VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.

LAB REPORT

on

Operating Systems

(23CS4PCOPS)

Submitted by:

AKSHAY RAJ ARYAN (1BM22CS032)

in partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

B.M.S. COLLEGE OF ENGINEERING

(Autonomous Institution under VTU)

BENGALURU-560019

June 2024 - August 20224

B. M. S. College of Engineering,
Bull Temple Road, Bangalore 560019
(Affiliated To Visvesvaraya Technological University, Belgaum)
Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the Lab work entitled "**Operating Systems**" carried out by **XXXXXX (1BM21CS069),** who is bonafide student of **B. M. S. College of Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2022-23. The Lab report has been approved as it satisfies the academic requirements in respect of **Operating Systems - (23CS4PCOPS)** work prescribed for the said degree.

Basavaraj Jakkalli
Associate Professor
Department of CSE
BMSCE, Bengaluru

Dr. Jyothi S Nayak
Professor and Head
Department of CSE
BMSCE, Bengaluru

Table Of Contents

Course Outcomes

CO1: Apply the different concepts and functionalities of Operating System.

CO2: Analyse various Operating system strategies and techniques.

CO3: Demonstrate the different functionalities of Operating System.

CO4: Conduct practical experiments to implement the functionalities of Operating system.

1. Experiments

1.1 Experiment - 1

1.1.1 Question:

Write a C program to simulate the following non-pre-emptive CPU scheduling algorithm to find turnaround time and waiting time.

(a) FCFS

(b) SJF

1.1.2 Code:

```
#include<stdio.h>
   int n, i, j, pos, temp, choice, Burst_time[20], Waiting_time[20],
   Turn_around_time[20], process[20], total=0;
   float avg_Turn_around_time=0, avg_Waiting_time=0;

int FCFS()
{
   Waiting_time[0]=0;

   for(i=1;i<n;i++)
   {
   printf("\nAverage Turnaround Time:%.2f\n",avg_Turn_around_time);

   return 0;
}

int SJF()
{
   //sorting
   for(i=0;i<n;i++)
   {
      pos=i;
      for(j=i+1;j<n;j++)
      {
         if(Burst_time[j]<Burst_time[pos])
            pos=j;
      }

      temp=Burst_time[i];
      Burst_time[i]=Burst_time[pos];
      Burst_time[pos]=temp;
```

```
    temp=process[i];
    process[i]=process[pos];
    process[pos]=temp;
}
    Waiting_time[0]=0;


for(i=1;i<n;i++)
{
    Waiting_time[i]=0;

    for(j=0;j<i;j++)
        Waiting_time[i]+=Burst_time[j];

    total+=Waiting_time[i];
}

avg_Waiting_time=(float)total/n;
total=0;

printf("\nProcess\t\tBurst Time\t\tWaiting Time\t\tTurnaround Time");

for(i=0;i<n;i++)
{
    Turn_around_time[i]=Burst_time[i]+Waiting_time[i];
    total+=Turn_around_time[i];
    printf("P[%d]:",i+1);
    scanf("%d",&Burst_time[i]);
    process[i]=i+1;
}

while(1)
{   printf("\n-----MAIN MENU-----\n");
    printf("1. FCFS Scheduling\n2. SJF Scheduling\n");
    printf("\nEnter your choice:");
    scanf("%d", &choice);
    switch(choice)
    {
        case 1: FCFS();
        break;

        case 2: SJF();
        break;

        default: printf("Invalid Input!!!");
```

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

1.1.3 Output:

a.

```
ArrivalTime.c -o FCFS_ArrivalTime } ; if ($?) { .\FCFS_ArrivalTime }
Enter the number of processes: 4
Enter the process ids:
1 2 3 4
Enter arrival time and burst time for process 1: 0 8
Enter arrival time and burst time for process 2: 1 4
Enter arrival time and burst time for process 3: 2 9
Enter arrival time and burst time for process 4: 3 5

Process Arrival Time    Burst Time      Waiting Time    Turnaround Time
1       0               8               0               8
2       1               4               7               11
3       2               9               10              19
4       3               5               18              23

Average Waiting Time: 8.75
Average Turnaround Time: 15.25
PS C:\Users\Nisarga Gondi\OneDrive\Desktop\Nisarga\lV SEM\OS 4th sem\os lab>
```

b.

```
P.c -o SJF_NP } ; if ($?) { .\SJF_NP }
Enter the number of processes:
4
Enter the burst time of process 1:
8
Enter the burst time of process 2:
4
Enter the burst time of process 3:
9
Enter the burst time of process 4:
5
BurstTime       WaitingTime     TurnAroundtime
4.00            0.00            4.00
5.00            4.00            9.00
8.00            9.00            17.00
9.00            17.00           26.00
Average waiting time:7.500000
Average turn around time:14.000000
```

Experiment - 2

1.1.4 Question:

Write a C program to simulate the following CPU scheduling algorithm to find turnaround time and waiting time.

(a) Priority (pre-emptive & Non-pre-emptive)

(b) Round Robin (Experiment with different quantum sizes for RR algorithm)

1.1.5 Code:

(a) <u>Priority (Non-pre-emptive)</u>

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

(b) <u>Round Robin (Non-pre-emptive)</u>

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

int turnarroundtime(int processes[], int n, int bt[], int wt[], int tat[]) {
   for (int i = 0; i < n ; i++)
   tat[i] = bt[i] + wt[i];
   return 1;
}

int waitingtime(int processes[], int n, int bt[], int wt[], int quantum)
{
   int rem_bt[n];
   for (int i = 0 ; i < n ; i++)
   rem_bt[i] = bt[i];
   int t = 0;

    while (1)
    {
       bool done = true;
       for (int i = 0 ; i < n; i++)
       {
          if (rem_bt[i] > 0)
```

```
            {
                done = false;
                if (rem_bt[i] > quantum)
                {

    printf("\nAverage waiting time = %f", (float)total_wt / (float)n);
    printf("\nAverage turnaround time = %f", (float)total_tat / (float)n);
    return 1;
}

int main()
{
   int n, processes[n], burst_time[n], quantum;
   printf("Enter the Number of Processes: ");
   scanf("%d",&n);

   printf("\nEnter the quantum time: ");
   scanf("%d",&quantum);

    int i=0;
    for(i=0;i<n;i++)
    {
        printf("\nEnter the process: ");
        scanf("%d",&processes[i]);
        printf("Enter the Burst Time:");
        scanf("%d",&burst_time[i]);
   }

   findavgTime(processes, n, burst_time, quantum);
   return 0;
}
```

2.2.3 Output:
(a) Priority (Non-pre-emptive)

```
ity_nonPreemptive.c -o Priority_nonPreemptive } ; if ($?) { .\Priority_nonPreemptive }
Enter the number of processes:
5
Enter the process id:
1 2 3 4 5
Enter the arrival time of the processes:
0 1 2 3 4
Enter the burst time of the processes:
5 3 6 2 4
Enter the priority of processes:
3 2 1 4 5
Pid     ArrivalTime     BurstTime       Priority        TAT     WaitingTime
5           4               4               5           5           1
4           3               2               4           8           6
1           0               5               3           5           0
2           1               3               2           13          10
3           2               6               1           18          12
Average turn around time:9.8
Average waiting time:5.8
PS C:\Users\Nisarga Gondi\OneDrive\Desktop\Nisarga\lV SEM\OS 4th sem\os lab>
```

(b) Round Robin (Non-pre-emptive)

```
Robin.c -o RoundRobin } ; if ($?) { .\RoundRobin }
Enter the Number of Processes: 3

Enter the quantum time: 2

Enter the process: 1
Enter the Burst Time:4

Enter the process: 2
Enter the Burst Time:3

Enter the process: 3
Enter the Burst Time:5


Processes           Burst Time          Waiting Time        turnaround time

        1               4                   4                   8

        2               3                   6                   9

        3               5                   7                   12

Average waiting time = 5.666667
Average turnaround time = 9.666667
```

1.2 Experiment - 3

1.2.1 Question:

Write a C program to simulate multi-level queue scheduling algorithm considering the following scenario. All the processes in the system are divided into two categories – system processes and user processes. System processes are to be given higher priority than user processes. Use FCFS scheduling for the processes in each queue.

```
        }
        else if(user_queue[u].arrival_time <= time){
            user_queue[u].waiting_time = time - user_queue[u].arrival_time;
                time += user_queue[u].burst_time;
                user_queue[u].turnaround_time = user_queue[u].waiting_time +

    return 0;
}
```

2.3.3 Output:

```
 if ($?) { gcc multilevelqueue.c -o multilevelqueue } ; if ($?) { .\multilevelqueue }
Enter the number of processes: 4
Enter arrival time, burst time, and priority (0-System/1-User) for process 1: 0 3 0
Enter arrival time, burst time, and priority (0-System/1-User) for process 2: 1 3 1
Enter arrival time, burst time, and priority (0-System/1-User) for process 3: 8 3 0
Enter arrival time, burst time, and priority (0-System/1-User) for process 4: 8 3 1
PID     Burst Time      Priority        Queue Type      Waiting Time    Turnaround Time
1       3               0               System          0               3
3       3               0               System          0               3
2       3               1               User            2               5
4       3               1               User            3               6
Average Waiting Time: 1.25
Average Turnaround Time: 4.25
```

1.3 Experiment - 4

1.3.1 Question:

Write a C program to simulate Real-Time CPU Scheduling algorithms:
(a) Rate- Monotonic
(b) Earliest-deadline First

1.3.2 Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#
        for (int j = 0; j < cycles; j++)
        {
            if (process_list[j] == i + 1)
                printf("|####");
            else
                printf("|    ");
        }
        printf("|\n");
    }
}

void rate_monotonic(int time)
{
    int
            if ((i + 1) % period[k] == 0)
            {
                remain_time[k] = execution_time[k];
                next_process = k;
            }
        }
    }
    print_schedule(process_list, time);
}

void
    }

    for(int i=0; i<num_of_process; i++){
        for(int j=i+1; j<num_of_process; j++){
            if(deadline[j] < deadline[i]){
                int temp = execution_time[j];
                execution_time[j] = execution_time[i];
                execution_time[i] = temp;
```

```
                temp = deadline[j];
                deadline[j] = deadline[i];
                deadline[i] = temp;
                temp = process[j];
                process[j] = process[i];
                process[i] = temp;
            }
        }
    }

    for(int i=0; i<num_of_process; i++){
        remain_time[i] = execution_time[i];
        remain_deadline[i] = deadline[i];
    }

    print_schedule(process_list, time);
}

int main()
{
    int option;

    }
    return 0;
}
```

1.3.3 Output:

(a) Rate Monotonic:

```
1. Rate Monotonic
2. Earliest Deadline first
3. Proportional Scheduling

Enter your choice: 1
Enter total number of processes (maximum 10): 3

Process 1:
==> Execution time: 3
==> Period: 20

Process 2:
==> Execution time: 2
==> Period: 5

Process 3:
==> Execution time: 2
==> Period: 10

Scheduling:

Time: | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
P[1]: |    |    |    |    |####|    |    |####|####|    |    |    |    |    |    |    |    |    |    |    |
P[2]: |####|####|    |    |    |####|####|    |    |    |####|####|    |    |    |####|####|    |    |    |
P[3]: |    |    |####|####|    |    |    |    |    |    |    |    |####|####|    |    |    |    |    |    |
```

(b) Earliest Deadline First:

```
1. Rate Monotonic
2. Earliest Deadline first
3. Proportional Scheduling

Enter your choice: 2
Enter total number of processes (maximum 10): 3

Process 1:
==> Execution time: 3
==> Deadline: 7

Process 2:
==> Execution time: 2
==> Deadline: 4

Process 3:
==> Execution time: 2
==> Deadline: 8

Scheduling:

Time: | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 |
P[1]: |    |    |####|####|####|    |    |    |
P[2]: |####|####|    |    |    |    |    |####|
P[3]: |    |    |    |    |    |####|####|    |
```

Experiment - 5

1.3.4 Question:

Write a C program to simulate producer-consumer problem using semaphores.

1.3.5 Code:

```
    printf(" P%d\n", ans[n - 1]);
  }
  return 0;
}
```

1.3.6 Output:

```
rs.c -o Bankers } ; if ($?) { .\Bankers }
Enter number of processes and number of resources required
5 3
Enter the max matrix for all process
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3
Enter number of allocated resources 5 for each process
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
Enter number of available resources
3 3 2
Resouces can be allocated to Process:2 and available resources are: 3 3 2
Resouces can be allocated to Process:4 and available resources are: 5 3 2
Resouces can be allocated to Process:5 and available resources are: 7 4 3
Resouces can be allocated to Process:1 and available resources are: 7 4 5
Resouces can be allocated to Process:3 and available resources are: 7 5 5

Need Matrix:
7 4 3
1 2 2
6 0 0
0 1 1
4 3 1

System is in safe mode
<P2 P4 P5 P1 P3 >
```

2.7 Experiment - 8

1.3.7 Question:

Write a C program to simulate deadlock detection.

1.3.8 Code:

```
#include<stdio.h>


        p[k]=in[i];
```

1.3.9 Output:

(a) FCFS:

```
Enter the number of Requests
8
Enter the Requests sequence
95 180 34 119 11 123 62 64
Enter initial head position
50
Total head moment is 644
```

(b) SCAN:

```
Enter the number of Requests
6
Enter the Requests sequence
90 120 30 60 50 80
Enter initial head position
70
Enter total disk size
200
Enter the head movement direction for high 1 and for low 0
0
Total head movement is 190
```

(c) C-SCAN:

```
Enter the number of Requests
3
Enter the Requests sequence
2 1 0
Enter initial head position
1
Enter total disk size
3
Enter the head movement direction for high 1 and for low 0
1
Total head movement is 4
```