ps
    PID TTY          TIME CMD
   5179 pts/0    00:00:00 bash
   6482 pts/0    00:00:00 sleep
   6528 pts/0    00:00:00 ps
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/Downloads$ kill 6482
[1]+  Terminated              sleep 1000
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/Downloads$ ps
    PID TTY          TIME CMD
   5179 pts/0    00:00:00 bash
   6536 pts/0    00:00:00 ps
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/Downloads$
```

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.

```
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/Downloads$ nano parentwait.cpp
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/Downloads$ cat parentwait.cpp
#include <iostream>
#include <unistd.h>     // fork, sleep, getpid
#include <sys/types.h>  // pid_t
#include <sys/wait.h>   // wait

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);   // simulate some work
        cout << "Child finished\n";
    }
    else {
        wait(NULL); // parent waits for child to finish
        cout << "Parent waited for child. Parent PID: " << getpid() << endl;
    }

    return 0;
}
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/Downloads$ g++ parentwait.cpp -o parentwait
./parentwait
Child running... PID: 8031
Child finished
Parent waited for child. Parent PID: 8030
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/Downloads$
```

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

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

int main() {
    cout << "--- Kernel 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:~/Downloads$ g++ report.cpp -o report ./report
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/Downloads$ g++ report.cpp -o report
./report
--- Kernel information ---
Linux
6.17.0-6-generic
#6-Ubuntu SMP PREEMPT_DYNAMIC Tue Oct  7 13:34:17 UTC 2025
x86_64

--- CPU information ---
Architecture:                x86_64
  CPU op-mode(s):            32-bit, 64-bit
  Address sizes:             39 bits physical, 48 bits virtual
  Byte Order:                Little Endian
CPU(s):                      12
  On-line CPU(s) list:       0-11
Vendor ID:                   GenuineIntel
  Model name:                12th Gen Intel(R) Core(TM) i5-1235U
    CPU family:              6
    Model:                   154
    Thread(s) per core:      2
    Core(s) per socket:      10
    Socket(s):               1
    Stepping:                4
    CPU(s) scaling MHz:      60%
    CPU max MHz:             4400.0000
    CPU min MHz:             400.0000
    BogoMIPS:                4992.00
```

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

```
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/Downloads$ nano memory.cpp
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/Downloads$
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/Downloads$ cat memory.cpp
#include <fstream>
#include <string>
#include <iostream>
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:~/Downloads$ g++ memory.cpp -o memory
./memory
----memory information----
MemTotal:       15514076 kB
MemFree:         9153676 kB
MemAvailable:   11775572 kB
Buffers:           97428 kB
Cached:          3420956 kB
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/Downloads$
```

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

```
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/Downloads$ nano copy_syscalls.cpp
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/Downloads$ cat copy_syscalls.cpp

#include <iostream>
#include <fstream>
#include <unistd.h>     // For read(), write(), close()
#include <fcntl.h>      // For open() and file flags (O_RDONLY, O_WRONLY, etc.)
#include <sys/stat.h>   // For file permissions (S_IRUSR, S_IWUSR, etc.)
#include <errno.h>      // For the errno variable
#include <cstring>      // For strerror() function declaration

using namespace std;

// Define a reasonable buffer size for efficient copying (e.g., 4KB)
#define BUFFER_SIZE 4096

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

    const char* source_path = argv[1];
    const char* dest_path = argv[2];

    // --- 1. Open Source File for Reading ---
    // O_RDONLY: Read-only access
    int input_fd = open(source_path, O_RDONLY);

    if (input_fd < 0) {
        // If open fails, report error and exit
        cerr << "Error opening source file (" << source_path << "): " << strerror(errno) << endl;
        return 2;
    }

    // --- 2. Open/Create Destination File for Writing ---
    // O_WRONLY: Write-only access
    // O_CREAT: Create file if it doesn't exist
    // O_TRUNC: Truncate file to zero length if it exists
    // S_IRUSR | S_IWUSR: File permissions (User Read/Write)
    int output_fd = open(dest_path, O_WRONLY | O_CREAT | O_TRUNC, S_IRUSR | S_IWUSR);

    if (output_fd < 0) {
        // If open fails, report error and close the source file before exiting
        cerr << "Error opening/creating destination file (" << dest_path << "): " << strerror(errno) << endl;
        close(input_fd);
        return 3;
    }

    // --- 3. Copy Data in Chunks ---
    char buffer[BUFFER_SIZE];
    ssize_t bytes_read;
    ssize_t bytes_written;

    cout << "Starting copy from " << source_path << " to " << dest_path << "..." << endl;

    // Loop until the end of the input file is reached (read() returns 0)
    while ((bytes_read = read(input_fd, buffer, BUFFER_SIZE)) > 0) {
        // Write the chunk of data we just read to the output file
        bytes_written = write(output_fd, buffer, bytes_read);

        if (bytes_written != bytes_read) {
            // This handles cases where only a partial write occurred or an error happened
            cerr << "Error during writing. Only wrote " << bytes_written << " of " << bytes_read << " bytes." << endl;
            close(input_fd);
            close(output_fd);
            return 4;
        }
    }

    // Check if the loop terminated because of an error during reading
    if (bytes_read < 0) {
        cerr << "Error during reading source file: " << strerror(errno) << endl;
        close(input_fd);
        close(output_fd);
        return 5;
    }

    // --- 4. Close File Descriptors ---
    close(input_fd);
    close(output_fd);

    // The logic below was incorrect, as input_fd is a file descriptor number, not the count.
    // We cannot easily track total bytes copied using only system calls without a counter variable.
    // For simplicity, we just confirm success.
    cout << "Copy successful!" << endl;

    return 0;
}
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/Downloads$ ^[[200~g++ copy_syscalls.cpp -o copy_syscalls
g++: command not found
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/Downloads$ g++ copy_syscalls.cpp -o copy_syscalls
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/Downloads$ echo "hello os lab" > src.txt
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/Downloads$ ./copy_syscalls src.txt dst.txt
Starting copy from src.txt to dst.txt...
Copy successful!
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/Downloads$ cat dst.txt
hello os lab
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/Downloads$
```

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

```
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/Downloads$ nano fcfs.cpp
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/Downloads$ cat fcfs.cpp
#include <iostream>
#include <vector>
#include <algorithm>
#include <iomanip>

using namespace std;

// Simplified structure for the process
struct Job {
    int id;
    int arrival;
    int burst;
    int waiting_time;
    int turnaround_time;
};

void run_simple_fcfs(vector<Job>& jobs) {
    // FCFS rule: Sort by Arrival Time
    sort(jobs.begin(), jobs.end(), [](const Job& a, const Job& b) {
        return a.arrival < b.arrival;
    });

    int current_time = 0;
    double total_waiting = 0;
    double total_turnaround = 0;

    for (Job& j : jobs) {
        // 1. Calculate the time the job STARTS
        int start_time = max(current_time, j.arrival);

        // 2. Calculate Waiting Time
        // Waiting time is the time it was ready (start_time) minus when it arrived (j.arrival).
        j.waiting_time = start_time - j.arrival;
        total_waiting += j.waiting_time;

        // 3. Update Current Time (Completion Time)
        current_time = start_time + j.burst;

        // 4. Calculate Turnaround Time (Completion Time - Arrival Time)
        j.turnaround_time = current_time - j.arrival;
        total_turnaround += j.turnaround_time;
    }

    // Display Results
    cout << "--- Simple FCFS Results ---" << endl;
    cout << "ID | Arrival | Burst | Waiting | Turnaround" << endl;
    cout << "---------------------------------------" << endl;

    for (const auto& j : jobs) {
        cout << setw(2) << j.id << " |"
             << setw(7) << j.arrival << " |"
             << setw(5) << j.burst << " |"
             << setw(7) << j.waiting_time << " |"
             << setw(10) << j.turnaround_time << endl;
    }

    // Display Averages
    int n = jobs.size();
    cout << "\nAverage Waiting Time:    " << fixed << setprecision(2) << total_waiting / n << endl;
    cout << "Average Turnaround Time: " << fixed << setprecision(2) << total_turnaround / n << endl;
}

int main() {
    // Example Jobs: (ID, Arrival Time, Burst Time)
    vector<Job> jobs = {
        {1, 0, 10},
        {2, 1, 5},
        {3, 2, 2}
    };

    run_simple_fcfs(jobs);

    return 0;
}
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/Downloads$ g++ fcfs.cpp -o fcfs
./fcfs
--- Simple FCFS Results ---
ID | Arrival | Burst | Waiting | Turnaround
---------------------------------------
 1 |       0 |    10 |       0 |         10
 2 |       1 |     5 |       9 |         14
 3 |       2 |     2 |      13 |         15

Average Waiting Time:    7.33
Average Turnaround Time: 13.00
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/Downloads$
```

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

```
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/Downloads$ nano sjf.cpp
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/Downloads$ cat sjf.cpp
#include <iostream>
#include <vector>
#include <algorithm>
#include <iomanip>
#include <limits> // Used for numeric_limits<int>::max()

using namespace std;

// Simplified structure for the process
struct Job {
    int id;
    int arrival;
    int burst;
    int waiting_time;
    int turnaround_time;
    bool is_completed = false; // Flag to track completion
};

void run_simple_sjf(vector<Job>& jobs) {
    int n = jobs.size();
    int current_time = 0;
    int completed_count = 0;
    double total_waiting = 0;
    double total_turnaround = 0;

    // We process jobs based on arrival AND burst time
    while (completed_count < n) {
        int shortest_burst = numeric_limits<int>::max();
        int shortest_job_index = -1;

        // 1. Scan for the best job: Find the shortest job that has arrived
        for (int i = 0; i < n; ++i) {
            if (jobs[i].arrival <= current_time && !jobs[i].is_completed) {
                if (jobs[i].burst < shortest_burst) {
                    shortest_burst = jobs[i].burst;
                    shortest_job_index = i;
                }
            }
        }

        if (shortest_job_index != -1) {
            // A job is ready to run (Shortest Job First)
            Job& j = jobs[shortest_job_index];

            // 2. Calculate Start Time (it starts now)
            int start_time = current_time;

            // 3. Update Current Time (Completion Time)
            current_time = start_time + j.burst;
            j.is_completed = true;
```

```
            // 3. Update Current Time (Completion Time)
            current_time = start_time + j.burst;
            j.is_completed = true;
            completed_count++;

            // 4. Calculate Metrics
            j.turnaround_time = current_time - j.arrival;
            j.waiting_time = j.turnaround_time - j.burst;

            total_turnaround += j.turnaround_time;
            total_waiting += j.waiting_time;

        } else {
            // If no process is ready (CPU is idle), advance time
            current_time++;
        }
    }

    // Sort the results back by ID for clean display
    sort(jobs.begin(), jobs.end(), [](const Job& a, const Job& b) {
        return a.id < b.id;
    });

    // Display Results
    cout << "--- Non-Preemptive SJF Results ---" << endl;
    cout << "ID | Arrival | Burst | Waiting | Turnaround" << endl;
    cout << "----------------------------------------" << endl;

    for (const auto& j : jobs) {
        cout << setw(2) << j.id << " |"
             << setw(7) << j.arrival << " |"
             << setw(5) << j.burst << " |"
             << setw(7) << j.waiting_time << " |"
             << setw(10) << j.turnaround_time << endl;
    }

    cout << "\nAverage Waiting Time:    " << fixed << setprecision(2) << total_waiting / n << endl;
    cout << "Average Turnaround Time: " << fixed << setprecision(2) << total_turnaround / n << endl;
}

int main() {
    // Example Jobs: (ID, Arrival Time, Burst Time)
    // The scheduler should run P1, then P2, then P3, then P4.
    vector<Job> jobs = {
        {1, 0, 7},
        {2, 2, 4},
        {3, 4, 1}, // This will run second because of the shortest burst time
        {4, 5, 4}
    };

    run_simple_sjf(jobs);

    return 0;
}
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/Downloads$ g++ sjf.cpp -o sjf
./sjf
--- Non-Preemptive SJF Results ---
ID | Arrival | Burst | Waiting | Turnaround
----------------------------------------
 1 |       0 |     7 |       0 |          7
 2 |       2 |     4 |       6 |         10
 3 |       4 |     1 |       3 |          4
 4 |       5 |     4 |       7 |         11

Average Waiting Time:    4.00
Average Turnaround Time: 8.00
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/Downloads$
```

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

```
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/Downloads$ nano priority.cpp
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/Downloads$ cat priority.cpp
#include <iostream>
#include <vector>
#include <algorithm>
#include <iomanip>
#include <limits> // Used for numeric_limits<int>::max()

using namespace std;

// Structure includes priority field
struct Job {
    int id;
    int arrival;
    int burst;
    int priority;    // Lower number = Higher Priority (Common convention)
    int waiting_time;
    int turnaround_time;
    bool is_completed = false; // Flag to track completion
};

void run_simple_priority(vector<Job>& jobs) {
    int n = jobs.size();
    int current_time = 0;
    int completed_count = 0;
    double total_waiting = 0;
    double total_turnaround = 0;

    // We process jobs based on arrival AND priority
    while (completed_count < n) {
        int highest_priority = numeric_limits<int>::max();
        int highest_priority_index = -1;

        // 1. Scan for the best job: Find the job with the highest priority that has arrived
        for (int i = 0; i < n; ++i) {
            if (jobs[i].arrival <= current_time && !jobs[i].is_completed) {
                // Find the process with the smallest priority number
                if (jobs[i].priority < highest_priority) {
                    highest_priority = jobs[i].priority;
                    highest_priority_index = i;
                }
            }
        }

        if (highest_priority_index != -1) {
            // A job is ready to run (Highest Priority First)
            Job& j = jobs[highest_priority_index];

            // 2. Calculate Start Time (it starts now)
            int start_time = current_time;

            // 3. Update Current Time (Completion Time)
```

```
            // 3. Update Current Time (Completion Time)
            current_time = start_time + j.burst;
            j.is_completed = true;
            completed_count++;

            // 4. Calculate Metrics
            j.turnaround_time = current_time - j.arrival;
            j.waiting_time = j.turnaround_time - j.burst;

            total_turnaround += j.turnaround_time;
            total_waiting += j.waiting_time;

        } else {
            // If no process is ready (CPU is idle), advance time
            current_time++;
        }
    }

    // Sort the results back by ID for clean display
    sort(jobs.begin(), jobs.end(), [](const Job& a, const Job& b) {
        return a.id < b.id;
    });

    // Display Results
    cout << "--- Non-Preemptive Priority Results (Low Number = High Priority) ---" << endl;
    cout << "ID | Arrival | Burst | Pri | Waiting | Turnaround" << endl;
    cout << "-----------------------------------------------" << endl;

    for (const auto& j : jobs) {
        cout << setw(2) << j.id << " |"
             << setw(7) << j.arrival << " |"
             << setw(5) << j.burst << " |"
             << setw(3) << j.priority << " |"
             << setw(7) << j.waiting_time << " |"
             << setw(10) << j.turnaround_time << endl;
    }

    cout << "\nAverage Waiting Time:    " << fixed << setprecision(2) << total_waiting / n << endl;
    cout << "Average Turnaround Time: " << fixed << setprecision(2) << total_turnaround / n << endl;
}

int main() {
    // Example Jobs: (ID, Arrival Time, Burst Time, Priority)
    // P2 has the highest priority (1) and should run first after P1, regardless of its burst time.
    vector<Job> jobs = {
        {1, 0, 10, 3}, // Priority 3
        {2, 1, 5, 1},  // Priority 1 (Highest)
        {3, 2, 2, 4},  // Priority 4 (Lowest)
        {4, 3, 3, 2}   // Priority 2
    };

    run_simple_priority(jobs);

    return 0;
}
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/Downloads$ g++ priority.cpp -o priority
./priority
--- Non-Preemptive Priority Results (Low Number = High Priority) ---
ID | Arrival | Burst | Pri | Waiting | Turnaround
-----------------------------------------------
 1 |       0 |    10 |   3 |       0 |         10
 2 |       1 |     5 |   1 |       9 |         14
 3 |       2 |     2 |   4 |      16 |         18
 4 |       3 |     3 |   2 |      12 |         15

Average Waiting Time:    9.25
Average Turnaround Time: 14.25
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/Downloads$
```

10. Write a program to implement SRTF scheduling algorithm.

```
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/Downloads$ nano srtf.cpp
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/Downloads$ cat srtf.cpp
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

struct Process {
    int pid;
    int burst_time;
    int remaining_time;
    int arrival_time;
    int start_time;
    int completion_time;
    int waiting_time;
    int turnaround_time;

    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) {}
};

void SRTFScheduling(vector<Process> &p) {
    int n = p.size();
    int current_time = 0;
    int completed = 0;

    // sort by arrival time
    sort(p.begin(), p.end(),
         [](const Process &a, const Process &b) {
             return a.arrival_time < b.arrival_time;
         });

    vector<bool> done(n, false);

    while (completed < n) {
        int idx = -1;
        int min_rem = 1e9;

        // pick arrived process with smallest remaining time
        for (int i = 0; i < n; ++i) {
            if (!done[i] &&
                p[i].arrival_time <= current_time &&
                p[i].remaining_time < min_rem &&
                p[i].remaining_time > 0) {
                min_rem = p[i].remaining_time;
                idx = i;
```

```
                min_rem = p[i].remaining_time;
                idx = i;
            }
        }

        if (idx == -1) {
            // no process ready, CPU idle
            current_time++;
            continue;
        }

        // start time for first execution
        if (p[idx].start_time == -1)
            p[idx].start_time = current_time;

        // execute 1 time unit (preemptive)
        p[idx].remaining_time--;
        current_time++;

        // if finished
        if (p[idx].remaining_time == 0) {
            done[idx] = true;
            completed++;

            p[idx].completion_time = current_time;
            p[idx].turnaround_time =
                p[idx].completion_time - p[idx].arrival_time;
            p[idx].waiting_time =
                p[idx].turnaround_time - p[idx].burst_time;
        }
    }

    // print table
    cout << "PID\tBurst\tArrival\tCompletion\tWaiting\tTurnaround\n";
    int total_wt = 0, total_tat = 0;
    for (const auto &pr : p) {
        cout << pr.pid << "\t"
             << pr.burst_time << "\t"
             << pr.arrival_time << "\t"
             << pr.completion_time << "\t\t"
             << pr.waiting_time << "\t"
             << pr.turnaround_time << "\n";
        total_wt += pr.waiting_time;
        total_tat += pr.turnaround_time;
    }
    cout << "Average Waiting Time: "
         << (float)total_wt / n << "\n";
    cout << "Average Turnaround Time: "
         << (float)total_tat / n << "\n";
}

int main() {
    // example set (PID, Burst, Arrival)
    vector<Process> procs = {
        Process(1, 6, 0),
        Process(2, 8, 1),
        Process(3, 7, 2),
        Process(4, 3, 3)
    };

    SRTFScheduling(procs);
    return 0;
}
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/Downloads$ g++ srtf.cpp -o srtf
./srtf
PID     Burst   Arrival Completion      Waiting Turnaround
1       6       0       6               0       6
2       8       1       24              15      23
3       7       2       16              7       14
4       3       3       9               3       6
Average Waiting Time: 6.25
Average Turnaround Time: 12.25
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/Downloads$
```

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.

```
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/Downloads$ nano pthread.cpp
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/Downloads$ cat pthread.cpp
#include <iostream>
#include <vector>
#include <pthread.h>
#include <numeric>
#include <cmath>

using namespace std;

// Define the number of threads to use
#define NUM_THREADS 4

// Global data structure to hold thread-specific information
struct ThreadData {
    int thread_id;          // Identifier for the thread
    int start_index;        // Starting index in the array for this thread
    int end_index;          // Ending index (exclusive)
    const vector<int>* array; // Pointer to the shared array
    long long partial_sum;  // Thread's calculated sum
};

// Global array and total size
vector<int> numbers;
int array_size = 0;

// Function executed by each thread
void* calculate_partial_sum(void* arg) {
    ThreadData* data = (ThreadData*)arg;
    long long sum = 0;

    for (int i = data->start_index; i < data->end_index; ++i) {
        sum += (*data->array)[i];
    }

    data->partial_sum = sum;
    pthread_exit(NULL);
}

int main() {
    array_size = 200000; // Example: Sum 200,000 numbers

    // Initialize the array with simple values (e.g., all 1s)
    numbers.resize(array_size);
    iota(numbers.begin(), numbers.end(), 1); // Fill with 1, 2, 3, ...

    pthread_t threads[NUM_THREADS];
    ThreadData thread_data[NUM_THREADS];
    int chunk_size = array_size / NUM_THREADS;
    long long final_total_sum = 0;

    cout << "Starting multi-threaded sum calculation for " << array_size << " numbers using " << NUM_THREADS << " threads." << endl;
```

```
    cout << "Starting multi-threaded sum calculation for " << array_size << " numbers using " << NUM_THREADS << " threads." << endl;

    // 1. Create and Launch Threads
    for (int i = 0; i < NUM_THREADS; ++i) {
        thread_data[i].thread_id = i;
        thread_data[i].array = &numbers;
        thread_data[i].start_index = i * chunk_size;

        // Ensure the last thread handles any remainder
        if (i == NUM_THREADS - 1) {
            thread_data[i].end_index = array_size;
        } else {
            thread_data[i].end_index = (i + 1) * chunk_size;
        }

        cout << "  Thread " << i << " created: processing indices "
             << thread_data[i].start_index << " to "
             << thread_data[i].end_index - 1 << endl;

        // Create the thread and tell it to run calculate_partial_sum
        int rc = pthread_create(&threads[i], NULL, calculate_partial_sum, (void*)&thread_data[i]);
        if (rc) {
            cerr << "Error: unable to create thread, " << rc << endl;
            return 1;
        }
    }

    // 2. Wait for all threads to complete (Join)
    for (int i = 0; i < NUM_THREADS; ++i) {
        pthread_join(threads[i], NULL);
    }

    cout << "\nAll threads completed execution." << endl;

    // 3. Combine Partial Sums
    for (int i = 0; i < NUM_THREADS; ++i) {
        final_total_sum += thread_data[i].partial_sum;
    }

    // 4. Verification (Sum of 1 to N is N*(N+1)/2)
    long long verification_sum = (long long)array_size * ((long long)array_size + 1) / 2;

    cout << "----------------------------------------" << endl;
    cout << "Calculated Total Sum: " << final_total_sum << endl;
    cout << "Verification Sum:     " << verification_sum << endl;

    if (final_total_sum == verification_sum) {
        cout << "Result Verified: Success!" << endl;
    } else {
        cerr << "Result Error: Sums do not match." << endl;
    }

    return 0;
}
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/Downloads$ g++ pthread.cpp -o pthread -lpthread
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/Downloads$ ./pthread
Starting multi-threaded sum calculation for 200000 numbers using 4 threads.
  Thread 0 created: processing indices 0 to 49999
  Thread 1 created: processing indices 50000 to 99999
  Thread 2 created: processing indices 100000 to 149999
  Thread 3 created: processing indices 150000 to 199999

All threads completed execution.
----------------------------------------
Calculated Total Sum: 20000100000
Verification Sum:     20000100000
Result Verified: Success!
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/Downloads$
```

```
    // 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 alloca on strategies.

```
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/Downloads$ nano allocation.cpp
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/Downloads$ cat a;;ocation.cpp
bash: syntax error near unexpected token `;;'
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/Downloads$ cat allocation.cpp
#include <iostream>
#include <vector>
#include <algorithm>
#include <iomanip>
#include <numeric>
#include <limits>

using namespace std;

// Represents a memory block (partition)
struct Partition {
    int id;
    int size;
    bool is_occupied;
    int process_id;
};

// Represents a process requesting memory
struct Process {
    int id;
    int size;
    // We keep track of allocation state externally for this simulation to reuse the Process list
};

// --- Helper Function to print allocation status ---
void print_status(const vector<Partition>& partitions, const string& strategy) {
    cout << "\n==========================================" << endl;
    cout << "--- " << strategy << " Results ---" << endl;
    cout << "==========================================" << endl;
    cout << left << setw(10) << "Block ID"
         << setw(13) << "Block Size"
         << setw(10) << "Status"
         << setw(10) << "Process ID" << endl;
    cout << setfill('-') << setw(53) << "" << setfill(' ') << endl;

    for (const auto& p : partitions) {
        cout << left << setw(10) << p.id
             << setw(13) << to_string(p.size) + " KB"
             << setw(10) << (p.is_occupied ? "USED" : "FREE")
             << setw(10) << (p.is_occupied ? to_string(p.process_id) : "-") << endl;
    }
}

// --- 1. First-Fit Strategy ---
void first_fit(vector<Partition> partitions, const vector<Process>& processes) {
    int allocated_count = 0;

    for (const Process& proc : processes) {
        for (int i = 0; i < partitions.size(); ++i) {
```

```
    for (const Process& proc : processes) {
        for (int i = 0; i < partitions.size(); ++i) {
            Partition& block = partitions[i];

            // Check 1: Is the block free?
            // Check 2: Is the block large enough?
            if (!block.is_occupied && block.size >= proc.size) {
                // Allocate the first suitable block found
                block.is_occupied = true;
                block.process_id = proc.id;
                allocated_count++;

                // Allocation is done for this process, move to the next process
                break;
            }
        }
    }

    print_status(partitions, "First-Fit Allocation");
    cout << "Total Processes Allocated: " << allocated_count << " out of " << processes.size() << endl;
}

// --- 2. Best-Fit Strategy ---
void best_fit(vector<Partition> partitions, const vector<Process>& processes) {
    int allocated_count = 0;

    for (const Process& proc : processes) {
        int best_block_index = -1;
        // Start with the largest possible waste value
        int min_waste = numeric_limits<int>::max();

        for (int i = 0; i < partitions.size(); ++i) {
            Partition& block = partitions[i];

            // Check if the block is free AND big enough
            if (!block.is_occupied && block.size >= proc.size) {
                int waste = block.size - proc.size;

                // Find the block that leaves the smallest internal fragmentation (waste)
                if (waste < min_waste) {
                    min_waste = waste;
                    best_block_index = i;
                }
            }
        }

        if (best_block_index != -1) {
            // Allocate the block that is the "closest fit"
            partitions[best_block_index].is_occupied = true;
            partitions[best_block_index].process_id = proc.id;
            allocated_count++;
        }
```

```
            allocated_count++;
        }
    }

    print_status(partitions, "Best-Fit Allocation");
    cout << "Total Processes Allocated: " << allocated_count << " out of " << processes.size() << endl;
}

// --- 3. Worst-Fit Strategy ---
void worst_fit(vector<Partition> partitions, const vector<Process>& processes) {
    int allocated_count = 0;

    for (const Process& proc : processes) {
        int worst_block_index = -1;
        // Start with a small negative value
        int max_waste = -1;

        for (int i = 0; i < partitions.size(); ++i) {
            Partition& block = partitions[i];

            // Check if the block is free AND big enough
            if (!block.is_occupied && block.size >= proc.size) {
                int waste = block.size - proc.size;

                // Find the block that leaves the largest internal fragmentation (waste)
                if (waste > max_waste) {
                    max_waste = waste;
                    worst_block_index = i;
                }
            }
        }

        if (worst_block_index != -1) {
            // Allocate the block that is the "worst fit" (largest block)
            partitions[worst_block_index].is_occupied = true;
            partitions[worst_block_index].process_id = proc.id;
            allocated_count++;
        }
    }

    print_status(partitions, "Worst-Fit Allocation");
    cout << "Total Processes Allocated: " << allocated_count << " out of " << processes.size() << endl;
}


int main() {
    // Initial memory blocks (partitions)
    vector<Partition> initial_partitions = {
        {1, 100, false, -1}, // Block 1: 100 KB
        {2, 500, false, -1}, // Block 2: 500 KB
        {3, 200, false, -1}, // Block 3: 200 KB
        {4, 300, false, -1}, // Block 4: 300 KB
        {5, 600, false, -1}  // Block 5: 600 KB
```

```
    vector<Partition> initial_partitions = {
        {1, 100, false, -1}, // Block 1: 100 KB
        {2, 500, false, -1}, // Block 2: 500 KB
        {3, 200, false, -1}, // Block 3: 200 KB
        {4, 300, false, -1}, // Block 4: 300 KB
        {5, 600, false, -1}  // Block 5: 600 KB
    };

    // Processes requesting memory (KB)
    vector<Process> processes_to_allocate = {
        {101, 212}, // P1: Needs 212 KB
        {102, 417}, // P2: Needs 417 KB
        {103, 112}, // P3: Needs 112 KB
        {104, 400}  // P4: Needs 400 KB
    };

    cout << "--- Memory Allocation Simulation Start ---" << endl;
    cout << "Initial Memory Blocks (KB): 100, 500, 200, 300, 600" << endl;
    cout << "Processes (KB): P1 (212), P2 (417), P3 (112), P4 (400)" << endl;


    // Run all three strategies sequentially, passing a fresh copy of partitions each time
    first_fit(initial_partitions, processes_to_allocate);
    best_fit(initial_partitions, processes_to_allocate);
    worst_fit(initial_partitions, processes_to_allocate);

    return 0;
}
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/Downloads$ g++ allocation.cpp -o allocation
./allocation
--- Memory Allocation Simulation Start ---
Initial Memory Blocks (KB): 100, 500, 200, 300, 600
Processes (KB): P1 (212), P2 (417), P3 (112), P4 (400)

================================================
--- First-Fit Allocation Results ---
================================================
Block ID  Block Size   Status    Process ID
------------------------------------------------
1         100 KB       FREE      -
2         500 KB       USED      101
3         200 KB       USED      103
4         300 KB       FREE      -
5         600 KB       USED      102
Total Processes Allocated: 3 out of 4

================================================
--- Best-Fit Allocation Results ---
================================================
Block ID  Block Size   Status    Process ID
------------------------------------------------
1         100 KB       FREE      -
2         500 KB       USED      102
3         200 KB       USED      103
```

```
    cout << "Processes (KB): P1 (212), P2 (417), P3 (112), P4 (400)" << endl;


    // Run all three strategies sequentially, passing a fresh copy of partitions each time
    first_fit(initial_partitions, processes_to_allocate);
    best_fit(initial_partitions, processes_to_allocate);
    worst_fit(initial_partitions, processes_to_allocate);

    return 0;
}
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/Downloads$ g++ allocation.cpp -o allocation
./allocation
--- Memory Allocation Simulation Start ---
Initial Memory Blocks (KB): 100, 500, 200, 300, 600
Processes (KB): P1 (212), P2 (417), P3 (112), P4 (400)

================================================
--- First-Fit Allocation Results ---
================================================
Block ID  Block Size   Status    Process ID
------------------------------------------------
1         100 KB       FREE      -
2         500 KB       USED      101
3         200 KB       USED      103
4         300 KB       FREE      -
5         600 KB       USED      102
Total Processes Allocated: 3 out of 4

================================================
--- Best-Fit Allocation Results ---
================================================
Block ID  Block Size   Status    Process ID
------------------------------------------------
1         100 KB       FREE      -
2         500 KB       USED      102
3         200 KB       USED      103
4         300 KB       USED      101
5         600 KB       USED      104
Total Processes Allocated: 4 out of 4

================================================
--- Worst-Fit Allocation Results ---
================================================
Block ID  Block Size   Status    Process ID
------------------------------------------------
1         100 KB       FREE      -
2         500 KB       USED      102
3         200 KB       FREE      -
4         300 KB       USED      103
5         600 KB       USED      101
Total Processes Allocated: 3 out of 4
khushveer@khushveer-HP-Laptop-15s-fq5xxx:~/Downloads$
```