RAMANUJAN COLLEGE

UNIVERSITY OF DELHI

SOUTH CAMPUS IN KALKAJI 110019

PRACTICAL FILE

Operating System

Submitted By: Yuvraj Singh

Examination Roll Number: 23020570056

Class Roll Number: 20231471

Submitted to: Ms. Sheetal Singh

Acknowledgment

I would like to express my sincere gratitude to my instructor, Ms. Sheetal Singh for providing me with the opportunity to work on this series of practical. Their invaluable guidance and support have been instrumental in understanding the core concepts of Operating System

I would also like to thank Ramanujan College, University of Delhi, for offering such a rich curriculum that fosters technical learning and hands-on experience. The practical applications of various algorithms and transformation techniques have deepened my knowledge in the field.

Furthermore, I would like to thank my peers for their encouragement and collaboration during the course of this project. Their inputs have enhanced my learning experience and contributed to a stimulating environment for problem-solving.

Index

P1. Execute various LINUX commands for:

i. Information Maintenance: wc, clear, cal, who, date, pwd

```
uv_goswami@Ubuntu:~/os$ wc temp.txt
 1  5 21 temp.txt
uv_goswami@Ubuntu:~/os$ cal
   November 2024
Su Mo Tu We Th Fr Sa
                1  2
 3  4  5  6  7  8  9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30

uv_goswami@Ubuntu:~/os$ who
uv_goswami tty2         2024-11-25 03:33 (tty2)
uv_goswami@Ubuntu:~/os$ date
Monday 25 November 2024 04:20:15 AM IST
uv_goswami@Ubuntu:~/os$ pwd
/home/uv_goswami/os
uv_goswami@Ubuntu:~/os$ clear
```

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

```
uv_goswami@Ubuntu:~/os$ cat temp.txt
This is a test file.
uv_goswami@Ubuntu:~/os$ cp temp.txt copy.txt
uv_goswami@Ubuntu:~/os$ ls
copy.txt  delete.txt  temp.txt
uv_goswami@Ubuntu:~/os$ rm delete.txt
uv_goswami@Ubuntu:~/os$ ls
copy.txt  temp.txt
uv_goswami@Ubuntu:~/os$ mv copy.txt duplicate.txt
uv_goswami@Ubuntu:~/os$ ls
duplicate.txt  temp.txt
uv_goswami@Ubuntu:~/os$ comm temp.txt duplicate.txt
                This is a test file.
uv_goswami@Ubuntu:~/os$ diff temp.txt duplicate.txt
uv_goswami@Ubuntu:~/os$ find ./ -name temp.txt
./temp.txt
uv_goswami@Ubuntu:~/os$ grep 'i' temp.txt
This is a test file.
uv_goswami@Ubuntu:~/os$ awk '{print $2, $3}' temp.txt
is a
```

iii. Directory Management: cd, mkdir, rmdir, ls

```
uv_goswami@Ubuntu:~$ mkdir os
uv_goswami@Ubuntu:~$ cd os
uv_goswami@Ubuntu:~/os$ mkdir temp
uv_goswami@Ubuntu:~/os$ ls
temp
uv_goswami@Ubuntu:~/os$ rmdir temp
uv_goswami@Ubuntu:~/os$ ls
uv_goswami@Ubuntu:~/os$
```

P2. Execute various LINUX commands for:

i. Process Control: fork, getpid, ps, kill,

```cpp
#include <iostream>
#include <unistd.h>
#include <sys/wait.h>
using namespace std;

int main() {
    cout << "Process Control Demonstration:\n";
    pid_t pid = fork();
    if (pid == 0) {
        cout << "Child process running PID: " << getpid() << endl;
        sleep(2);
        cout << "Child process finished.\n";
        exit(0);
    } else if (pid>0) {
        cout << "Parent process running. PID: " <<getpid() << endl;
        cout<< "waiting for the Child process to finish..." << endl;
        wait(nullptr);
    } else{
        cerr << "Fork Failed." <<endl;
        return 1;
    }
    cout << "Listing current processes using 'ps': \n";
    system("ps");
    cout << "demonstrating Kill: " <<getpid()<<"\n";
    kill (getpid(), SIGTERM);
}
```

```
PROBLEMS 1  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  SOLIDITY: TRANSACTIONS  ...  wsl

uv-goswami@uvgoswami:/mnt/d/Codes/os$ g++ -o process_control Q2.cpp
uv-goswami@uvgoswami:/mnt/d/Codes/os$ ./process_control
Process Control Demonstration:
Parent process running. PID: 15756
waiting for the Child process to finish...
Child process running PID: 15757
Child process finished.
Listing current processes using 'ps':
    PID TTY          TIME CMD
  15710 pts/2    00:00:00 bash
  15756 pts/2    00:00:00 process_control
  15758 pts/2    00:00:00 sh
  15759 pts/2    00:00:00 ps
demonstrating Kill: 15756
Terminated
uv-goswami@uvgoswami:/mnt/d/Codes/os$
```

ii. Communication: Input-output redirection, Pipe

```
uv-goswami@uvgoswami:/mnt/d/Codes/os$ ls
Q2.cpp  output.txt  process_control  temp.txt
uv-goswami@uvgoswami:/mnt/d/Codes/os$ echo "This is a test." > output.txt
uv-goswami@uvgoswami:/mnt/d/Codes/os$ cat output.txt
This is a test.
uv-goswami@uvgoswami:/mnt/d/Codes/os$ wc -l < temp.txt
0
uv-goswami@uvgoswami:/mnt/d/Codes/os$ ls -l | grep ".txt"
-rwxrwxrwx 1 uv-goswami uv-goswami    16 Nov 25 05:55 output.txt
-rwxrwxrwx 1 uv-goswami uv-goswami     0 Nov 25 05:49 temp.txt
uv-goswami@uvgoswami:/mnt/d/Codes/os$
```

iii. Protection Management: chmod, chown, chgrp

```
root@uvgoswami:/mnt/d/Codes/os# ls
Q2.cpp  output.txt  process_control  temp.txt
root@uvgoswami:/mnt/d/Codes/os# chmod 755 temp.txt
root@uvgoswami:/mnt/d/Codes/os# ls -l temp.txt
-rwxrwxrwx 1 uv-goswami uv-goswami 0 Nov 25 05:49 temp.txt
root@uvgoswami:/mnt/d/Codes/os# chown root:root temp.txt
root@uvgoswami:/mnt/d/Codes/os# ls -l temp.txt
-rwxrwxrwx 1 uv-goswami uv-goswami 0 Nov 25 05:49 temp.txt
root@uvgoswami:/mnt/d/Codes/os# chgrp staff temp.txt
root@uvgoswami:/mnt/d/Codes/os# temp.txt
temp.txt: command not found
root@uvgoswami:/mnt/d/Codes/os# ls -l temp.txt
-rwxrwxrwx 1 uv-goswami uv-goswami 0 Nov 25 05:49 temp.txt
```

P3. Write a program (using fork () and/or exec () commands) where parent and child execute:

i. same program, same code.

ii. same program, different code.

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

Code:

```cpp
#include <iostream>
#include <unistd.h>
#include <sys/wait.h>
using namespace std;

void execute_same_program_same_code() {
    cout << "Executing the same program with same code.\n";}
void execute_same_program_different_code() {
    cout << "Parent is executing its task.\n";
    execlp("pwd", "pwd",nullptr);
    perror("exec failed");}

int main() {
    pid_t pid = fork();
    if (pid == -1) {
        cerr << "Fork failed!\n";
        return 1;}
    if (pid == 0) {
        cout << "Child process (PID: " << getpid() << ") started.\n";
        execute_same_program_same_code();
        execute_same_program_different_code();
    } else {
        cout << "Parent process (PID: " << getpid() << ") started.\n";
        execute_same_program_same_code();
        cout << "Parent is waiting for child to finish...\n";
        wait(NULL);
        cout << "Child has finished. Parent is now terminating.\n";}
    return 0;}
```

```
PROBLEMS 1  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  SOLIDITY: TRANSACTIONS  ...

Parent process (PID: 16451) started.
Executing the same program with same code.
Parent is waiting for child to finish...
Child process (PID: 16452) started.
Executing the same program with same code.
Parent is executing its task.
/mnt/d/Codes/os
Child has finished. Parent is now terminating.
root@uvgoswami:/mnt/d/Codes/os#
```

P4. Write a program to report behaviour of Linux kernel including kernel version, CPU type and CPU information.

Code:

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

int main() {
printf("Kernel Version:\n");
system("uname -r");
printf("\nCPU Type:\n");
system("uname -m");
printf("\nDetailed CPU Information:\n");
system("lscpu");
return 0;
}
```

Output:

```
root@uvgoswami:/mnt/d/Codes/os# ./Q4
Kernel Version:
5.15.167.4-microsoft-standard-WSL2

CPU Type:
x86_64

Detailed 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):                  16
  On-line CPU(s) list:   0-15
Vendor ID:               AuthenticAMD
  Model name:            AMD Ryzen 7 5800HS with Radeon Graphics
    CPU family:          25
    Model:               80
    Thread(s) per core:  2
    Core(s) per socket:  8
    Socket(s):           1
    Stepping:            0
    BogoMIPS:            6387.83
```

P5. Write a program to report behaviour of Linux kernel including information on configured memory, amount of free and used memory.

Code:

```cpp
#include <iostream>
#include <fstream>
#include <string>
using namespace std;
void get_memory_info() {
    ifstream meminfo("/proc/meminfo");
    string line;
    if (!meminfo.is_open()) {
        cerr << "Failed to open /proc/meminfo" << endl;
        exit(1);}
    cout << "\nMemory Information:\n";
    while (getline(meminfo, line)) {
        if (line.find("MemTotal") != string::npos) {
            cout << line << endl;}
        if (line.find("MemFree") != string::npos) {
            cout << line << endl; }
        if (line.find("MemAvailable") != string::npos) {
            cout << line << endl;}
        if (line.find("Buffers") != string::npos) {
            cout << line << endl; }
        if (line.find("Cached") != string::npos) {
            cout << line << endl;}
        if (line.find("SwapTotal") != string::npos) {
            cout << line << endl; }
        if (line.find("SwapFree") != string::npos) {
            cout << line << endl;}}
    meminfo.close();}
int main() {
    cout << "Linux Kernel Memory Information:\n";
    get_memory_info();
    return 0;}
```

Output:

```
root@uvgoswami:/mnt/d/Codes/os# ./Q5
Linux Kernel Memory Information:

Memory Information:
MemTotal:        7825748 kB
MemFree:         6493980 kB
MemAvailable:    6976756 kB
Buffers:           10444 kB
Cached:           633844 kB
SwapCached:            0 kB
SwapTotal:       2097152 kB
SwapFree:        2097152 kB
root@uvgoswami:/mnt/d/Codes/os#
```

P6.Write a program to copy files using system calls.

Code:

```cpp
#include <iostream>
#include <fcntl.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
using namespace std;
void copy_file(const char* source, const char* destination) {
    int source_fd = open(source, O_RDONLY);
    if (source_fd == -1) {
        perror("Failed to open source file");
        exit(1);}
    int dest_fd = open(destination, O_WRONLY | O_CREAT | O_TRUNC, S_IRUSR | S_IWUSR);
    if (dest_fd == -1) {
        perror("Failed to open destination file");
        close(source_fd);
        exit(1);}
    char buffer[1024];
    ssize_t bytes_read, bytes_written;
    while ((bytes_read = read(source_fd, buffer, sizeof(buffer))) > 0) {
        bytes_written = write(dest_fd, buffer, bytes_read);
        if (bytes_written != bytes_read) {
            perror("Error writing to destination file");
            close(source_fd);
            close(dest_fd);
            exit(1);}}
    if (bytes_read == -1) {
        perror("Error reading from source file");}
    close(source_fd);
    close(dest_fd);
    cout << "File copied successfully from " << source << " to " << destination << endl;}
int main() {
    const char* source = "source.txt";
    const char* destination = "destination.txt";
    copy_file(source, destination);
    return 0;}
```

Output:

```
root@uvgoswami:/mnt/d/Codes/os# ./Q6
File copied successfully from source.txt to destination.txt
root@uvgoswami:/mnt/d/Codes/os#
```

P7. Write a program to implement FCFS scheduling algorithm.

Code:

```cpp
#include <iostream>
using namespace std;
void First_Come_First_Serve(string name_of_process[], int burst_time[], int number_of_processes) {
    double waiting_time = 0;
    double turnaround_time = 0;
    double total_waiting_time = 0;
    double total_turnaround_time = 0;
    cout << "Name Of Process\tBurst Time\tWaiting Time\tTurnAround Time" << endl;
    for (int i = 0; i < number_of_processes; i++) {
        turnaround_time += burst_time[i];
        total_turnaround_time += turnaround_time;
        cout << name_of_process[i] << "\t\t" << burst_time[i] << "\t\t" << waiting_time << "\t\t";
        cout<< turnaround_time << endl;
        total_waiting_time += waiting_time;
        waiting_time += burst_time[i];
    }
    cout<<"\nAverage Waiting Time is:: "<<total_waiting_time/number_of_processes<< "\nAverage TurnAround Time is::" ;
    cout << total_turnaround_time / number_of_processes << endl;
}
int main() {
    int number_of_processes;
    cout << "Enter the number of processes: ";
    cin >> number_of_processes;
    string name_of_process[number_of_processes];
    int burst_time[number_of_processes];

    for (int i = 0; i < number_of_processes; i++) {
        cout << "Enter Name of Process:: ";
        cin >> name_of_process[i];
        cout << "Enter Burst Time of Process " << name_of_process[i] << ":: ";
        cin >> burst_time[i];
    }

    First_Come_First_Serve(name_of_process, burst_time, number_of_processes);
    return 0;
}
```

Output:

```
root@uvgoswami:/mnt/d/Codes/os# ./Q7
Enter the number of processes: 4
Enter Name of Process:: p1
Enter Burst Time of Process p1:: 6
Enter Name of Process:: p2
Enter Burst Time of Process p2:: 9
Enter Name of Process:: p3
Enter Burst Time of Process p3:: 7
Enter Name of Process:: p4
Enter Burst Time of Process p4:: 1
Name Of Process Burst Time      Waiting Time    TurnAround Time
p1              6               0               6
p2              9               6               15
p3              7               15              22
p4              1               22              23

Average Waiting Time is:: 10.75
Average TurnAround Time is::16.5
root@uvgoswami:/mnt/d/Codes/os#
```

P8.Write a program to implement SJF scheduling algorithm.

Code:

```
#include <iostream>
using namespace std;
void First_Come_First_Serve(string name_of_process[], int burst_time[], int number_of_processes) {
    double waiting_time = 0;
    double turnaround_time = 0;
    double total_waiting_time = 0;
    double total_turnaround_time = 0;
    cout << "Name Of Process\tBurst Time\tWaiting Time\tTurnAround Time" << endl;
    for (int i = 0; i < number_of_processes; i++) {
        turnaround_time += burst_time[i];
        total_turnaround_time += turnaround_time;
        cout << name_of_process[i] << "\t\t" << burst_time[i] << "\t\t" << waiting_time << "\t\t";
        cout<< turnaround_time << endl;
        total_waiting_time += waiting_time;
        waiting_time += burst_time[i];
    }
    cout << "\nAverage Waiting Time is: " << total_waiting_time / number_of_processes;
    cout << "\nAverage TurnAround Time is: " << total_turnaround_time / number_of_processes << endl;}
void Sort(string name_of_process[], int burst_time[], int number_of_processes) {
    for (int i = 0; i < number_of_processes; i++) {
        for (int j = 0; j < number_of_processes; j++) {
            if (burst_time[i] < burst_time[j]) {      You, 11 hours ago • first commit
                int temp_burst = burst_time[i];
                burst_time[i] = burst_time[j];
                burst_time[j] = temp_burst;
                string temp_name_of_process = name_of_process[i];
                name_of_process[i] = name_of_process[j];
                name_of_process[j] = temp_name_of_process;
            }
        }
    }
    First_Come_First_Serve(name_of_process, burst_time, number_of_processes);}
int main() {
    int number_of_processes;
    cout << "Enter the number of processes: ";
    cin >> number_of_processes;
    string name_of_process[number_of_processes];
    int burst_time[number_of_processes];
    for (int i = 0; i < number_of_processes; i++) {
        cout << "Enter Name of Process: ";
        cin >> name_of_process[i];
        cout << "Enter Burst Time of Process " << name_of_process[i] << ": ";
        cin >> burst_time[i];
    }
    Sort(name_of_process, burst_time, number_of_processes);
    return 0;
}
```

Output:

```
root@uvgoswami:/mnt/d/Codes/os# g++ -o Q8 Q8.cpp
root@uvgoswami:/mnt/d/Codes/os# ./Q8
Enter the number of processes: 4
Enter Name of Process: p1
Enter Burst Time of Process p1: 7
Enter Name of Process: p2
Enter Burst Time of Process p2: 9
Enter Name of Process: p3
Enter Burst Time of Process p3: 3
Enter Name of Process: p4
Enter Burst Time of Process p4: 1
Name Of Process Burst Time      Waiting Time    TurnAround Time
p4              1               0               1
p3              3               1               4
p1              7               4               11
p2              9               11              20

Average Waiting Time is: 4
Average TurnAround Time is: 9
root@uvgoswami:/mnt/d/Codes/os#
```

P9. Write a program to implement non-preemptive priority-based scheduling algorithm.

Code:

```
#include <iostream>
using namespace std;
void First_Come_First_Serve(string name_of_process[], int burst_time[], int priority[], int number_of_processes)
    double waiting_time = 0;
    double turnaround_time = 0;
    double total_waiting_time = 0;
    double total_turnaround_time = 0;
    cout << "Name Of Process\tBurst Time\tPriority\tWaiting Time\tTurnAround Time" << endl;
    for (int i = 0; i < number_of_processes; i++) {
        turnaround_time += burst_time[i];
        total_turnaround_time += turnaround_time;
        cout << name_of_process[i] << "\t\t" << burst_time[i] << "\t\t" << priority[i];
        cout << "\t\t" << waiting_time << "\t\t" << turnaround_time << endl;
        total_waiting_time += waiting_time;
        waiting_time += burst_time[i];}
    cout << "\nAverage Waiting Time is: " << total_waiting_time / number_of_processes ;
    cout << "\nAverage TurnAround Time is: " << total_turnaround_time / number_of_processes << endl;}
void Sort(string name_of_process[], int burst_time[], int priority[], int number_of_processes) {
    for (int i = 0; i < number_of_processes; i++) {
        for (int j = 0; j < number_of_processes; j++) {
            if (priority[i] < priority[j]) {
                int temp_burst = burst_time[i];
                burst_time[i] = burst_time[j];      You, 11 hours ago • first commit
                burst_time[j] = temp_burst;
                int temp_priority = priority[i];
                priority[i] = priority[j];
                priority[j] = temp_priority;
                string temp_name_of_process = name_of_process[i];
                name_of_process[i] = name_of_process[j];
                name_of_process[j] = temp_name_of_process;}}}
    First_Come_First_Serve(name_of_process, burst_time, priority, number_of_processes);}
int main() {
    int number_of_processes;
    cout << "Enter the number of processes: ";
    cin >> number_of_processes;
    string name_of_process[number_of_processes];
    int burst_time[number_of_processes];
    int priority[number_of_processes];
    for (int i = 0; i < number_of_processes; i++) {
        cout << "Enter Name of Process: ";
        cin >> name_of_process[i];
        cout << "Enter Burst Time of Process " << name_of_process[i] << ": ";
        cin >> burst_time[i];
        cout << "Enter Priority of Process " << name_of_process[i] << ": ";
        cin >> priority[i];}
    Sort(name_of_process, burst_time, priority, number_of_processes);
    return 0;}
```

Output:

```
root@uvgoswami:/mnt/d/Codes/os# ./Q9
Enter the number of processes: 4
Enter Name of Process: p1
Enter Burst Time of Process p1: 7
Enter Priority of Process p1: 2
Enter Name of Process: p2
Enter Burst Time of Process p2: 9
Enter Priority of Process p2: 1
Enter Name of Process: p3
Enter Burst Time of Process p3: 1
Enter Priority of Process p3: 3
Enter Name of Process: p4
Enter Burst Time of Process p4: 3
Enter Priority of Process p4: 4
Name Of Process Burst Time      Priority        Waiting Time    TurnAround Time
p2              9               1               0               9
p1              7               2               9               16
p3              1               3               16              17
p4              3               4               17              20

Average Waiting Time is: 10.5
Average TurnAround Time is: 15.5
root@uvgoswami:/mnt/d/Codes/os#
```

P10. Write a program to implement SRTF scheduling algorithm.

Code:

```
#include <iostream>
#include <algorithm>
#include <limits>

using namespace std;

void display(string name_of_process[],int burst_time[],int arrival_time[],int number_of_processes,int waiting_time[],int turnaround_time[]){
    double total_waiting_time = 0;
    double total_turnaround_time = 0;

    cout << "Name Of Process\tBurst Time\tArrival Time\tWaiting Time\tTurnAround Time" << endl;
    for (int num = 0; num < number_of_processes; num++) {
        total_waiting_time += waiting_time[num];
        total_turnaround_time += turnaround_time[num];
        cout << name_of_process[num] << "\t\t" << burst_time[num] << "\t\t" << arrival_time[num];
        cout<< "\t\t" << waiting_time[num];
        cout << "\t\t" << turnaround_time[num] << endl;
    }
    cout << "\nAverage Waiting Time is: " << total_waiting_time/number_of_processes;
    cout << "\nAverage TurnAround Time is: ";
    cout<< total_turnaround_time / number_of_processes << endl;
}
```

```
void shortest_remaining_time_first(string name_of_process[], int number_of_processes, int burst_time[], int arrival_time[]) {
    int waiting_time[number_of_processes];
    int turnaround_time[number_of_processes];
    int remaining_time[number_of_processes];
    copy(burst_time, burst_time + number_of_processes, remaining_time);

    int clock = 0;
    int completed = 0;
    float min_burst = numeric_limits<float>::infinity();
    int shortest = -1;
    bool finished = false;

    while (completed != number_of_processes) {
        for (int num = 0; num < number_of_processes; num++) {
            if (arrival_time[num] <= clock && remaining_time[num] < min_burst && remaining_time[num] > 0) {
                min_burst = remaining_time[num];
                shortest = num;
                finished = true;
            }
        }
```

```
while (completed != number_of_processes) {
    for (int num = 0; num < number_of_processes; num++) {
        if (arrival_time[num] <= clock && remaining_time[num] < min_burst && remaining_time[num] > 0) {
            min_burst = remaining_time[num];
            shortest = num;
            finished = true;
        }
    }

    if (!finished) {
        clock++;
        continue;
    }

    remaining_time[shortest]--;
    min_burst = remaining_time[shortest];
    if (min_burst == 0) {
        min_burst = numeric_limits<float>::infinity();
    }
```

```
        if (remaining_time[shortest] == 0) {
            completed++;
            finished = false;
            int finish_time = clock + 1;
            waiting_time[shortest] = finish_time - burst_time[shortest] - arrival_time[shortest];
            turnaround_time[shortest] = finish_time - arrival_time[shortest];

            if (waiting_time[shortest] < 0) {
                waiting_time[shortest] = 0;
            }
        }

        clock++;
    }

    display(name_of_process, burst_time, arrival_time, number_of_processes, waiting_time, turnaround_time);
}
```

```
int main() {
    int number_of_processes;
    cout << "Enter the number of processes: ";
    cin >> number_of_processes;
    string name_of_process[number_of_processes];
    int burst_time[number_of_processes];
    int arrival_time[number_of_processes];
    for (int i = 0; i < number_of_processes; i++) {
        cout << "Enter Name of Process: ";
        cin >> name_of_process[i];
        cout << "Enter Burst Time of Process " << name_of_process[i] << ": ";
        cin >> burst_time[i];
        cout << "Enter Arrival Time of Process " << name_of_process[i] << ": ";
        cin >> arrival_time[i];
    }

    shortest_remaining_time_first(name_of_process, number_of_processes, burst_time, arrival_time);
    return 0;
}
```

Output:

```
root@uvgoswami:/mnt/d/Codes/os# ./Q10
Enter the number of processes: 4
Enter Name of Process: p1
Enter Burst Time of Process p1: 7
Enter Arrival Time of Process p1: 2
Enter Name of Process: p2
Enter Burst Time of Process p2: 9
Enter Arrival Time of Process p2: 5
Enter Name of Process: p3
Enter Burst Time of Process p3: 3
Enter Arrival Time of Process p3: 1
Enter Name of Process: p4
Enter Burst Time of Process p4: 1
Enter Arrival Time of Process p4: 4
Name Of Process Burst Time      Arrival Time    Waiting Time    TurnAround Time
p1              7               2               3               10
p2              9               5               7               16
p3              3               1               0               3
p4              1               4               0               1

Average Waiting Time is: 2.5
Average TurnAround Time is: 7.5
root@uvgoswami:/mnt/d/Codes/os#
```

P11.Write a program to calculate sum of n numbers using Pthreads. A list of n numbers is divided into two smaller list of equal size, two separate threads are used to sum the sub lists.

Code:

```
#include <pthread.h>
#include <iostream>
#include <vector>
using namespace std;
struct ThreadData {
    vector<int>& arr;
    int start;
    int end;
    int sum;
};
void* calculate_sum(void* arg) {
    ThreadData* data = static_cast<ThreadData*>(arg);
    data->sum = 0;
    for (int i = data->start; i < data->end; ++i) {
        data->sum += data->arr[i];
    }
    pthread_exit(nullptr);
}
int main() {
    int n;
    cout << "Enter the number of elements: ";
    cin >> n;
    vector<int> arr(n);
    cout << "Enter " << n << " elements: " << endl;
    for (int i = 0; i < n; ++i) {
        cin >> arr[i];}
    int mid = n / 2;
    ThreadData data1 = {arr, 0, mid, 0};
    ThreadData data2 = {arr, mid, n, 0};
    pthread_t thread1, thread2;
    pthread_create(&thread1, nullptr, calculate_sum, static_cast<void*>(&data1));
    pthread_create(&thread2, nullptr, calculate_sum, static_cast<void*>(&data2));
    pthread_join(thread1, nullptr);
    pthread_join(thread2, nullptr);
    int total_sum = data1.sum + data2.sum;
    cout << "The sum of the numbers is: " << total_sum << endl;
    return 0;}
```

Output:

```
root@uvgoswami:/mnt/d/Codes/os# ./Q11
Enter the number of elements: 4
Enter 4 elements:
2 4 6 8
The sum of the numbers is: 20
root@uvgoswami:/mnt/d/Codes/os#
```

P12. Write a program to implement first-fit, best-fit and worst-fit allocation strategies.

Code:

```
#include <iostream>
#include <vector>
#include <string>
#include <sstream>
using namespace std;
void display(const vector<pair<string, int>>& memory) {
    int count = 1;
    cout << "Memory Status:" << endl;
    for (const auto& frame : memory) {
        cout << "Frame: " << count << " | Process: " << frame.first << " | Size: " << frame.second << endl;
        count++;
    }
}
void firstFit(int noOfFrames, vector<pair<string, int>>& memory, const pair<string, int>& process) {
    bool flag = false;
    for (int i = 0; i < noOfFrames; i++) {
        if (memory[i].first == "free" && memory[i].second >= process.second) {
            memory[i].first = process.first;
            display(memory);
            flag = true;
            break;
        }
    }
    if (!flag) {
        cout << "\nYou do not have enough space to run the new process" << endl;
    }
}
void bestFit(int noOfFrames, vector<pair<string, int>>& memory, const pair<string, int>& process) {
    int flag = -1;
    for (int i = 0; i < noOfFrames; i++) {
        if (memory[i].first == "free" && memory[i].second >= process.second) {
            if (flag == -1 || memory[i].second < memory[flag].second) {
                flag = i;
            }
        }
    }
    if (flag != -1) {
        memory[flag].first = process.first;
        display(memory);
    } else {
        cout << "\nYou do not have enough space to run the new process." << endl;
    }
}
```

```cpp
void worstFit(int noOfFrames, vector<pair<string, int>>& memory, const pair<string, int>& process) {
    int flag = -1;
    for (int i = 0; i < noOfFrames; i++) {
        if (memory[i].first == "free" && memory[i].second >= process.second) {
            if (flag == -1 || memory[i].second > memory[flag].second) {
                flag = i;
            }}}
    if (flag != -1) {
        memory[flag].first = process.first;
        display(memory);
    } else {
        cout << "\nYou do not have enough space to run the new process." << endl;}}
int main() {
    vector<pair<string, int>> memory;
    while (true) {
        cout << "Enter memory type (used/free) and value, or 'q' to quit: ";
        string input;
        getline(cin, input);
        if (input == "q") {
            break;
        }
        istringstream iss(input);
        string memoryType;
        int value;
        iss >> memoryType >> value;
        memory.push_back({memoryType, value});
    }
    cout << "Enter process name and value: ";
    string processName;
    int processValue;
    cin >> processName >> processValue;
    pair<string, int> process = {processName, processValue};
    cout << "Main Menu\n1. First Fit\n2. Best Fit\n3. Worst Fit\nEnter your choice: ";
    int choice;
    cin >> choice;
    if (choice == 1) {
        firstFit(memory.size(), memory, process);
    } else if (choice == 2) {
        bestFit(memory.size(), memory, process);
    } else {
        worstFit(memory.size(), memory, process);}
    return 0;
}
```

Output:

```
root@uvgoswami:/mnt/d/Codes/os# g++ -o Q12 Q12.cpp
root@uvgoswami:/mnt/d/Codes/os# ./Q12
Enter memory type (used/free) and value, or 'q' to quit: used 45
Enter memory type (used/free) and value, or 'q' to quit: free 12
Enter memory type (used/free) and value, or 'q' to quit: used 31
Enter memory type (used/free) and value, or 'q' to quit: free 9
Enter memory type (used/free) and value, or 'q' to quit: q
Enter process name and value: p1 9
Memory Status:
Frame: 1 | Process: used | Size: 45
Frame: 2 | Process: p1 | Size: 12
Frame: 3 | Process: used | Size: 31
Frame: 4 | Process: free | Size: 9
Memory Status:
Frame: 1 | Process: used | Size: 45
Frame: 2 | Process: p1 | Size: 12
Frame: 3 | Process: used | Size: 31
Frame: 4 | Process: p1 | Size: 9

You do not have enough space to run the new process.
root@uvgoswami:/mnt/d/Codes/os#
```