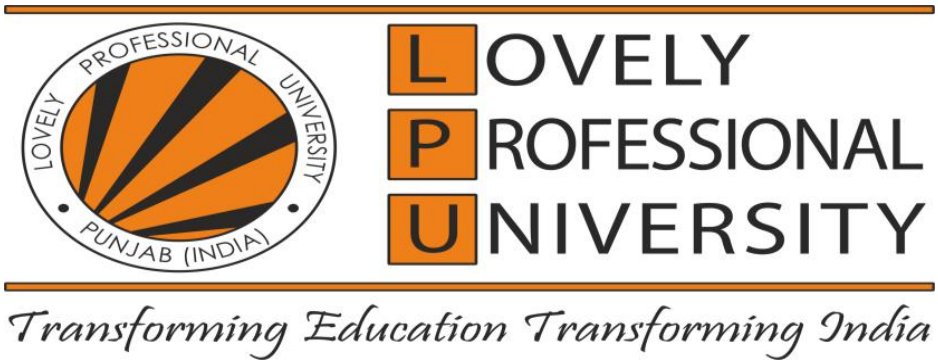
Stimulation based assignment of

OPERATING SYSTEM

Course code: CSE316

Section: K18FR



Submitted by:-

Name: Satyam Rana

Roll No.: 62

Registration No.: 11805134

Submitted to:-

Name of the faculty: Ms. Samreen

Date of submission of report: 15-April-2020

Department of Intelligent Systems
School of Computer Science Engineering
Lovely Professional University, Jalandhar
APRIL-2020

Student Name: SATYAM RANA

Student ID: 11805134
Email Address: satyamrana2599@gmail.com

GitHub Link: https://github.com/SatyamRana25/Operating-System.git

Code: C language program

```
//header files

#include<stdio.h>

#include<string.h>

//P FOR PROCESS NAME || PT FOR PROCESS TIME || WT FOR WAITING TIME

//NOG FOR NO OF GIFTS || NOP FOR NO OF PERSONS

//TOTWT FOR TOTAL WAITING TIME || AVGWT FOR AVERAGE WAITING TIME

struct process

{char p[10]; int pt,

        wt, nog;

}s[20];

//overall complexity is O(n^2)

int main()

{char temp[10];

        int i, j, totwt=0,

        temp1; int nop;

        float avgwt;

        printf("\n\t***Welcome everyone to the operating system*\n");

        printf("\t         **Process execution started**\n");

        printf("_____\n");

        printf("Enter number of persons:\t");
```

```
        scanf("%d",&nop);
//complexity is O(n)
        for(i=0;i<nop;i++)
        {
                printf("\nEnter person %d name:\t",i+1);
                scanf("%s",s[i].p);
                printf("Enter process time:\t");
                scanf("%d",&s[i].pt);
                printf("Enter number of gifts:\t");
                scanf("%d",&s[i].nog);
        }
//complexity is O(n^2)
        for(i=0;i<nop-1;i++)
        {for(j=i+1;j<nop;j++)
                {if(s[j].nog>s[i].nog)
                        {//gift order
                                temp1=s[i].nog;
                                s[i].nog=s[j].nog;
                                s[j].nog=temp1;
                                //process time
                                order
                                temp1=s[i].pt;
                                s[i].pt=s[j].pt;
                                s[j].pt=temp1;
                                //process name
```

order

```
                    strcpy(temp,s[i].p);
                    strcpy(s[i].p,s[j].p);
                    strcpy(s[j].p,temp);
        }       }       }
        s[0].wt=0;
//complexity is O(n)
        for(i=1;i<nop;i++)
        {s[i].wt=s[i-1].pt+s[i-1].wt;
            totwt=totwt+s[i].wt;
        }
//converting int to float type conversion
        avgwt=(float)totwt/nop;
        printf("_____");
        printf("\nPersons_Name\tProcess_Time\tNo_Of_Gifts\tWaiting_Ti
        me\n"); printf("_____\n");
//complexity is O(n)
        for(i=0;i<nop;i++)
        {printf(" %s\t\t%d\t\t%d\t\t%d\n",s[i].p,s[i].pt,s[i].nog,s[i].wt);
        }
        printf("\nTotal waiting time=%d\nAvg waiting
        time=%f\n",totwt,avgwt); printf("_____\n");
        printf("\t       **Process execution completed**\n");   }
```

In this problem I use C language to solve the problem of order of billing perform by the accountant for the customer.

PROBLEM IN TERMS OF OPERATING SYSTEM

Ten students (a, b, c, d, e, f, g, h, i, j) are going to attend an event. There are lots of gift shops, they all are going to the gift shops and randomly picking the gifts. After picking the gifts, they are randomly arriving in the billing counter. The accountant gives the preference to that student who has maximum number of gifts. Create a C or Java program to define order of billed student.

Priority scheduling is a method of scheduling processes based on priority. In this method, the scheduler chooses the tasks to work as per the priority, which is different from other types of scheduling, for example, a **simple round robin**.

Priority scheduling involves priority assignment to every process, and processes with higher priorities are carried out first, whereas tasks with equal priorities are carried out on a first-come-first-served (FCFS) or round robin basis. An example of a general-priority-scheduling algorithm is the shortest-job-first (SJF) algorithm.

This concept is purely based on priority scheduling with non-preemptive concept. Because according to the 10 students (a, b, c, d, e, f, g, h, i, j) which one has maximum number of gifts, that student has given highest priority by the accountant and which has minimum has given least priority by the accountant. Suppose, if student A has maximum no of gifts and B has minimum no of gifts then accountant will give first preference to student A and then at last it will give preference to student B. This all move around **priority scheduling algorithm**.

ALGORITHM:

Take all the inputs from the user.
Arrange the processes in descending order according to their number of gifts.
Calculate waiting time for all the processes.
Summarise the total waiting time and average waiting time.

COMPLEXITY OF THE ALGORITHM:

The complexity of the scheduling algorithm is O(n^2).

CONSTRAINTS

In priority-based scheduling algorithms, a major problem is indefinite block, or starvation.

A process that is ready to run but waiting for the CPU can be considered blocked.

A priority scheduling algorithm can leave some low-priority processes waiting indefinitely.

A steady stream of higher-priority processes can prevent a low-priority process from ever getting the CPU.

CODE SNIPPET:

```
for (i=0;i<nop-1;i++)
    {for (j = i+1; j < nop; j++)
        {if (s[j].nog>s[i].nog)
            {//gift order
                temp1=s[i].nog;
                s[i].nog=s[j].nog;
                s[j].nog=temp1;
                //process time
                order
                temp1=s[i].pt;
                s[i].pt=s[j].pt;
                s[j].pt=temp1;
                //process name
                order
                strcpy(temp,s[i].p);
                strcpy(s[i].p,s[j].p);
                strcpy(s[j].p,temp);
```

}   }   }

BOUNDARY CONDITION:

Once resources are allocated to a process, the process holds it till it completes its burst time or switches to waiting state.

Process cannot be interrupted until it terminates itself or its time is up.

If a process with long burst time is running CPU, then later coming process with less CPU burst time may starve.

It does not have overheads.

The process is rigid.

No cost associated.

TEST CASE