Operating System Practicals

Name: Nishant Kumar Giri

Roll No.: AC-1254

Semester: III

Practical 1

Write a program (using fork () and/or exec () commands) where parent and child execute: a) same program, same code. b) same program, different code. c) before terminating, the parent waits for the child to finish its task.

CODE

(a) same program, same code.

```cpp
#include <iostream>     //input output stream
#include <unistd.h>     //it is used because fork() is defined in it
#include <sys/types.h>      //it is used for type pid_t for process ID

using namespace std;

int main()
{
    pid_t pid;          //creating process id using system id pid_t

    pid=fork();         //calling fork command

    cout<<"pid = "<<pid<<endl;  //printing process id, this statement will run for both child and process

    return 0;
}
```

OUTPUT

```
➜ OSPracticals g++ Practical1a.cpp -o Practical1a
➜ OSPracticals ./Practical1a
pid = 1275
pid = 0
➜ OSPracticals
```

CODE

(b) same program, different code.

```c
#include <sys/wait.h>
#include <stdio.h>
#include <unistd.h>
int main()
{
    pid_t pid, pid1;
    /* fork a child process */
    pid = fork();
    if (pid < 0)
    { /* error occurred */
        fprintf(stderr, "Fork Failed!");
        return 1;
    }
    else if (pid == 0)
    { /* child process */
        pid1 = getpid();
        printf("\nchild: pid = %d \n",pid); /* A */
        printf("child: pid1 = %d \n",pid1); /* B */
    }
    else
    { /* parent process */
        pid1 = getpid();
        printf("\nparent: pid = %d \n",pid); /* C */
        printf("parent: pid1 = %d \n",pid1); /* D */
        wait(NULL);
    }
    return 0;
}
```

OUTPUT

```
➜ OSPracticals g++ Practical1b.cpp -o Practical1b
➜ OSPracticals ./Practical1b

parent: pid = 1530
parent: pid1 = 1529

child: pid = 0
child: pid1 = 1530
➜ OSPracticals
```

CODE

(c) before terminating, the parent waits for the child to finish its task.

```c
#include <sys/wait.h>
#include <stdio.h>
#include <unistd.h>
int main()
{
    pid_t pid;
/* fork a child process */
    pid = fork();
    if (pid < 0) { /* error occurred */
        fprintf(stderr, "Fork Failed");
        return 1;
    }
    else if (pid == 0) { /* child process */
        printf("Child Process");
        printf("\nChild Process Terminated");
    }
    else { /* parent process */
        /* parent will wait for the child to complete */
        wait(NULL);
        printf("\nChild Complete \n");
    }
    return 0;
}
```

OUTPUT

```
➜ OSPracticals g++ Practical1c.cpp -o Practical1c
➜ OSPracticals ./Practical1c
Child Process
Child Process Terminated
Child Complete
➜ OSPracticals
```

Practical 2

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

CODE

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

int main()
{
    cout<<"\n----------CPU Information----------\n";
    system("cat /proc/cpuinfo | grep 'cpu family'");
    system("cat /proc/cpuinfo | grep 'model'");
    system("cat /proc/cpuinfo | grep 'vendor'");

    cout<<"\n----------KERNEL Information----------\n";
    system("cat /proc/sys/kernel/osrelease");


    return 0;
}
```

OUTPUT

```
➜ OSPracticals ./Practical2

----------CPU Information----------
cpu family      : 6
cpu family      : 6
cpu family      : 6
cpu family      : 6
model           : 78
model name      : Intel(R) Core(TM) i5-6300U CPU @ 2.40GHz
model           : 78
model name      : Intel(R) Core(TM) i5-6300U CPU @ 2.40GHz
model           : 78
model name      : Intel(R) Core(TM) i5-6300U CPU @ 2.40GHz
model           : 78
model name      : Intel(R) Core(TM) i5-6300U CPU @ 2.40GHz
vendor_id       : GenuineIntel
vendor_id       : GenuineIntel
vendor_id       : GenuineIntel
vendor_id       : GenuineIntel

----------KERNEL Information----------
5.10.16.3-microsoft-standard-WSL2
➜ OSPracticals
```

Practical 3

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

CODE

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

int main()
{
    cout << "----------MEMORY Information----------\n";
    system("cat /proc/meminfo | grep 'MemTotal'");
    system("cat /proc/meminfo | grep 'MemFree'");
    system("cat /proc/meminfo | grep 'MemAvailable'");

    cout << "\n\n";
    system("vmstat -s | grep 'total memory'");
    system("vmstat -s | grep 'used memory'");
    system("vmstat -s | grep 'free memory'");

    return 0;
}
```

OUTPUT

```
➜ OSPracticals g++ Practical3.cpp -o Practical3
➜ OSPracticals ./Practical3
----------MEMORY Information----------
MemTotal:        7810360 kB
MemFree:         7438176 kB
MemAvailable:    7501256 kB


      7810360 K total memory
        80980 K used memory
      7436640 K free memory
➜ OSPracticals
```

Practical 4

Write a program to print file details including owner access permissions, file access time, where file name is given as argument.

CODE

```cpp
#include<string.h>
#include<iostream>
using namespace std;

int main(int argc , char**argv)
{
    char str[100]="";

    strcat(str , "ls -l ");

    strcat(str ,argv[1]);

    strcat(str ,"| awk '{print $1 , $6 , $7 , $8 }'");

    system(str);

    return 0;
}
```

OUTPUT

```
➜ OSPracticals g++ Practical4.cpp -o Practical4
➜ OSPracticals ./Practical4 sample.txt
-rwxrwxrwx Nov 28 20:40
➜ OSPracticals
```

Practical 5

Write a program to copy files using system calls.

CODE

```cpp
#include <unistd.h> //read(),write(),open(),...
#include <fcntl.h>  //flags O_RDONLY(),...
#include <iostream> //input output stream
using namespace std;

int main(int argc, char *argv[])
{
    char buf[128];
    int src, des, n; //src is source file id, des is destinations and n for checking number of bits read and written
    if (argc != 3)
    {
        std::cerr << "Error! Two files expected";
        exit(1);
    }
    else
    {
        src = open(argv[1], O_RDONLY); //open a file in readonly
        if (src == -1)
        {
            perror("Error");
            exit(0);
        }
        else
        {
            des = open(argv[2], O_WRONLY | O_CREAT, 0640); //set's the flag of opening file as writeonly
            if (des == -1)
            {
                perror("Error");
                close(src); //close() is used to close a file
                exit(0);
            }
```

```
            else
            {
                while ((n = read(src, &buf, 128)) > 0) //reading from src file
and storing the characters in buf and their count in n. A max of 128 characters
can be read at a time
                {
                    if (write(des, &buf, n) != n) //if the number of characters
sent to be written is not same as number of written characters written in the
file then it will throw an error
                    {
                        perror("Error:");
                        close(src);
                        close(des);
                        exit(0);
                    }
                }
                write(STDOUT_FILENO, "copy operation completed!\n",30);
//STDOUT_FILENO is the macro storing value for console output
                close(src);
                close(des);
            }
        }
    }
    return 0;
}
```

OUTPUT

```
➜ OSPracticals g++ Practical5.cpp -o Practical5
➜ OSPracticals ./Practical5 sample.txt target.txt
copy operation completed!
➜ OSPracticals
```

Practical 6

Write program to implement FCFS scheduling algorithm.

CODE

```
//FCFS scheduling algorithm
#include<iostream>
using namespace std;

int main()
{
    int n;

    cout<<"Please enter the number of processes: ";
    cin>>n;

    int burst_time[n];
    for(int i=1; i<=n; i++)
    {
        cout<<"Please enter the Burst time for P"<<i<<": ";
        cin>>burst_time[i];
    }

    int wt_time[n];

    wt_time[1]=0;
    for(int i=2; i<=n; i++)                //calculating waiting time for each
process
    {
        wt_time[i]=wt_time[i-1]+burst_time[i-1];
    }

    int turnaround_time[n];
    for(int i=1; i<=n; i++)                //calculating turnarond time for
each process
    {
        turnaround_time[i]=wt_time[i]+burst_time[i];
    }

    float avg_wait_time=0, avg_turnaround_time=0;
```

```
    for(int i=1;i<=n;i++)
    {
        avg_wait_time+= wt_time[i];                 //calculating sum of waiting
time of all process
        avg_turnaround_time+= turnaround_time[i];       //calculating sum of
trunaround time of all process
    }

    cout<<"     Burst Time \tWaiting Time \tTurnaround Time"<<endl;
    for(int i=1; i<=n; i++)
    {
        cout<<"P"<<i+1<<"  \t"<<burst_time[i]<<"\t\t"<<wt_time[i]<<"\t\t"<<turna
round_time[i]<<endl;
    }


    avg_wait_time= avg_wait_time/n;
    avg_turnaround_time= avg_turnaround_time/n;

    cout<<"\nAverage Waiting time = "<<avg_wait_time<<endl;
    cout<<"\nAverage Turnaround time = "<<avg_turnaround_time<<endl;
    return 0;
}
```

OUTPUT

```
➜ OSPracticals g++ Practical6.cpp -o Practical6
➜ OSPracticals ./Practical6
Please enter the number of processes: 4
Please enter the Burst time for P1: 21
Please enter the Burst time for P2: 7
Please enter the Burst time for P3: 14
Please enter the Burst time for P4: 3
     Burst Time         Waiting Time    Turnaround Time
P2      21              0               21
P3      7               21              28
P4      14              28              42
P5      4               42              46

Average Waiting time = 22.75

Average Turnaround time = 34.25
➜ OSPracticals
```

Practical 7

Write program to implement Round Robin scheduling algorithm

CODE

```
#include <iostream>
using namespace std;
//Ready Queue

void queueUpdation(int queue[], int timer, int arrival[], int n, int
maxProccessIndex)
{
    int zeroIndex;
    for (int i = 0; i < n; i++)
    {
        if (queue[i] == 0)
        {
            zeroIndex = i;
            break;
        }
    }
    queue[zeroIndex] = maxProccessIndex + 1;
}
//queue maintained
void queueMaintainence(int queue[], int n)
{
    for (int i = 0; (i < n - 1) && (queue[i + 1] != 0); i++)
    {
        int temp = queue[i];
        queue[i] = queue[i + 1];
        queue[i + 1] = temp;
    }
}

void checkNewArrival(int timer, int arrival[], int n, int maxProccessIndex, int
queue[])
```

```
{
    if (timer <= arrival[n - 1])
    {
        bool newArrival = false;
        for (int j = (maxProccessIndex + 1); j < n; j++)
        {
            if (arrival[j] <= timer)
            {
                if (maxProccessIndex < j)
                {
                    maxProccessIndex = j;
                    newArrival = true;
                }
            }
        }
        //adds the incoming process to the ready queue

        if (newArrival)
        {
            queueUpdation(queue, timer, arrival, n, maxProccessIndex);
        }
    }
}

int main()
{
    int n, tq, timer = 0, maxProccessIndex = 0;
    float avgWait = 0, avgTT = 0;
    cout << "\nPlease enter the Time Quantum : ";
    cin >> tq;
    cout << "\nPlease enter the number of processes : ";
    cin >> n;
    int arrival[n], burst[n], wait[n], turn[n], queue[n], temp_burst[n];
    bool complete[n];

    cout << "\nPlease enter the Arrival Time (in ascending order) : ";
    for (int i = 0; i < n; i++)
    {
        cin >> arrival[i];
    }
```

```
    cout << "\nPlease enter the CPU Burst Time of the processes : ";
    for (int i = 0; i < n; i++)
    {
        cin >> burst[i];
        temp_burst[i] = burst[i];
    }

    for (int i = 0; i < n; i++)
    { //Initializing the queue and complete array
        complete[i] = false;
        queue[i] = 0;
    }

    while (timer < arrival[0]) //Incrementing Timer until the first process
arrives
    {
        timer++;
    }
    queue[0] = 1;

    while (true)
    {
        bool flag = true;
        for (int i = 0; i < n; i++)
        {
            if (temp_burst[i] != 0)
            {
                flag = false;
                break;
            }
        }
        if (flag)
        {
            break;
        }

        for (int i = 0; (i < n) && (queue[i] != 0); i++)
        {
            int ctr = 0;
            while ((ctr < tq) && (temp_burst[queue[0] - 1] > 0))
            {
```

```
                temp_burst[queue[0] - 1] -= 1;
                timer += 1;
                ctr++;

                //Checking and Updating the ready queue until all the processes
arrive
                checkNewArrival(timer, arrival, n, maxProccessIndex, queue);
            }

            if ((temp_burst[queue[0] - 1] == 0) && (complete[queue[0] - 1] ==
false))
            {
                //turn array currently stores the completion time
                turn[queue[0] - 1] = timer;
                complete[queue[0] - 1] = true;
            }

            bool idle = true;
            if (queue[n - 1] == 0)
            {
                for (int i = 0; i < n && queue[i] != 0; i++)
                {
                    if (complete[queue[i] - 1] == false)
                    {
                        idle = false;
                    }
                }
            }
            else
            {
                idle = false;
            }
            if (idle)
            {
                timer++;
                checkNewArrival(timer, arrival, n, maxProccessIndex, queue);
            }

            //Maintaining the entries of processes
            //after each premption in the ready Queue
            queueMaintainence(queue, n);
```

```
        }
    }

    for (int i = 0; i < n; i++)
    {
        turn[i] = turn[i] - arrival[i];
        wait[i] = turn[i] - burst[i];
    }

    cout << "\nProcesses\tArrival Time\tCPU Burst Time\tWaiting Time\tTurnaround
Time" << endl;
    for (int i = 0; i < n; i++)
    {
        cout << i + 1 << "\t\t" << arrival[i] << "\t\t"
             << burst[i] << "\t\t" << wait[i] << "\t\t" << turn[i] << endl;
    }
    for (int i = 0; i < n; i++)
    {
        avgWait += wait[i];
        avgTT += turn[i];
    }
    cout << "\nAverage Waiting Time : " << (avgWait / n)
         << "\nAverage Turn Around Time : " << (avgTT / n);

    return 0;
}
```

OUTPUT

```
➜ OSPracticals g++ Practical7.cpp -o Practical7
➜ OSPracticals ./Practical7

Please enter the Time Quantum : 3

Please enter the number of processes : 3

Please enter the Arrival Time (in ascending order) : 0
2
3

Please enter the CPU Burst Time of the processes : 10
20
30

Processes       Arrival Time    CPU Burst Time  Waiting Time    Turnaround Time
1               0               10              18              28
2               2               20              26              46
3               3               30              27              57

Average Waiting Time : 23.6667
Average Turn Around Time : 43.6667%
➜ OSPracticals
```

Practical 8

Write program to implement SJF scheduling algorithm

CODE

```
//SJF scheduling algorithm

#include<iostream>
using namespace std;
int main()
{
    int n;
    cout<<"Please enter the number of processes: ";
    cin>>n;

    int burst[n],process[n];

    for(int i=0; i<n; i++)
    {
        cout<<"Please enter the CPU Burst Time for process P"<<i+1<<" :";
        cin>>burst[i];
        process[i]=i+1;
    }

    int j,k;
    int temp1,temp2;
    for(j=1; j<n; j++)                  //sorting burst time and swapping elements
of process[] along with burst[]
    {
        temp1=burst[j];
        temp2=process[j];
        for(k=j; k>0 && temp1<burst[k-1]; k--)
        {
            burst[k]=burst[k-1];
            process[k]=process[k-1];
        }
```

```
        burst[k]=temp1;
        process[k]=temp2;
    }


    int wait_time[n],turnaround_time[n];

    wait_time[0]=0;
    for(int i=1; i<n; i++)
        wait_time[i]=wait_time[i-1]+burst[i-1];     //calculating wait time

    for(int i=0; i<n; i++)
        turnaround_time[i]=wait_time[i]+burst[i];       //calculating turnaround
time



    float avg_wt=0,avg_tt=0;

    cout<<"Processes \t  Burst Time \t   Waiting Time  \t  Turnaround
Time"<<endl;       //printing wait time & turnaround time
    for(int i=0; i<n; i++)
    {
        cout<<"P"<<process[i]<<"  \t\t   "<<burst[i]<<"   \t\t      "<<wait_time
[i]<<"   \t\t\t   "<<turnaround_time[i]<<endl;
        avg_wt+=wait_time[i];               //calculating total sum of wait time
        avg_tt+=turnaround_time[i];         //calculating total sum of
turnaround time
    }

    avg_wt=avg_wt/n;                //calculating avg wait time
    avg_tt=avg_tt/n;                //calculating avg turnaround time

    cout<<"\nAverage Waiting Time = "<<avg_wt<<endl;
    cout<<"\nAverage Turnaround Time = "<<avg_tt<<endl;

    return 0;
}
```

OUTPUT

```
➜ OSPracticals g++ Practical8.cpp -o Practical8
➜ OSPracticals ./Practical8
Please enter the number of processes: 4
Please enter the CPU Burst Time for process P1 :21
Please enter the CPU Burst Time for process P2 :69
Please enter the CPU Burst Time for process P3 :42
Please enter the CPU Burst Time for process P4 :22
Processes         Burst Time       Waiting Time          Turnaround Time
P1                 21                          0                          21
P4                 22                          21                         43
P3                 42                          43                         85
P2                 69                          85                         154

Average Waiting Time = 37.25

Average Turnaround Time = 75.75
➜ OSPracticals
```

Practical 9

Write program to implement non-preemptive priority based scheduling algorithm

CODE

```cpp
//non-preemptive priority scheduling algorithm
#include <bits/stdc++.h>
#include <iostream>
using namespace std;

struct Process
{
    int pid;
    int bt;
    int priority;
};

bool comparison(Process a, Process b)
{
    return (a.priority > b.priority);
}

void findWaitingTime(Process proc[], int n, int wt[])
{
    wt[0] = 0;

    for (int i = 1; i < n; i++)
        wt[i] = proc[i - 1].bt + wt[i - 1];
}

void findTurnAroundTime(Process proc[], int n, int wt[], int tat[])
{
    for (int i = 0; i < n; i++)
        tat[i] = proc[i].bt + wt[i];
}

void findavgTime(Process proc[], int n)
```

```
{
    int wt[n], tat[n], total_wt = 0, total_tat = 0;
    findWaitingTime(proc, n, wt);
    findTurnAroundTime(proc, n, wt, tat);

    cout << "\nProcesses  "
         << " CPU Burst time  "
         << " Waiting time  "
         << " Turn around time\n";

    for (int i = 0; i < n; i++)
    {
        total_wt = total_wt + wt[i];
        total_tat = total_tat + tat[i];
        cout << "    " << proc[i].pid << "\t\t" << proc[i].bt << "\t    " <<
wt[i] << "\t\t  " << tat[i] << endl;
    }

    cout << "\nAverage Waiting Time = " << (float)total_wt / (float)n;
    cout << "\nAverage Turn around Time = " << (float)total_tat / (float)n;
}

void priorityScheduling(Process proc[], int n)
{
    std::sort(proc, proc + n, comparison);

    cout << "\nOrder of execution: ";
    for (int i = 0; i < n; i++)
    {
        cout << proc[i].pid << " ";
    }
    cout<< endl;
    findavgTime(proc, n);
}

int main()
{
    int n;
    cout << "\nPriority Scheduling\nPlease enter the number of Processes = ";
    cin >> n;
    Process *proc = new Process[n];
```

```
    for (int i = 0; i < n; i++)
    {
        cout << "\nPlease enter the CPU Burst Time for Process P" << i + 1 << "=
";
        cin >> proc[i].bt;
        cout << "Please enter the Priority of Process P" << i + 1 << "= ";
        cin >> proc[i].priority;
        proc[i].pid = i + 1;
    }
    priorityScheduling(proc, n);
    return 0;
}
```

OUTPUT

```
➜ OSPracticals g++ Practical9.cpp -o Practical9
➜ OSPracticals ./Practical9

Priority Scheduling
Please enter the number of Processes = 3

Please enter the CPU Burst Time for Process P1= 4
Please enter the Priority of Process P1= 2

Please enter the CPU Burst Time for Process P2= 6
Please enter the Priority of Process P2= 1

Please enter the CPU Burst Time for Process P3= 8
Please enter the Priority of Process P3= 4

Order of execution: 3 1 2

Processes   CPU Burst time   Waiting time   Turn around time
   3            8           0              8
   1            4           8              12
   2            6           12             18

Average Waiting Time = 6.66667
Average Turn around Time = 12.6667%
➜ OSPracticals
```

Practical 10

Write program to implement preemptive priority based scheduling algorithm

CODE

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

int main()
{
    int n;
    cout << "Please enter the number of processes: ";
    cin >> n;
    float total, wait[n];
    float p[n], twaiting = 0, waiting = 0;
    int proc;
    int stack[n];
    float brust[n], arrival[n], sbrust, temp[n], top = n, prority[n];
    int i;

    for (i = 0; i < n; i++)
    {
        p[i] = i;
        stack[i] = i;
        cout << "\nPlease enter the Arrival Time: ";
        cin >> arrival[i];
        cout << "Please enter the CPU Brust Time: ";
        cin >> brust[i];
        cout << "Please enter the Priority Time: ";
        cin >> prority[i];
        temp[i] = arrival[i];
        sbrust = brust[i] + sbrust;
    }

    for (i = 0; i < sbrust; i++)
```

```
    {
        //section 1
        proc = stack[0];
        if (temp[proc] == i)
            twaiting = 0;
        else
            twaiting = i - (temp[proc]);
        temp[proc] = i + 1;
        wait[proc] = wait[proc] + twaiting;
        waiting = waiting + (twaiting);
        brust[proc] = brust[proc] - 1;

        if (brust[proc] == 0)
        {
            for (int x = 0; x < top - 1; x++)
                stack[x] = stack[x + 1];
            top = top - 1;
        }

        for (int z = 0; z < top - 1; z++)
        {
            if ((prority[stack[0]] > prority[stack[z + 1]]) && (arrival[stack[z
+ 1]] <= i + 1))
            {
                int t = stack[0];
                stack[0] = stack[z + 1];
                stack[z + 1] = t;
            }
        }
    }

    cout << "\nAverage Waiting Time : " << waiting / n;
    float tu = (sbrust + waiting) / n;
    cout << endl
         << "Average Turnaround Time : " << tu << endl;
    return 0;
}
```

OUTPUT

```
PS C:\Users\nisha\Desktop\OSPracticals> g++ .\Practical10.cpp -o .\Practical10
PS C:\Users\nisha\Desktop\OSPracticals> .\Practical10
Please enter the number of processes: 3

Please enter the Arrival Time: 0
Please enter the CPU Brust Time: 2
Please enter the Priority Time: 1

Please enter the Arrival Time: 0
Please enter the CPU Brust Time: 3
Please enter the Priority Time: 4

Please enter the Arrival Time: 0
Please enter the CPU Brust Time: 1
Please enter the Priority Time: 2

Average Waiting Time : 1.66667
Average Turnaround Time : 3.66667
PS C:\Users\nisha\Desktop\OSPracticals>
```

Practical 11

Write program to implement SRJF scheduling algorithm

CODE

```
#include <iostream>

using namespace std;

void waiting_time(struct process a[], int n);

struct process
{

    int process_id;
    int burst_time;
    int waiting_time;
    int arrival_time;
    int remain_time;

} arr[100];

int process_finish[100];

int main()
{
    arr[99].remain_time = 9999;

    int n; //No of process in variable n

    cout << "\nPlease enter the number of Processes : ";
```

```
    cin >> n;
    cout << endl;

    for (int i = 0; i < n; i++) //Take the Burst time for each process by using
loop
    {

        arr[i].process_id = i + 1; //increment the process_id by 1 after each
burst_time

        cout << "Please enter the CPU Burst Time of P" << i + 1 << " : ";

        cin >> arr[i].burst_time;

        arr[i].remain_time = arr[i].burst_time; //copy each process burst_time
to another array remain_time[]

        cout << "Please enter the Arrival Time : ";

        cin >> arr[i].arrival_time;
        cout << endl;
    }

    waiting_time(arr, n);
    return 0;
}

void waiting_time(struct process a[], int n)

{

    int remain = 0, sum_wait = 0, sum_turnaround = 0, endTime, smallest;

    cout << "\n\nProcess  Turnaround Time  Waiting Time\n\n";

    int process_f = 0; // handle the INDEX of array process_finish.

    for (int time = 0; remain != n; time++)
    {
        smallest = 99;
```

```
        for (int i = 0; i < n; i++)
        {
            if ((a[i].arrival_time <= time) && (a[i].remain_time <
a[smallest].remain_time) && (a[i].remain_time > 0))
            {
                smallest = i;
            }
        }

        a[smallest].remain_time--;

        if (a[smallest].remain_time == 0)
        {
            process_finish[process_f] = smallest + 1; //to assign a process #
which finish the total job
            process_f++;
            a[smallest].process_id = smallest + 1; //to ssign a process_id

            int tt;

            remain++; //One process complete the total job

            endTime = time + 1; //Total competional time of process

            tt = endTime - a[smallest].arrival_time; //Calculate the TURNaround
TIME (competionalTime - TT )

            a[smallest].waiting_time = tt - a[smallest].burst_time; //Calculate
the Waiting Time

            cout << "\nP[" << smallest + 1 << "]\t\t" << tt << "\t\t" <<
a[smallest].waiting_time;

            sum_wait += tt - a[smallest].burst_time; //For find Average Waiting
Time
        }
    }

    cout << "\n\nAverage Waiting Time = " << sum_wait * 1.0 / n;
}
```

OUTPUT

```
➜ OSPracticals g++ Practical11.cpp -o Practical11
➜ OSPracticals ./Practical11

Please enter the number of Processes : 3

Please enter the CPU Burst Time of P1 : 10
Please enter the Arrival Time : 1

Please enter the CPU Burst Time of P2 : 20
Please enter the Arrival Time : 2

Please enter the CPU Burst Time of P3 : 30
Please enter the Arrival Time : 3



Process  Turnaround Time  Waiting Time


P[1]            10              0
P[2]            29              9
P[3]            58              28

Average Waiting Time = 12.3333%
➜ OSPracticals
```

Practical 12

Write program to calculate sum of n numbers using thread library.

CODE

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

int sum;
void *runner(void *param);  // threads calls this function

int main(int argc , char *argv[])
{
    pthread_t tid;
    pthread_attr_t attr;
    if(argc!=2)
    {
        fprintf(stderr,"Usage : a.out<integer value>\n");
        return -1;
    }
    if(atoi(argv[1])<0)
    {
        fprintf(stderr,"%d must be >=0\n",atoi(argv[1]));
        return -1;
    }
    pthread_attr_init(&attr);
    pthread_create(&tid,&attr,runner,argv[1]);
    pthread_join(tid,NULL);
    printf("Sum = %d\n",sum);
}
```

```c
void *runner(void *param)
{
    int i , upper=atoi(param);
    sum=0;
    for(i=1 ;i<=upper ;i++)
    {
        sum += i;
    }
    pthread_exit(0);
}
```

OUTPUT

```
➜ OSPracticals gcc Practical12.c -o Practical12 -lpthread
➜ OSPracticals ./Practical12 5
Sum = 15
➜ OSPracticals ./Practical12 7
Sum = 28
➜ OSPracticals
```

Practical 13

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

CODE

(a) first fit

```
/* program to implement first-fit allocation strategies */

#include <iostream>
using namespace std;

int main()
{ // main function starts
    int MemoryBlock[10], Process[10], NumberOfBlock, NumberOfProcess,
flags[10],
        allocation[10], i, j;

    for (i = 0; i < 10; i++)
    { // updating initial allocation status
        flags[i] = 0;
        allocation[i] = -1;
    }

    cout << "Please enter the number of Memory Blocks: ";
    cin >> NumberOfBlock; // enter number of memory block

    cout << "\nPlease enter the Size of each Memory Block: ";
    for (i = 0; i < NumberOfBlock; i++)
    {
        cin >> MemoryBlock[i];
    } // enter size of each memory block
```

```
    cout << "\nPlease enter the number of Processes: ";
    cin >> NumberOfProcess; // enter number of processes

    cout << "\nPlease enter each Process size: ";
    for (i = 0; i < NumberOfProcess; i++)
    {
        cin >> Process[i];
    } // enter size of each process

    /* allocating according to first fit strategies */
    for (i = 0; i < NumberOfProcess;
         i++)
    { // comparing each process to each memory block
        for (j = 0; j < NumberOfBlock; j++)
        {
            if (flags[j] == 0 && MemoryBlock[j] >= Process[i])
            {
                /* if the mem block is not allocated and size of process is
less
                than mem block it will be allocated */

                allocation[j] = i; /* updating status of memory block to
                                      allocated and storing process number */
                flags[j] = 1;
                break;
            }
        }
    }
    /* displaying gannt chart table */
    cout << "\nBlock no.\tSize\t\tProcess number.\t\t Process Size";
    for (i = 0; i < NumberOfBlock; i++)
    {
        cout << "\n"
             << i + 1 << "\t\t" << MemoryBlock[i] << "\t\t";
        if (flags[i] == 1)
        {
            cout << allocation[i] + 1 << "\t\t\t" << Process[allocation[i]];
        }
        else
        {
            cout << "Not allocated";
```

```
        }
    }
    return 0;
}
```

OUTPUT

```
➜ OSPracticals g++ Practical13FirstFit.cpp -o Practical13FirstFit
➜ OSPracticals ./Practical13FirstFit
Please enter the number of Memory Blocks: 3

Please enter the Size of each Memory Block: 200
400
60

Please enter the number of Processes: 3

Please enter each Process size: 300
25
125

Block no.       Size            Process number.          Process Size
1               200             2                       25
2               400             1                       300
3               60              Not allocated%
➜ OSPracticals
```

CODE

(b) best fit

```
/* program to implement best-fit allocation strategies */

#include <iostream>  //input output
using namespace std; //standard namespace
int main()
{ //main function
    int MemoryBlock[10], Processes[10], numberOfMemoryBlocks, numberOfProc,
        flags[10], allocation[10];
    int i, j, smallest;
    //setting intial status of memory block to not allocated
    for (i = 0; i < 10; i++)
    {
        flags[i] = 0;
        allocation[i] = -1;
    }

    cout << "Please enter the number of Memory Partitions: ";
    cin >> numberOfMemoryBlocks; //enter number of mem block
    cout << "\nPlease enter size of each partiton: ";
    for (i = 0; i < numberOfMemoryBlocks; i++)
    {
        cin >> MemoryBlock[i];
    } //enter size of each memory block

    cout << "\nPlease enter number of processes: ";
    cin >> numberOfProc; //enter number of processess
    cout << "\nPlease enter the size of each process: ";
    for (i = 0; i < numberOfProc; i++)
    {
```

```
        cin >> Processes[i];
    } //enter size of each process

    // allocation as per best fit
    for (i = 0; i < numberOfProc; i++)
    {                   //comparing each process to each mem block
        smallest = -1; //initiating smallest memory block
        for (j = 0; j < numberOfMemoryBlocks; j++)
            if (flags[j] == 0 && MemoryBlock[j] >= Processes[i])
            {
                smallest = j;
                break;
            }
        for (j = 0; j < numberOfMemoryBlocks; j++)
        {
            if (flags[j] == 0 && MemoryBlock[j] >= Processes[i] &&
                MemoryBlock[j] < MemoryBlock[smallest])
                smallest = j;
        }
        if (smallest != -1)
        {
            allocation[smallest] = i;
            flags[smallest] = 1;
        }
    }

    /* displaying details */
    cout << "\nPartition\tSize\tProcess No.\tSize";
    for (i = 0; i < numberOfMemoryBlocks; i++)
    {
        cout << "\n"
             << i + 1 << "\t\t" << MemoryBlock[i] << "\t";
        if (flags[i] == 1)
            cout << allocation[i] + 1 << "\t\t" << Processes[allocation[i]];
        else
            cout << "Not allocated";
    }
    cout << endl;
    return 0;
}
```

OUTPUT

```
➜ OSPracticals g++ Practical13BestFit.cpp -o Practical13BestFit
➜ OSPracticals ./Practical13BestFit
Please enter the number of Memory Partitions: 3

Please enter size of each partiton: 200
400
60

Please enter number of processes: 3

Please enter the size of each process: 300
25
125

Partition       Size    Process No.     Size
1               200     3               125
2               400     1               300
3               60      2               25
➜ OSPracticals
```

CODE

(c) worst fit

```cpp
/* program to implement worst-fit allocation strategies */

#include <iostream>  //input output stream
using namespace std; // standard namespace

int main()
{ // main function
    int NumberOfBlock, NumberOfProcess, MemoryBlock[20], Processes[20];

    cout << " Please enter the number of Memory Blocks: ";
    cin >> NumberOfBlock; // enter number of blocks

    cout << " Please enter the number of processes: ";
    cin >> NumberOfProcess; // enter number of processes

    cout << " Please enter the size of " << NumberOfBlock << " blocks: ";
    for (int i = 0; i < NumberOfBlock; i++)
    {
        cin >> MemoryBlock[i];
    } // enter size of each mem block

    cout << " Please enter the size of " << NumberOfProcess << " processes: ";
    for (int i = 0; i < NumberOfProcess; i++)
    {
        cin >> Processes[i];
    } // enter size of each processes

    // performing worst fit allocation strategies
```

```
    for (int i = 0; i < NumberOfProcess; i++)
    {
        /* comparing each process with each memory block */
        int max = MemoryBlock[0];
        int pos = 0;
        for (int j = 0; j < NumberOfBlock; j++)
            if (max < MemoryBlock[j])
            {
                max = MemoryBlock[j];
                pos = j;
            }
        /* displaying details */
        if (max >= Processes[i])
        {
            cout << "\nProcess " << i + 1 << " is allocated to block "
                 << pos + 1;
            MemoryBlock[pos] = MemoryBlock[pos] - Processes[i];
        }
        else
        {
            cout << "\nProcess " << i + 1 << " can't be allocated!";
        }
    }
    cout << endl;
    return 0;
}
```

OUTPUT

```
➜ OSPracticals g++ Practical13WorstFit.cpp -o Practical13WorstFit
➜ OSPracticals ./Practical13WorstFit
 Please enter the number of Memory Blocks: 3
 Please enter the number of processes: 3
 Please enter the size of 3 blocks: 200
400
60
 Please enter the size of 3 processes: 300
125
25

Process 1 is allocated to block 2
Process 2 is allocated to block 1
Process 3 is allocated to block 2
➜ OSPracticals
```