STUDENT NAME: SIDDHARTH AGRAWAL

Student ID (Reg. No.): 11615148

Roll No.: A-21

E-mail Address: siddharthagrawal.lpu@gmail.com

GitHub **Link:** https://github.com/SiddharthAgrawal2106/OS-Assignment

Warning-Please Do not open OS-project for evaluation as it was my 1st attempt and a test for GitHub on how it works so it was a dummy test.

CODE: 18,19

Ques. 18. 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 students?

1.Descriptipn: Here, we are taking students as processes. So there are process basically which may have different/same arrival time which are acting as if students are going to the accountant for billing , and accountant is acting as an CPU. and since students are buying gifts and every student is buying different/same number of gifts so they have their burst time equal to number of gifts they purchased respectively. So here we are implementing Non-premtive Longest Job first(LJF) to determine the order of the bill of the student.

2.Algorithm:

Step-1 Sorting all students data i.e. their Arrival time, Number of Gifts, and burst time according to their increasing Arrival Time.

CODE SNIPPET

```
        for(i=0;i<n-1;i++)
{
        for(j=0;j<n-1;j++)
        {
                if(student[j+1].arr_time<student[j].arr_time)
                {
                        temp1=student[j];
                        student[j]=student[j+1];
                        student[j+1]=temp1;
                }
        }
}
```

COMPLEXITY **[O(n2)]**

Step-2 Selecting Student one by one and billing them according to their Burst time and and also looking for students whose bust time is less but has more number of gifts as comared to another student which have same or greter burst time.

CODE SNIPPET

```
int timer=0;
int finished=0;

int f=0;
timer=student[f].arr_time;
while(finished<n)
{
        int max_gifts=0;
        int chosen=-1;
        for(int i=0;i<n;i++)
        {
                if(student[i].arr_time<=timer && student[i].state==0)
                {
                        if(student[i].gifts>max_gifts)
                        {
                                chosen=i;
                                max_gifts=student[i].gifts;
                        }
                }
        }
        if(chosen!=-1)
        {
                printf("%c  :",student[chosen].name);
                finished++;
                student[chosen].state=1;
                timer+=student[chosen].gifts;
                printf(": completion time: %d\n",timer);
        }
        else
        {
                timer++;
        }
}
```

COMPLEXITY **[O(n2)]**

Final Test Cases Result 1:

Enter number of students : 3

Enter arrival time for A : 1
Enter number of gifts bought by A : 2
Burst time for A is : 2

Enter arrival time for B : 3
Enter number of gifts bought by B : 4
Burst time for B is : 4

Enter arrival time for C : 5
Enter number of gifts bought by C : 6
Burst time for C is : 6

**********----------Entered Details are ----------**********

**********----------Number of Students are 3 -------*******
Student Name: A Arrival Time: 1 Gifts: 2 Burst Time: 2

Student Name: B Arrival Time: 3 Gifts: 4 Burst Time: 4

Student Name: C Arrival Time: 5 Gifts: 6 Burst Time: 6

*****-----Sorted Students according to the ARRIVAL TIME ARE------******

Student Name: A Arrival Time: 1 Gifts: 2 Burst Time: 2

Student Name: B Arrival Time: 3 Gifts: 4 Burst Time: 4

Student Name: C Arrival Time: 5 Gifts: 6 Burst Time: 6

**********FINAL ANSWER**********
A :: completion time: 3
B :: completion time: 7
C :: completion time: 13

Final Test Cases Result 2:

Enter number of students : 3

Enter arrival time for A : 1
Enter number of gifts bought by A : 4
Burst time for A is : 4

Enter arrival time for B : 6
Enter number of gifts bought by B : 3
Burst time for B is : 3

Enter arrival time for C : 8
Enter number of gifts bought by C : 6
Burst time for C is : 6

**********----------Entered Details are ----------**********

**********----------Number of Students are 3 -------*******
Student Name: A Arrival Time: 1 Gifts: 4 Burst Time: 4

Student Name: B Arrival Time: 6 Gifts: 3 Burst Time: 3

Student Name: C Arrival Time: 8 Gifts: 6 Burst Time: 6

*****-----Sorted Students according to the ARRIVAL TIME ARE------******

Student Name: A Arrival Time: 1 Gifts: 4 Burst Time: 4

Student Name: B Arrival Time: 6 Gifts: 3 Burst Time: 3

Student Name: C Arrival Time: 8 Gifts: 6 Burst Time: 6

**********FINAL ANSWER**********
A :: completion time: 5
B :: completion time: 9
C :: completion time: 15

Final Test Case Result 3:

Enter number of students : 3

Enter arrival time for A : 4
Enter number of gifts bought by A : 6
Burst time for A is : 6

Enter arrival time for B : 4
Enter number of gifts bought by B : 6
Burst time for B is : 6

Enter arrival time for C : 1
Enter number of gifts bought by C : 9
Burst time for C is : 9

**********----------Entered Details are ----------**********

**********----------Number of Students are 3 -------*******
Student Name: A Arrival Time: 4 Gifts: 6 Burst Time: 6

Student Name: B Arrival Time: 4 Gifts: 6 Burst Time: 6

Student Name: C Arrival Time: 1 Gifts: 9 Burst Time: 9

****

*****-----Sorted Students according to the ARRIVAL TIME ARE------******

Student Name: C Arrival Time: 1 Gifts: 9 Burst Time: 9

Student Name: A Arrival Time: 4 Gifts: 6 Burst Time: 6

Student Name: B Arrival Time: 4 Gifts: 6 Burst Time: 6

****
C :: completion time: 10
A :: completion time: 16
B :: completion time: 22

Final Test Case Result 4:

Enter number of students : 3

Enter arrival time for A : 1
Enter number of gifts bought by A : 9
Burst time for A is : 9

Enter arrival time for B : 4
Enter number of gifts bought by B : 6
Burst time for B is : 6

Enter arrival time for C : 4
Enter number of gifts bought by C : 8
Burst time for C is : 8

**********----------Entered Details are ----------**********

**********----------Number of Students are 3 -------*******
Student Name: A Arrival Time: 1 Gifts: 9 Burst Time: 9

Student Name: B Arrival Time: 4 Gifts: 6 Burst Time: 6

Student Name: C Arrival Time: 4 Gifts: 8 Burst Time: 8

****

*****-----Sorted Students according to the ARRIVAL TIME ARE------******

Student Name: A Arrival Time: 1 Gifts: 9 Burst Time: 9

Student Name: B Arrival Time: 4 Gifts: 6 Burst Time: 6

Student Name: C Arrival Time: 4 Gifts: 8 Burst Time: 8

****
A :: completion time: 10
C :: completion time: 18
B :: completion time: 24

Ques. 19. There are 5 processes and 3 resource types, resource A with 10 instances, B with 5 instances and C with 7 instances. Consider following and write a c code to find whether the system is in safe state or not?

Available			Processes	Allocation			Max		
A	B	C		A	B	C	A	B	C
3	3	2	P0	0	1	0	7	5	3
			P1	2	0	0	3	2	2
			P2	3	0	2	9	0	2
			P3	2	1	1	2	2	2
			P4	0	0	2	4	3	3

1.Description:

To illustrate the use of the banker's algorithm, consider a system with five processes *P*0 through *P*4 and three resource types *A, B,* and *C.* Resource type *A* has ten instances, resource type *B* has five instances, and resource type *C* has seven instances. Suppose that, at time *T*0, the following snapshot of the system has been taken as shown above

The content of the matrix ***Need*** is defined to be ***Max*** − ***Allocation*** and is as follows:

Need

A B C

*P*0 7 4 3

*P*1 1 2 2

*P*2 6 0 0

*P*3 0 1 1

*P*4 4 3 1

We claim that the system is currently in a safe state. Indeed, the sequence <*P*1, *P*3, *P*4, *P*2, *P*0> satisfies the safety criteria. Suppose now that process *P*1 requests one additional instance of resource type *A* and two instances of resource type *C*, so *Request*1 = (1,0,2). To decide whether this request can be immediately granted, we first check that *Request*1 ≤ ***Available***—that is, that (1,0,2) ≤ (3,3,2), which is true. We then pretend that this request has been fulfilled, and we arrive at the following new state:

	Allocation	*Need*	*Available*
	A B C	*A B C*	*A B C*
*P*0	0 1 0	7 4 3	2 3 0
*P*1	3 0 2	0 2 0	
*P*2	3 0 2	6 0 0	
*P*3	2 1 1	0 1 1	
*P*4	0 0 2	4 3 1	

We must determine whether this new system state is safe. To do so, we execute our safety algorithm and find that the sequence <*P*1, *P*3, *P*4, *P*0, *P*2> satisfies the safety requirement. Hence, we can immediately grant the request of process *P*1.
So it is in safe state

2.ALGORITHM:

Step1: Declaring all process state as zero ,meaning that no process is completed in the initial and making it one if process completed.

```
int allo[no_of_proc][no_of_reso];
int max[no_of_proc][no_of_reso];
int avail[no_of_reso];



int count1=0;
int count2=0;
int proc_state[no_of_proc];

for(i=0;i<no_of_proc;i++)
{
        proc_state[i]=0;
}
```

STEP 2: Applying the main logic of **BANKERS ALGORITHM.**

```
int x=0;
for(x=0;x<no_of_proc;x++)
{
        for(i=0;i<no_of_proc;i++)
        {
                count1=0;
                count2=0;
                if(proc_state[i]==0)
                {
                        for(j=0;j<no_of_reso;j++)
                                {
                                        if(max[i][j]-allo[i][j]<=0)
                                        {
                                                count1=count1+1;
                                        }

                                        if((avail[j]+allo[i][j])>=max[i][j])
                                        {
                                                count2=count2+1;
                                        }
                                }
                        if(count1==no_of_reso||count2==no_of_reso)
                        {
                                proc_state[i]=1;
                                printf("\n**\tProcess P%d COMPLETED  **",i);
                                int y=0;
                                for(y=0;y<no_of_reso;y++)
                                {
                                        avail[y]=avail[y]+allo[i][y];
                                }

                                printf("\n\n PRESENT AVAILABLE RESOURCES:  ");
                                int z=0;
                                for(z=0;z<no_of_reso;z++)
                                {
                                        printf("\t%d",avail[z]);
                                }
                                printf("\n");

                        }
                }
        }
}
```

COMPLEXITY: O[(n^3)]

Test Case Result 1:

PROBLEM 19::

-----Complete Resource Allocation Graph------
Available
3 3 2

Allocation
Process P0: 0 Process P0: 1 Process P0: 0
Process P1: 2 Process P1: 0 Process P1: 0
Process P2: 3 Process P2: 0 Process P2: 2
Process P3: 2 Process P3: 1 Process P3: 1
Process P4: 0 Process P4: 0 Process P4: 2

Max
Process P0: 7 Process P0: 5 Process P0: 3
Process P1: 3 Process P1: 2 Process P1: 2
Process P2: 9 Process P2: 0 Process P2: 2
Process P3: 2 Process P3: 2 Process P3: 2
Process P4: 4 Process P4: 3 Process P4: 3

** Process P1 COMPLETED **

PRESENT AVAILABLE RESOURCES: 5 3 2

** Process P3 COMPLETED **

PRESENT AVAILABLE RESOURCES: 7 4 3

** Process P4 COMPLETED **

PRESENT AVAILABLE RESOURCES: 7 4 5

** Process P0 COMPLETED **

PRESENT AVAILABLE RESOURCES: 7 5 5

** Process P2 COMPLETED **

PRESENT AVAILABLE RESOURCES: 10 5 7

FINAL AVAILABLE RESOURCES: 10 5 7

***** SAFE STATE *****

Test Case Result 2: SOLUTION TO PROBLEM 22 which is exactly the same as my problem and since my algorithm is dynamic so very easily implemented.

Available
1 5 2 0

Allocation
Process P0: 0 Process P0: 0 Process P0: 1 Process P0: 2
Process P1: 1 Process P1: 0 Process P1: 0 Process P1: 0
Process P2: 1 Process P2: 3 Process P2: 5 Process P2: 4
Process P3: 0 Process P3: 6 Process P3: 3 Process P3: 2
Process P4: 0 Process P4: 0 Process P4: 1 Process P4: 4

Max
Process P0: 0 Process P0: 0 Process P0: 1 Process P0: 2
Process P1: 1 Process P1: 7 Process P1: 5 Process P1: 0
Process P2: 2 Process P2: 3 Process P2: 5 Process P2: 6
Process P3: 0 Process P3: 6 Process P3: 5 Process P3: 2
Process P4: 0 Process P4: 6 Process P4: 5 Process P4: 6

** Process P0 COMPLETED **

PRESENT AVAILABLE RESOURCES: 1 5 3 2

** Process P2 COMPLETED **

PRESENT AVAILABLE RESOURCES: 2 8 8 6

** Process P3 COMPLETED **

PRESENT AVAILABLE RESOURCES: 2 14 11 8

** Process P4 COMPLETED **

PRESENT AVAILABLE RESOURCES: 2 14 12 12

** Process P1 COMPLETED **

PRESENT AVAILABLE RESOURCES: 3 14 12 12

FINAL AVAILABLE RESOURCES: 3 14 12 12

***** SAFE STATE *****

Test Case Result 3: Showing UN-safe State.

Available
3 3 2

Allocation
Process P0: 0 Process P0: 1 Process P0: 0
Process P1: 2 Process P1: 0 Process P1: 0
Process P2: 3 Process P2: 0 Process P2: 2
Process P3: 2 Process P3: 1 Process P3: 1
Process P4: 0 Process P4: 0 Process P4: 2

Max
Process P0: 7 Process P0: 5 Process P0: 3
Process P1: 3 Process P1: 2 Process P1: 2
Process P2: 11 Process P2: 0 Process P2: 2
Process P3: 2 Process P3: 2 Process P3: 2
Process P4: 4 Process P4: 3 Process P4: 3

** Process P1 COMPLETED **

PRESENT AVAILABLE RESOURCES: 5 3 2

** Process P3 COMPLETED **

PRESENT AVAILABLE RESOURCES: 7 4 3

** Process P4 COMPLETED **

PRESENT AVAILABLE RESOURCES: 7 4 5

** Process P0 COMPLETED **

PRESENT AVAILABLE RESOURCES: 7 5 5

**** NOT SAFE STATE *****

**********THE END OF THE ASSIGNMENT**********