CSBB203: Operating System Lab

Lab Practical File

Submitted By:

Name: KARTIK MITTAL

Roll No: 221210056

Branch: CSE

Semester: 3

Group: 2

Submitted To: Dr. Amandeep Kaur

Department of Computer Science and Engineering

NATIONAL INSTITUTE OF TECHNOLOGY DELHI

2023

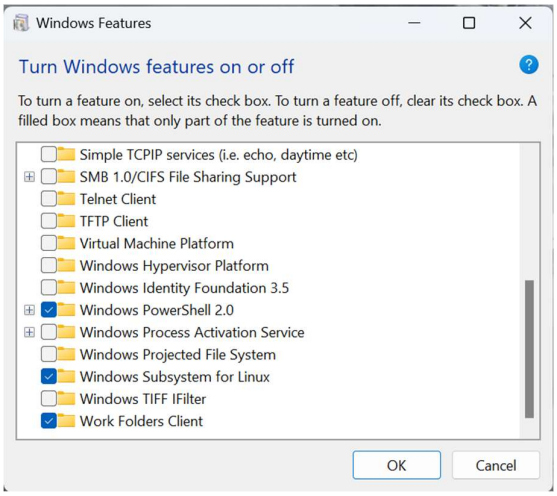
INDEX

Lab Assignment – 1

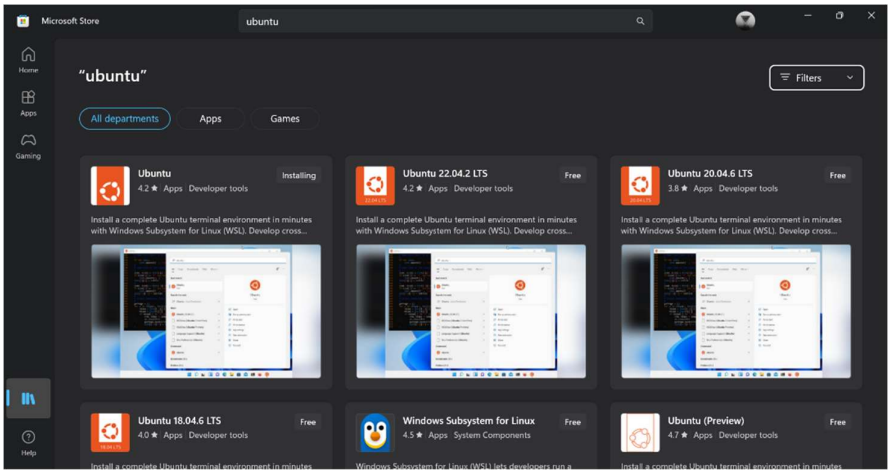
Aim: Installation of Ubuntu Linux

Theory: Following are the steps to setup Ubuntu Linux using WSL (Windows Subsystem for Linux)

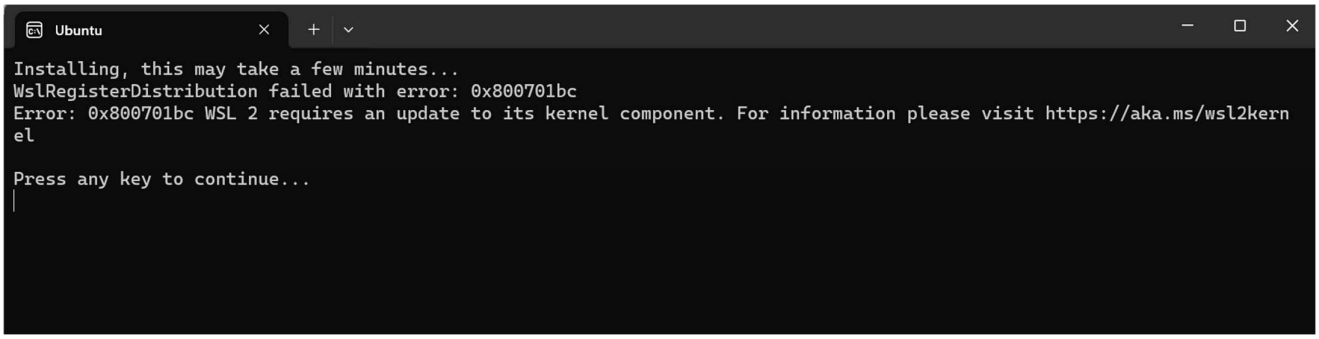
1. Search for "Turn Windows features on or off" in search bar. Then enable the Windows Subsystem for Linux and restart the system.



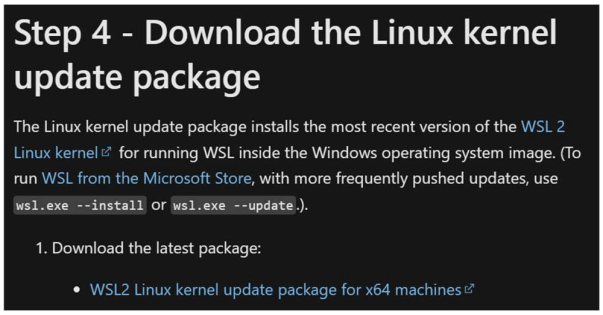
2. Install the Ubuntu Linux from the Windows Store.



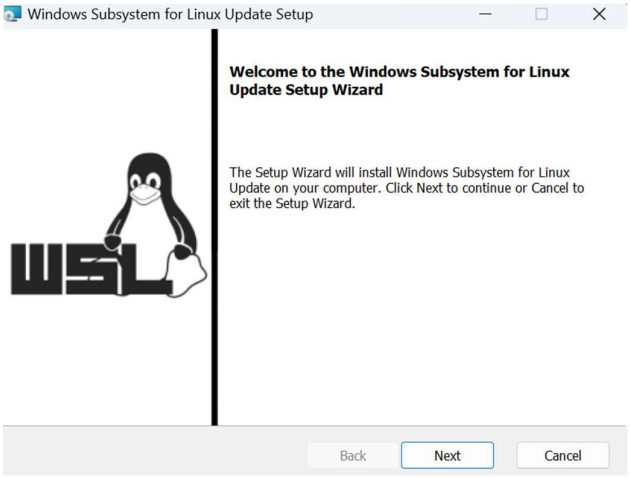
3. Open Ubuntu from windows search bar it would show the following error. To solve this visit the provided link.



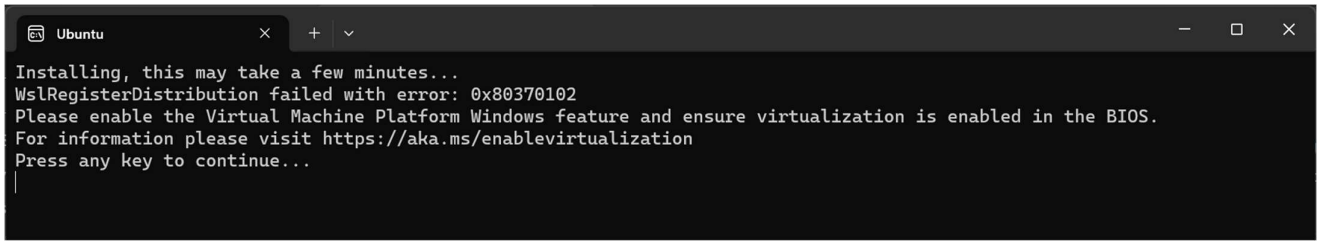
4. Link will redirect to download the WSL2 linux kernel update package.



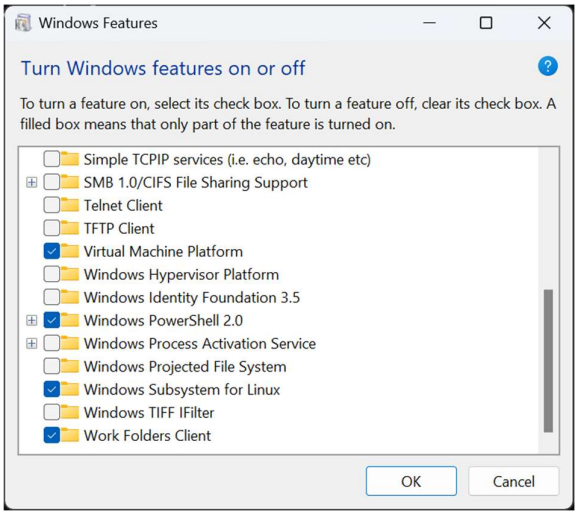
5. Install the downloaded WSL2 Linux Kernel Update Package.



6. Again open the Ubuntu from the windows search bar, it show error and ask to enable Virtualization of CPU and Virtual Machine Platform Windows Feature.



7. Search and Open “Turn Windows features on or off” and enable Virtual Machine Platform.



Lab Assignment – 2

Aim: Use some Linux Commands

Theory:

1 :Pwd command(Present Working Directory) :

Output :

```
kartikmittal@LAPTOP-NS0O0096:~$ pwd
/home/kartikmittal
kartikmittal@LAPTOP-NS0O0096:~$
```

2. cd ..

Output :

```
kartikmittal@LAPTOP-NS0O0096:~$ cd /dev
kartikmittal@LAPTOP-NS0O0096:/dev$ cd ..
kartikmittal@LAPTOP-NS0O0096:/$
```

3. ~cd or cd /

Output :

```
kartikmittal@LAPTOP-NS0O0096:/$ cd /
kartikmittal@LAPTOP-NS0O0096:/$
```

4. ls

Output :

```
kartikmittal@LAPTOP-NS0O0096:/$ ls
bin   dev  home  lib    lib64   lost+found  mnt  proc  run   snap  sys  usr
boot  etc  init  lib32  libx32  media       opt  root  sbin  srv   tmp  var
kartikmittal@LAPTOP-NS0O0096:/$
```

Basic Linux/Unix Commands :

1. **Ls-R** : Lists files in sub-directories as well **.**

```
kartikmittal@LAPTOP-NS0O0096:~$ ls -R
.:
hell  hello  hello.c
kartikmittal@LAPTOP-NS0O0096:~$
```

2. **Ls-a** : Lists hidden files as well.

```
kartikmittal@LAPTOP-NS0O0096:~$ ls -a
.   .bash_history  .bashrc  .motd_shown  .sudo_as_admin_successful  hell   hello.c
..  .bash_logout   .cache   .profile     .viminfo                   hello
kartikmittal@LAPTOP-NS0O0096:~$
```

3. **Cat >filename :** creates a new file.

```
kartikmittal@LAPTOP-NS0O0096:~$ cat >kartik
kartik
```

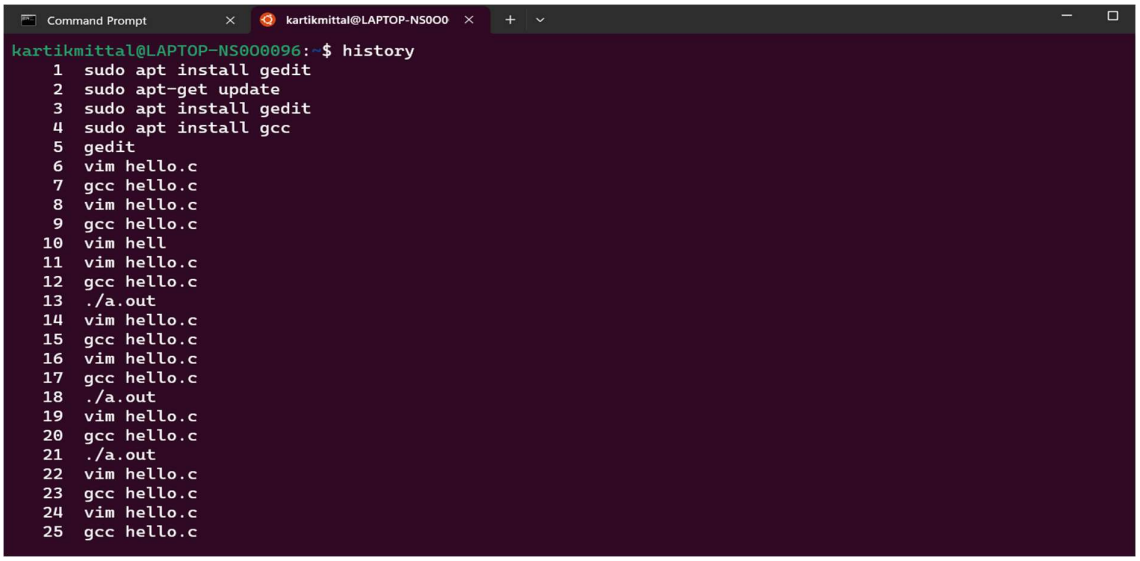
4. **Cat filename :** displays content of a file.

```
kartikmittal@LAPTOP-NS0O0096:~$ cat hello.c
#include<stdio.h>

int main(){


char c;
   scanf(" %c",&c);

    int a=c;
    long decimal=0;
    int temp;
    int pow=1;

    while(a>0){
        temp=a%8;
         decimal=decimal+temp*pow;

        pow=pow*10;

        a=a/8;
    }

   printf(" the octal number is :%ld ",decimal);
    return 0;
```

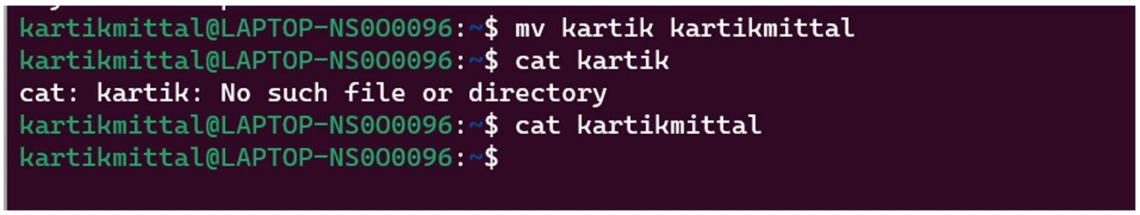
5. **Man**=gives help information on command.

```
CAT(1)                                  User Commands                                  CAT(1)

NAME
       cat - concatenate files and print on the standard output

SYNOPSIS
       cat [OPTION]... [FILE]...

DESCRIPTION
       Concatenate FILE(s) to standard output.

       With no FILE, or when FILE is -, read standard input.

       -A, --show-all
              equivalent to -vET

       -b, --number-nonblank
              number nonempty output lines, overrides -n

       -e     equivalent to -vE

       -E, --show-ends
              display $ at end of each line

       -n, --number
 Manual page cat(1) line 1 (press h for help or q to quit)
```

6. **History** : Gives a list of all past basic Linux commands list typed in the current terminal session.



7. **Mv** : renames a directory.



Lab Assignment – 3

Aim: Write C code for demonstrating following Process Operations:

a) Creation [Fork]
b) Program Execution [Execlp/ Execvp]
c) Child Process Handling [Wait]
d) Termination [exit / _exit with return from child]

Code & Output:

a) Creation [Fork]

Code:

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

void main() {
    int pid= fork();
    if(pid == 0){
        printf("pid child: %d", getpid());
        printf("\nHello everyone I am in child process.....!!!!\n");
    }

    else{
        printf("pid parent: %d", getpid());
        printf("\nHello everyone I am in parent process.....!!!!\n");
    }
}
```

Output:

```
kartikmittal@KARTIKWORLD:~$ gcc os_lab1.c
kartikmittal@KARTIKWORLD:~$ ./a.out
the parent id is : 81
hello, i am in parent
the child id is :82
hello everyone , i am in child!!!
kartikmittal@KARTIKWORLD:~$
```

b) Program Execution [Execlp/ Execvp]
c) Child Process Handling [Wait]
d) Termination [exit / _exit with return from child]

Code:

func.c

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

void main(int args, char *argv[]) {
    int i;

    if(args< 3){
        printf("Not enough arguments...!!!\n");
        exit(1);
    }

    for(i= atoi( argv[1]); i<= atoi(argv[2]); i++){
        printf("%d ", i);
    }
    printf("\n");
}
```

Os_lab2.c

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

void main(){
    int n, i, p, start, end;
    printf("Enter the ending number: ");
    scanf("%d", &n);
    printf("Enter the number of processes: ");
    scanf("%d", &p);

    start=1;
    end= n/p;
    char a[10], b[10];

    for(i=0; i<p; i++){
        sprintf(a, "%d", start);
        sprintf(b, "%d", end);
        int pid= fork();

        if(pid == 0){
            char *args[]= {"./func", a, b, NULL};
            execvp(args[0], args);
            printf("Execlp failed...!!!");
        }

        else{
            int status;
            wait(&status);
            start= end+ 1;
            end= end+ n/p;
        }
    }
}
```

Output:

```
Enter the ending number: 100
Enter the number of processes: 10
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
31 32 33 34 35 36 37 38 39 40
41 42 43 44 45 46 47 48 49 50
51 52 53 54 55 56 57 58 59 60
61 62 63 64 65 66 67 68 69 70
71 72 73 74 75 76 77 78 79 80
81 82 83 84 85 86 87 88 89 90
91 92 93 94 95 96 97 98 99 100
```

Lab Assignment – 4

Aim: Simulation of CPU scheduling algorithms:

a) Non – Pre Emptive:
- FCFS (First Come First Serve)
- SJF (Shortest Job First)
- HRN (Highest Response Ratio Next)

Code & Output:

FCFS (First Come First Serve) Code:

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

struct process{
    int id;
    int arrTime;
    int burstTime;
    int waitTime;
    int turnAroundTime;
    int compTime;
    int isCompleted;
};


int main() {
    int n;
    printf("Enter the number of processes: ");
    scanf("%d", &n);

    struct process processes[n];

    for(int i =0; i<n; i++){
        printf("\nEnter arrival and burst time for P%d: ", i);
        scanf("%d %d", &processes[i].arrTime, &processes[i].burstTime);
        processes[i].isCompleted= 0;
        processes[i].id =i;
    }

    int time= 0;
    int completed = 0;

    while (completed != n){
        int selectedIndex= -1;
        int selectedArrTime = INT_MAX;
        for(int i =0; i<n; i++){
            if(processes[i].arrTime <= time && !processes[i].isCompleted
            && processes[i].arrTime < selectedArrTime){
                selectedArrTime= processes[i].arrTime;
                selectedIndex =i;
            }
        }
        if(selectedIndex != -1){
            int i = selectedIndex;
            time += processes[i].burstTime;
            processes[i].compTime= time;
            processes[i].turnAroundTime = processes[i].compTime- processes[i].arrTime;
```

```
            processes[i].waitTime= processes[i].turnAroundTime- processes[i].burstTime;
            processes[i].isCompleted= 1;
            completed++;
        }

        if(selectedIndex == -1)
            time++;
    }

    int totalWaitTime= 0;
    int totalTurnAroundTime= 0;

    for(int i=0; i<n; i++){
        totalWaitTime+= processes[i].waitTime;
        totalTurnAroundTime += processes[i].turnAroundTime;
    }

    float avgTWT= (float) totalWaitTime/ n;
    float avgTAT= (float) totalTurnAroundTime/ n;

    printf("\n\nProcess\tArrTime\tBurstTime\tCompTime\tTurnArTime\tWaitTime");
    for(int i=0; i<n; i++){
        printf("\nP%d\t%d\t%d\t\t%d\t\t%d\t\t%d", processes[i].id, processes[i].arrTime,
        processes[i].burstTime, processes[i].compTime, processes[i].turnAroundTime,
        processes[i].waitTime);
    }

    printf("\nTotal Wait Time: %d", totalWaitTime);
    printf("\nTotal Turnaround Time: %d", totalTurnAroundTime);
    printf("\nAverage Wait Time: %.2f", avgTWT);
    printf("\nAverage Turn Around Time: %.2f", avgTAT);

    return 0;
}
```

FCFS (First Come First Serve) Output:

```
Enter the number of processes: 5
Enter arrival and burst time for P0: 0 10
Enter arrival and burst time for P1: 3 5
Enter arrival and burst time for P2: 5 2
Enter arrival and burst time for P3: 6 6
Enter arrival and burst time for P4: 8 4


Process ArrTime BurstTime       CompTime        TurnArTime      WaitTime
P0      0       10              10              10              0
P1      3       5               15              12              7
P2      5       2               17              12              10
P3      6       6               23              17              11
P4      8       4               27              19              15
Total Wait Time: 43
Total Turnaround Time: 70
Average Wait Time: 8.60
Average Turn Around Time: 14.00
```

SJF (Shortest Job First) Code:

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

struct process{
    int id;
    int arrTime;
    int burstTime;
    int waitTime;
    int turnAroundTime;
    int compTime;
    int isCompleted;
};

int main() {
    int n;
    printf("Enter the number of processes: ");
    scanf("%d", &n);

    struct process processes[n];

    for(int i =0; i<n; i++){
        printf("\nEnter arrival and burst time for P%d: ", i);
        scanf("%d %d", &processes[i].arrTime, &processes[i].burstTime);
        processes[i].isCompleted= 0;
        processes[i].id =i;
    }

    int time= 0;
    int completed = 0;

    while (completed != n){
        int selectedIndex= -1;
        int selectedShortestTime= INT_MAX;
        for(int i=0; i<n; i++){
            if(processes[i].arrTime <= time && !processes[i].isCompleted &&
            processes[i].burstTime< selectedShortestTime){
                selectedShortestTime = processes[i].burstTime;
                selectedIndex = i;
            }
        }

        if(selectedIndex != -1){
            int i= selectedIndex;
            time += processes[i].burstTime;
            processes[i].compTime= time;
            processes[i].turnAroundTime = processes[i].compTime- processes[i].arrTime;
            processes[i].waitTime= processes[i].turnAroundTime- processes[i].burstTime;
            processes[i].isCompleted= 1;
            completed++;
        }

        if(selectedIndex == -1)
            time++;
    }

    int totalWaitTime= 0;
    int totalTurnAroundTime= 0;

    for(int i=0; i<n; i++){
```

```
        totalWaitTime+= processes[i].waitTime;
        totalTurnAroundTime += processes[i].turnAroundTime;
    }

    float avgTWT= (float) totalWaitTime/ n;
    float avgTAT= (float) totalTurnAroundTime/ n;

    printf("\n\nProcess\tArrTime\tBurstTime\tCompTime\tTurnArTime\tWaitTime");
    for(int i=0; i<n; i++){
        printf("\nP%d\t%d\t%d\t\t%d\t\t%d\t\t%d", processes[i].id, processes[i].arrTime,
        processes[i].burstTime, processes[i].compTime, processes[i].turnAroundTime,
        processes[i].waitTime);
    }

    printf("\nTotal Wait Time: %d", totalWaitTime);
    printf("\nTotal Turnaround Time: %d", totalTurnAroundTime);
    printf("\nAverage Wait Time: %.2f", avgTWT);
    printf("\nAverage Turn Around Time: %.2f", avgTAT);

    return 0;
}
```

SJF (Shortest Job First) Output:

```
Enter the number of processes: 4
Enter arrival and burst time for P0: 2 3
Enter arrival and burst time for P1: 5 6
Enter arrival and burst time for P2: 4 2
Enter arrival and burst time for P3: 7 9


Process ArrTime BurstTime       CompTime        TurnArTime      WaitTime
P0      2       3               5               3               0
P1      5       6               13              8               2
P2      4       2               7               3               1
P3      7       9               22              15              6
Total Wait Time: 9
Total Turnaround Time: 29
Average Wait Time: 2.25
Average Turn Around Time: 7.25
```

HRN (Highest Response Ratio Next) Code:

```
#include<stdio.h>
#include<limits.h>

struct process{
    int id;
    int arrTime;
    int burstTime;
    int waitTime;
    int turnAroundTime;
    int compTime;
    int startTime;
    int isCompleted;
};

```

```c

int main() {
    int n;
    printf("Enter the number of processes: ");
    scanf("%d", &n);

    struct process processes[n];

    for(int i =0; i<n; i++){
        printf("Enter arrival and burst time for P%d: ", i+1);
        scanf("%d %d", &processes[i].arrTime, &processes[i].burstTime);
        processes[i].isCompleted= 0;
        processes[i].id =i+1;
    }

    int time= 0;
    int completed = 0;

    while(completed != n){
        int selectedIndex = -1;
        float highestRR= -9999;
        for(int i=0; i<n; i++){
            float responseRatio = (processes[i].burstTime + (time - processes[i].arrTime))
                                  / (float)processes[i].burstTime;

            if(processes[i].arrTime <= time && !processes[i].isCompleted &&
            responseRatio > highestRR){
                selectedIndex = i;
                highestRR = responseRatio;
            }
        }

        if(selectedIndex != -1){
            int j = selectedIndex;
            processes[j].startTime = time;
            time += processes[j].burstTime;
            processes[j].compTime= time;
            processes[j].turnAroundTime = processes[j].compTime- processes[j].arrTime;
            processes[j].waitTime= processes[j].turnAroundTime- processes[j].burstTime;
            processes[j].isCompleted= 1;
            completed++;
        }
        else{
            time++;
        }
    }

    int totalWaitTime= 0;
    int totalTurnAroundTime= 0;

    for(int i=0; i<n; i++){
        totalWaitTime+= processes[i].waitTime;
        totalTurnAroundTime += processes[i].turnAroundTime;
    }

    float avgTWT= (float) totalWaitTime/ n;
    float avgTAT= (float) totalTurnAroundTime/ n;

    printf("\n\nProcess\tArrTime\tBurstTime\tStartTime\tCompTime\tTurnArTime\tWaitTime");
    for(int i=0; i<n; i++){
        printf("\nP%d\t%d\t%d\t\t%d\t\t%d\t\t%d\t\t%d", processes[i].id, processes[i].arrTime,
        processes[i].burstTime, processes[i].startTime, processes[i].compTime,
        processes[i].turnAroundTime, processes[i].waitTime);
    }
```

```
79
80      printf("\nTotal Wait Time: %d", totalWaitTime);
81      printf("\nTotal Turnaround Time: %d", totalTurnAroundTime);
82      printf("\nAverage Wait Time: %.2f", avgTWT);
83      printf("\nAverage Turn Around Time: %.2f", avgTAT);
84
85      return 0;
86  }
```

HRN (Highest Response Ratio Next) Output :

```
Enter the number of processes: 5
Enter arrival and burst time for P1: 1 3
Enter arrival and burst time for P2: 3 6
Enter arrival and burst time for P3: 5 8
Enter arrival and burst time for P4: 7 4
Enter arrival and burst time for P5: 8 5


Process ArrTime BurstTime       StartTime       CompTime        TurnArTime      WaitTime
P1      1       3               1               4               3               0
P2      3       6               4               10              7               1
P3      5       8               19              27              22              14
P4      7       4               10              14              7               3
P5      8       5               14              19              11              6
Total Wait Time: 24
Total Turnaround Time: 50
Average Wait Time: 4.80
Average Turn Around Time: 10.00
```

Lab Assignment – 5

Aim: Simulation of CPU scheduling algorithms:

b) Pre Emptive:
- Round Robin
- SRTF (Shortest Remaining Time First)

Code & Output:

Round Robin Code:

```c
#include <stdio.h>

struct Process {
    int id;
    int arrival_time;
    int burst_time;
    int remaining_time;
    int complete_time;
    int waiting_time;
    int turnaround_time;
};

void roundRobin(struct Process processes[], int n, int quantum) {
    int time = 0;
    int completed = 0;

    while (completed < n) {

        for (int i = 0; i < n; i++) {
            if (processes[i].arrival_time <= time && processes[i].remaining_time > 0) {

                int execute_time;
                if(processes[i].remaining_time < quantum){
                    execute_time= processes[i].remaining_time;
                }
                else{
                    execute_time= quantum;
                }

                processes[i].remaining_time -= execute_time;
                time += execute_time;

                printf("Process %d: Executed for %d time units. Remaining time: %d\n", processes[i].id,
                execute_time, processes[i].remaining_time);

                if (processes[i].remaining_time == 0) {
                    processes[i].complete_time= time;
                    processes[i].turnaround_time= processes[i].complete_time- processes[i].arrival_time;
                    processes[i].waiting_time= processes[i].turnaround_time- processes[i].burst_time;
                    completed++;
                }
            }
        }
    }
}

int main() {
    int n, quantum;

    printf("Enter the number of processes: ");
```

```c
    scanf("%d", &n);

    struct Process processes[n];

    for (int i = 0; i < n; i++) {
        printf("Enter arrival time and burst time for process %d: ", i + 1);
        scanf("%d %d", &processes[i].arrival_time, &processes[i].burst_time);
        processes[i].id = i + 1;
        processes[i].remaining_time = processes[i].burst_time;
    }

    printf("Enter time quantum: ");
    scanf("%d", &quantum);

    printf("\nRound Robin Scheduling:\n");
    roundRobin(processes, n, quantum);

    // printing deatails
    printf("\nName\t\tArrival Time\tBurst Time\tEnd Time\tWaiting Time\tTurnaround Time\n");

    float totalTurnaroundTime = 0;
    float totalWaitingTime = 0;
    for (int i = 0; i < n; i++) {
        printf("P%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n", processes[i].id, processes[i].arrival_time,
        processes[i].burst_time, processes[i].complete_time, processes[i].waiting_time,
        processes[i].turnaround_time);
        totalTurnaroundTime += processes[i].turnaround_time;
        totalWaitingTime += processes[i].waiting_time;
    }

    printf("\nAverage Turnaround Time: %.2f", totalTurnaroundTime);
    printf("\nAverage Waiting Time: %.2f", totalWaitingTime);

    return 0;
}
```

Round Robin Output:

```
Enter the number of processes: 4
Enter arrival time and burst time for process 1: 0 8
Enter arrival time and burst time for process 2: 1 5
Enter arrival time and burst time for process 3: 2 10
Enter arrival time and burst time for process 4: 3 11
Enter time quantum: 6

Round Robin Scheduling:
Process 1: Executed for 6 time units. Remaining time: 2
Process 2: Executed for 5 time units. Remaining time: 0
Process 3: Executed for 6 time units. Remaining time: 4
Process 4: Executed for 6 time units. Remaining time: 5
Process 1: Executed for 2 time units. Remaining time: 0
Process 3: Executed for 4 time units. Remaining time: 0
Process 4: Executed for 5 time units. Remaining time: 0

Name            Arrival Time    Burst Time      End Time        Waiting Time    Turnaround Time
P1              0               8               25              17              25
P2              1               5               11              5               10
P3              2               10              29              17              27
P4              3               11              34              20              31

Average Turnaround Time: 93.00
Average Waiting Time: 59.00
```

SRTF (Shortest Remaining Time First) Code:

```c
    while (completed < n) {
        int selectedProcess = findSRTFProcess(processes, n, currentTime);

        if (selectedProcess == -1) {
            // idle, so time++
            printf("Idle ->");
            currentTime++;
        }
        else {
            // process executed, so time++
            processes[selectedProcess].remainingTime--;
            printf("P%d ->", processes[selectedProcess].id);
            currentTime++;

            if (processes[selectedProcess].remainingTime == 0) {
                // process is completed
                completed++;
                processes[selectedProcess].turnaroundTime = currentTime -
                                                            processes[selectedProcess].arrivalTime;
                processes[selectedProcess].waitingTime = processes[selectedProcess].turnaroundTime
                                                        - processes[selectedProcess].burstTime;
            }
        }
    }
    printf("All Processes Completed\n\n");
}

int main() {
    int n;
    printf("Enter the number of processes: ");
    scanf("%d", &n);

    struct Process processes[n];

    // input
    for (int i = 0; i < n; i++) {
        processes[i].id = i + 1;
        printf("Enter arrival time and burst time for process %d: ", i + 1);
        scanf("%d %d", &processes[i].arrivalTime, &processes[i].burstTime);
        processes[i].remainingTime = processes[i].burstTime;
    }

    SRTFScheduling(processes, n);

    // calculate total turn around time and total waiting time
    float totalTurnaroundTime = 0;
    float totalWaitingTime = 0;
    for (int i = 0; i < n; i++) {
        totalTurnaroundTime += processes[i].turnaroundTime;
        totalWaitingTime += processes[i].waitingTime;
        printf("Process %d - Turnaround Time: %d, Waiting Time: %d\n", processes[i].id,
        processes[i].turnaroundTime, processes[i].waitingTime);
    }

    // calculate average turnaround and waiting times
    printf("\nAverage Turnaround Time: %.2f\n", totalTurnaroundTime / n);
    printf("Average Waiting Time: %.2f\n", totalWaitingTime / n);

    return 0;
}
```

SRTF (Shortest Remaining Time First) Output:

```
Enter the number of processes: 3
Enter arrival time and burst time for process 1: 0 7
Enter arrival time and burst time for process 2: 1 3
Enter arrival time and burst time for process 3: 3 4


Gantt Chart:
P1 ->P2 ->P2 ->P2 ->P3 ->P3 ->P3 ->P3 ->P1 ->P1 ->P1 ->P1 ->P1 ->P1 ->All Processes Completed

Process 1 - Turnaround Time: 14, Waiting Time: 7
Process 2 - Turnaround Time: 3, Waiting Time: 0
Process 3 - Turnaround Time: 5, Waiting Time: 1

Average Turnaround Time: 7.33
Average Waiting Time: 2.67
```

Lab Assignment – 6

Aim: Simulation of Mutex and Semaphores :

- Producer – Consumer Problem

Code:

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

int mutex=1;//binary semaphore
int empty=5;//number of slots that are empty
int full=0;//slots that have been filled
int in=0,out=0;
int buffer[5];

int wait(int sem)
{
return(--sem);
}

int signal(int sem)
{
return (++sem);
}

void producer(int item)
{
mutex=wait(mutex);
full=signal(full);
empty=wait(empty);

buffer[in]=item;
in=(in+1)%10;
printf("Items produced by producer: %d \n",item);

mutex=signal(mutex);
}

void consumer()
{
mutex=wait(mutex);
empty=signal(empty);
full=wait(full);
int consumed=buffer[out];
buffer[out]=0;

printf("Items left: %d \n",full);
out=(out+1)%10;
mutex=signal(mutex);
}

int main()
{
int ch;
int item;

printf("\n1.Producer\n2.Consumer\n3.Exit\n");
while(1)
{
printf("Enter your choice \n");
scanf("%d",&ch);

switch(ch)
{
```

```
59    case 1:
60    if(mutex==1&&empty!=0)
61    {
62
63        printf("enter item\n");
64        scanf("%d",&item);
65    producer(item);
66    }
67    else
68    printf("No space left \n");
69    break;
70
71    case 2:
72    if(mutex==1&&full!=0)
73    consumer();
74    else
75    printf("No items left \n");
76    break;
77
78    case 3:
79    exit(0);
80    break;
81    }
82    }
83    }
```

Output:

```
kartikmittal@KARTIKWORLD:~$ vi producer.c
kartikmittal@KARTIKWORLD:~$ gcc producer.c
kartikmittal@KARTIKWORLD:~$ ./a.out

1.Producer
2.Consumer
3.Exit
Enter your choice
1
enter item
1
Items produced by producer: 1
Enter your choice
1
enter item
2
Items produced by producer: 2
Enter your choice
1
enter item
3
Items produced by producer: 3
Enter your choice
2
Items left: 2
Enter your choice
1
enter item
4
Items produced by producer: 4
Enter your choice
2
Items left: 2
```

Lab Assignment – 7

Aim: Simulation of Mutex and Semaphores :

- Dining Philosopher's Problem

Code:

```c
#include <pthread.h>
#include <semaphore.h>
#include <stdio.h>

#define N 5
#define THINKING 2
#define HUNGRY 1
#define EATING 0
#define LEFT (phnum + 4) % N
#define RIGHT (phnum + 1) % N

int state[N];
int phil[N] = { 0, 1, 2, 3, 4 };

sem_t mutex;
sem_t S[N];

void test(int phnum)
{
    if (state[phnum] == HUNGRY
        && state[LEFT] != EATING
        && state[RIGHT] != EATING) {
        // state that eating
        state[phnum] = EATING;

        sleep(2);

        printf("Philosopher %d takes fork %d and %d\n",
                    phnum + 1, LEFT + 1, phnum + 1);

```

```
31          printf("Philosopher %d is Eating\n", phnum + 1);
32
33          // sem_post(&S[phnum]) has no effect
34          // during takefork
35          // used to wake up hungry philosophers
36          // during putfork
37          sem_post(&S[phnum]);
38      }
39  }
40
41  // take up chopsticks
42  void take_fork(int phnum)
43  {
44
45      sem_wait(&mutex);
46
47      // state that hungry
48      state[phnum] = HUNGRY;
49
50      printf("Philosopher %d is Hungry\n", phnum + 1);
51
52      // eat if neighbours are not eating
53      test(phnum);
54
55      sem_post(&mutex);
56
57      // if unable to eat wait to be signalled
58      sem_wait(&S[phnum]);
59
60      sleep(1);
61  }
62
63  // put down chopsticks
64  void put_fork(int phnum)
65  {
66
67      sem_wait(&mutex);
68
69      // state that thinking
70      state[phnum] = THINKING;
71
72      printf("Philosopher %d putting fork %d and %d down\n",
73          phnum + 1, LEFT + 1, phnum + 1);
74      printf("Philosopher %d is thinking\n", phnum + 1);
75
76      test(LEFT);
77      test(RIGHT);
78
79      sem_post(&mutex);
80  }
81
82  void* philosopher(void* num)
83  {
84
85      while (1) {
86
87          int* i = num;
88
89          sleep(1);
```

```
        take_fork(*i);

        sleep(0);

        put_fork(*i);
    }
}

int main()
{

    int i;
    pthread_t thread_id[N];

    // initialize the semaphores
    sem_init(&mutex, 0, 1);

    for (i = 0; i < N; i++)

        sem_init(&S[i], 0, 0);

    for (i = 0; i < N; i++) {

        // create philosopher processes
        pthread_create(&thread_id[i], NULL,
                       philosopher, &phil[i]);

        printf("Philosopher %d is thinking\n", i + 1);
    }
```

Output:

```
Philosopher 1 is thinking
Philosopher 2 is thinking
Philosopher 3 is thinking
Philosopher 4 is thinking
Philosopher 5 is thinking
Philosopher 1 is Hungry
Philosopher 3 is Hungry
Philosopher 5 is Hungry
Philosopher 4 is Hungry
Philosopher 4 takes fork 3 and 4
Philosopher 4 is Eating
Philosopher 2 is Hungry
Philosopher 2 takes fork 1 and 2
Philosopher 2 is Eating
Philosopher 4 putting fork 3 and 4 down
Philosopher 4 is thinking
Philosopher 5 takes fork 4 and 5
Philosopher 5 is Eating
Philosopher 2 putting fork 1 and 2 down
Philosopher 2 is thinking
Philosopher 3 takes fork 2 and 3
Philosopher 3 is Eating
Philosopher 4 is Hungry
Philosopher 5 putting fork 4 and 5 down
Philosopher 5 is thinking
Philosopher 1 takes fork 5 and 1
Philosopher 1 is Eating
Philosopher 2 is Hungry
Philosopher 3 putting fork 2 and 3 down
Philosopher 3 is thinking
```

```
ayush@Ayush_Laptop:~/Lab10$ ./a.out

Philosopher 2 is thinking
Philosopher 2 try to pick left chopstick
Philosopher 2 try to pick right chopstick
Philosopher 2 is eating
Philosopher 3 is thinking
Philosopher 3 try to pick left chopstick
Philosopher 1 is thinking
Philosopher 0 is thinking
Philosopher 4 is thinking
Philosopher 2 Finished eating
Philosopher 3 try to pick right chopstick
Philosopher 3 is eating
Philosopher 1 try to pick left chopstick
Philosopher 1 try to pick right chopstick
Philosopher 1 is eating
Philosopher 0 try to pick left chopstick
Philosopher 0 try to pick right chopstick
Philosopher 3 Finished eating
Philosopher 1 Finished eating
Philosopher 0 is eating
Philosopher 4 try to pick left chopstick
Philosopher 4 try to pick right chopstick
Philosopher 0 Finished eating
Philosopher 4 is eating
Philosopher 4 Finished eating ayush@Ayush_Laptop:~/Lab10$
```

Lab Assignment – 8

Aim: Implementation of Banker's Algorithm

Code :

```c
#include <stdio.h>
int main()
{
    // P0, P1, P2, P3, P4 are the Process names here

    int n, m, i, j, k;
    n = 5; // Number of processes
    m = 3; // Number of resources
    int alloc[5][3] = {{0, 1, 0}, // P0 // Allocation Matrix
                       {2, 0, 0}, // P1
                       {3, 0, 2}, // P2
                       {2, 1, 1}, // P3
                       {0, 0, 2}}; // P4

    int max[5][3] = {{7, 5, 3}, // P0 // MAX Matrix
                     {3, 2, 2}, // P1
                     {9, 0, 2}, // P2
                     {2, 2, 2}, // P3
                     {4, 3, 3}}; // P4

    int avail[3] = {3, 3, 2}; // Available Resources

    int f[n], ans[n], ind = 0;
    for (k = 0; k < n; k++)
    {
        f[k] = 0;
    }
    int need[n][m];
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < m; j++)
            need[i][j] = max[i][j] - alloc[i][j];
    }
    int y = 0;
    for (k = 0; k < 5; k++)
    {
        for (i = 0; i < n; i++)
        {
            if (f[i] == 0)
            {
                int flag = 0;
                for (j = 0; j < m; j++)
                {
                    if (need[i][j] > avail[j])
                    {
                        flag = 1;
                        break;
                    }
                }
                if (flag == 0)
                {
                    ans[ind++] = i;
                    for (y = 0; y < m; y++)
                        avail[y] += alloc[i][y];
                    f[i] = 1;
                }
            }
        }
    }
    int flag = 1;
```

```
    int flag = 1;
    for (int i = 0; i < n; i++)
    {
        if (f[i] == 0)
        {
            flag = 0;
            printf("The following system is not safe");
            break;
        }
    }
    if (flag == 1)
    {
        printf("Following is the SAFE Sequence\n");
        for (i = 0; i < n - 1; i++)
            printf(" P%d ->", ans[i]);
        printf(" P%d", ans[n - 1]);
    }
    return (0);
}
```

Output:

```
kartikmittal@KARTIKWORLD:~$ vi banker.c
kartikmittal@KARTIKWORLD:~$ gcc banker.c
kartikmittal@KARTIKWORLD:~$ ./a.out
Following is the SAFE Sequence
 P1 -> P3 -> P4 -> P0 -> P2kartikmittal@KARTIKWORLD:~$
```

Lab Assignment – 9

Aim: Write C code for simulating the following Page Replacement Algorithm:

- FIFO
- LRU
- Optimal

Code & Output:

FIFO Page Replacement Algorithm Code:

```cpp
#include<iostream>
#include<unordered_map>
#include<queue>
using namespace std;

int main() {

    unordered_map<int,bool>mp;

    int n;
    cout<<"Enter no. of processes :"<<endl;
    cin>>n;

    queue<int>q;
    int p[n];

    for (int i = 0; i < n; i++)
    {
        cin>>p[i];
    }

    int f_size;
    cout<<"Enter frame size:"<<endl;
    cin>>f_size;

    int miss=0;
    int hit=0;
    for (int i = 0; i < n; i++)
    {
        if(mp[p[i]]==true){
            hit++;
            continue;
        }

        if(q.size()==f_size){
            int front=q.front();
            q.pop();
            mp[p[front]]=false;
        }

        q.push(p[i]);
        mp[p[i]]=true;

    }

    cout<<"No. of hits are: "<<hit<<endl;
    cout<<"No. of misses are: "<<n-hit<<endl;

    return 0;
}
```

FIFO Page Replacement Algorithm Output:

```
                                cd "c:\Users\HP\Desktop\college\OS\" ; if ($?) { g++ FIFO.cpp -o FIF
FO }
Enter no. of processes :
14
7 0 1 2 0 3 0 4 2 3 0 3 2 3
Enter frame size:
4
No. of hits are: 8
No. of misses are: 6
PS C:\Users\HP\Desktop\college\OS>
```

LRU Page Replacement Algorithm Code:

```cpp
#include <iostream>
#include <unordered_map>
#include <queue>
#include <climits>
using namespace std;

int find_min(vector<int> f, unordered_map<int, int> mp)
{
    int min = INT_MAX;
    int minindex = -1;
    for (int i = 0; i < f.size(); i++)
    {
        if (mp[f[i]] < min)
        {

            min = mp[f[i]];
            minindex = i;
        }
    }
    return minindex;
}

int main()
{

    unordered_map<int, int> mp;

    int n;
    cout << "Enter no. of processes :" << endl;
    cin >> n;
```

```
31
32      int p[n];
33      for (int i = 0; i < n; i++)
34      {
35          cin >> p[i];
36      }
37
38      int f_size;
39      cout << "Enter frame size:" << endl;
40      cin >> f_size;
41
42      vector<int> f;
43      int hit = 0;
44      for (int i = 0; i < n; i++)
45      {
46          if (mp[p[i]] > 0)
47          {
48              mp[p[i]] = i + 1;
49              hit++;
50              continue;
51          }
52
53          if (f.size() == f_size)
54          {
55
56              int minindex = find_min(f, mp);
57              mp[f[minindex]] = 0;
58              f[minindex] = p[i];
59              mp[p[i]] = i + 1;
60              continue;
61          }
62
63          f.push_back(p[i]);
64          mp[p[i]] = i + 1;
65      }
66
67      cout << "Hit count is :" << hit << endl;
68      cout << "Miss count is :" << n-hit << endl;
69      return 0;
70  }
```

LRU Page Replacement Algorithm Output:

```
Enter no. of processes :
14
7 0 1 2 0 3 0 4 2 3 0 3 2 3
Enter frame size:
4
Hit count is :8
Miss count is :6
PS C:\Users\HP\Desktop\college\OS>
```

Optimal Page Replacement Algorithm Code:

```cpp
#include <iostream>
#include <unordered_map>
#include <queue>
#include <climits>
using namespace std;

int find(vector<int> f, int p[], int n, int ci)
{

    unordered_map<int, int> mp;

    for (int i = 0; i < f.size(); i++)
    {
        mp[f[i]] = 0; // Initialize counts to zero
    }

    for (int i = ci; i < n; i++)
    {
        mp[p[i]]=n-i;
    }

    int min = INT_MAX;
    int minindex = -1;
    for (int i = 0; i < f.size(); i++)
    {
        if (mp[f[i]] < min)
        {
            min = mp[f[i]];
            minindex = i;
        }
    }
    return minindex;
}

int main()
{

    unordered_map<int, bool> check;

    int n;
    cout << "Enter no. of processes :" << endl;
    cin >> n;

    int p[n];
    for (int i = 0; i < n; i++)
    {
        cin >> p[i];
    }

    int f_size;
    cout << "Enter frame size:" << endl;
    cin >> f_size;

    vector<int> f;

    int hit = 0;
    for (int i = 0; i < n; i++)
    {
        if (check[p[i]])
        {
```

```
            hit++;
            continue;
        }
        if (f.size() == f_size)
        {
            int minindex = find(f, p, n, i);
            check[f[minindex]] = false;
            f[minindex] = p[i];
            check[p[i]] = true;
            continue;
        }
        f.push_back(p[i]);
        check[p[i]] = true;
    }

    cout << "Hit count are : " << hit << endl;
    cout << "Miss count are : " << n-hit << endl;

    return 0;
}
```

Optimal Page Replacement Algorithm Output:

```
                                cd "c:\Users\HP\Desktop\college\OS\" ; if ($?) { g++ optimal_page.cpp -o o
; if ($?) { .\optimal_page }
Enter no. of processes :
13
7 0 1 2 0 3 0 4 2 3 0 3 2
Enter frame size:
4
Hit count are : 7
Miss count are : 6
PS C:\Users\HP\Desktop\college\OS>
```

Lab Assignment – 10

Aim: Write C code for simulating the following Memory Management Algorithm:

- First Fit
- Best Fit
- Next Fit

Code & Output:

First Fit Code:

```cpp
#include <iostream>
#include <unordered_map>
using namespace std;

int main()
{

    int p;
    cout << "Enter the no. of processes:" << endl;
    cin >> p;

    unordered_map<int, int> mp;

    cout << "Enter the size of processes:" << endl;
    for (int i = 0; i < p; i++)
    {
        int p_size;
        cin >> p_size;
        mp[i] = p_size;
    }

    int blocks;
    cout << "Enter no. of blocks :" << endl;
    cin >> blocks;

    int arr[blocks];
    cout << "Enter the size of Blocks :" << endl;
    for (int i = 0; i < blocks; i++)
    {
        cin >> arr[i];
    }

    // Process starts
    int j = 0;
    while (j < p)
    {
        for (int i = 0; i < blocks; i++)
        {
            if (mp[j] <= arr[i])
            {
                arr[i] = arr[i] - mp[j];
                break;
            }
        }
            j++;
    }


    // Printing

    cout<<"Memory left after usage is : "<<endl;
    for (int i = 0; i < blocks; i++)
    {
        cout<<i<<"block has :"<<arr[i]<<" bytes left"<<endl;
    }
    cout<<endl;


    return 0;
}
```

First Fit Output:

```
                                    cd "c:\Users\HP\Desktop\college\OS\" ; if ($?) { g++ firs
$?) { .\first_fit }
Enter the no. of processes:
4
Enter the size of processes:
20 200 500 50
Enter no. of blocks :
4
Enter the size of Blocks :
30 50 200 700
Memory left after usage is :
0block has :10 bytes left
1block has :0 bytes left
2block has :0 bytes left
3block has :200 bytes left

PS C:\Users\HP\Desktop\college\OS>
```

Best Fit Code:

```cpp
#include <iostream>
#include <climits>
#include <unordered_map>
using namespace std;

int main()
{

    int p;
    cout << "Enter the no. of processes:" << endl;
    cin >> p;

    unordered_map<int, int> mp;

    cout << "Enter the size of processes:" << endl;
    for (int i = 0; i < p; i++)
    {
        int p_size;
        cin >> p_size;
        mp[i] = p_size;
    }

    int blocks;
    cout << "Enter no. of blocks :" << endl;
    cin >> blocks;

    int arr[blocks];
    cout << "Enter the size of Blocks :" << endl;
    for (int i = 0; i < blocks; i++)
    {
        cin >> arr[i];
    }

    // Process Starts

    int j = 0;
    while (j < p)
    {
        int min = INT_MAX;
        int minindex = -1;

        for (int i = 0; i < blocks; i++)
        {
```

```
            if (arr[i] < min && arr[i] >= mp[j])
            {
                min = arr[i];
                minindex = i;
            }
        }

        arr[minindex] = arr[minindex] - mp[j];
        j++;
    }

    cout << "Memory left after usage is : " << endl;
    cout << endl;
    for (int i = 0; i < blocks; i++)
    {
        cout << i << "block has :" << arr[i] << " bytes left" << endl;
    }
    cout << endl;
    return 0;
}
```

Best Fit Output:

```
Enter the no. of processes:

    4
Enter the size of processes:
    20 200 500 50
Enter no. of blocks :
    4
Enter the size of Blocks :
    30 50 200 700
Memory left after usage is :

0block has :10 bytes left
1block has :0 bytes left
2block has :0 bytes left
3block has :200 bytes left

PS C:\Users\HP\Desktop\college\OS>
```

Worst Fit Code:

```
#include <iostream>
#include <unordered_map>
#include <climits>
using namespace std;

int main()
{
    int p;
    cout << "Enter the no. of processes:" << endl;
    cin >> p;

    unordered_map<int, int> mp;
    cout << "Enter the size of processes:" << endl;

    for (int i = 0; i < p; i++)
    {
        int p_size;
        cin >> p_size;
        mp[i] = p_size;
    }

    int blocks;
    cout << "Enter no. of blocks :" << endl;
    cin >> blocks;

    int arr[blocks];
    cout << "Enter the size of Blocks :" << endl;
    for (int i = 0; i < blocks; i++)
    {
        cin >> arr[i];
```

```
    }
    // Process Starts

    int j = 0;
    while (j < p)
    {
        int max = INT_MIN;
        int maxindex = -1;

        for (int i = 0; i < blocks; i++)
        {
            if (arr[i] > max && arr[i] >= mp[j])
            {
                max = arr[i];
                maxindex = i;
            }
        }

        arr[maxindex] = arr[maxindex] - mp[j];
        j++;
    }

    cout << "Memory left after usage is : " << endl;
    for (int i = 0; i < blocks; i++)
    {
        cout << i << "block has :" << arr[i] << " bytes left" << endl;
    }
    cout << endl;
    return 0;
}
```

Worst Fit Output:

```
Enter the no. of processes:

3
Enter the size of processes:
30 100 45
Enter no. of blocks :
3
Enter the size of Blocks :
45 100 400
Memory left after usage is :
0block has :45 bytes left
1block has :100 bytes left
2block has :225 bytes left

PS C:\Users\HP\Desktop\college\OS>
```