

OPERATING SYSTEM

Simulation Based Assignment

NAME : **SRINJAY DAM**
ID : **11605431**
EMAIL : **srinjaydam6@gmail.com**
SECTION : **K1616**
QUESTIONS: **1&22**

QUESTION 1:

1) Explain the problem in terms of operating system concept?

ANS:

In question 1 , there can be any number of processes (we have taken 4 by default) , With different arrival and burst times taken from the user at runtime. I have performed 2 iterations naming Initial and Final iterations used in the code . At first the scheduler schedules the process by interrupting the process at every 3 units of time , considering the completion of the process in the processor. The scheduler then checks for any number of processes remaining in the ready queue and assigns the processor to the process by interrupting the processor after every 6 units of time , also considering the completion of the process at every iteration.

1. In initial iteration I have take round robin scheduling with 3 time units.
2. In final iteration I have taken round robin scheduling with 6 time units.

2)Write the algorithm for the proposed solution of the assigned problem?

Ans

1.We have taken a structure in which we have 2 components arrival time and burst time for (n) number of processes.

2. In main function I have taken the total time consumption variable as 't' and initialized it to '0'. Then taking input from the user on number of processes and the arrival time and burst time of each process we move on to initial iteration.

3. In initial iteration a variable of type Boolean is taken to check for completion of that particular process .

4. After checking for the completion of the process we check if the burst time of that process is greater than 3 which is the time slice of that iteration , and keep on incrementing the value of 't' by 3 after each execution of the process to indicate the total time the process has taken up the processor for its execution.

Else we add burst time to the existing time frame and decrement the value of burst time by 't' to calculate the waiting time.

After the process has completed its complete execution we set the burst time of that process to '0';

5. Repeat 3 & 4 but only change is we take 6 units of time rather than 3 units of time.

3) Calculate the complexity of implemented algorithm?

Ans

Overall complexity of the program is $O(n)$

1) In Initial iteration we have one for loop ,therefore complexity $O(n)$.

2) In Final iteration we have one for loop , therefore complexity $O(n)$.

Overall complexity $n+n=O(2n)$, therefore $O(n)$.

4) Explain all the constraints given in the problem .Attach the code snippet of the implemented constraints.

INITIAL ITERATION :

Jusers\Srinjay\Desktop\Untitled1.cpp - Dev-C++ 5.11

Edit Search View Project Execute Tools AStyle Window Help



t Classes Debug

[*] Untitled1.cpp

```
34 printf("Initial iteration\n");
35 for(i=0;i<n;i++)
36 {
37     bool completed;
38
39     if(p[i].arrivaltime<=t){
40         if (p[i].bursttime>0)
41         {
42             completed = false;
43             if (p[i].bursttime > 3)
44             {
45
46                 t += 3;
47
48                 p[i].bursttime -= 3;
49             }
50
51             else
52             {
53
54                 t = t + p[i].bursttime;
55
56                 wait[i] = t - p[i].bursttime;
57
58                 p[i].bursttime = 0;
59             }
60
61         }
62     }
63
64     if (completed == true)
65         break;
66 }
67
68 }
```

FINAL ITERATION :

```

69 printf("Final Iteration.....");
70 for(i=0;i<n;i++)
71 {
72     bool completed1;
73     if(p[i].arrivaltime<=t)
74     {
75         if (p[i].bursttime>0)
76         {
77             completed1 = false; // completed = false indicates that there is an incomplete process
78             if (p[i].bursttime > 6)
79             {
80                 t += 6; //t shows how much time a process has executed
81                 p[i].bursttime -= 6; //decrease the burst time by 6
82             }
83             else
84             {
85                 t = t + p[i].bursttime; //t shows how much time a process has executed
86                 wait[i] = t - p[i].bursttime; //wait time is current time minus burst time
87                 p[i].bursttime = 0;
88             }
89         }
90     }
91     if (completed1 == true) //when all processes are done
92     break;
93 }
94 printf("All processes are done in second iteration\n\n");
95 }

```

5)If you have implemented any additional algorithm to support the solution , explain the need and usage of the same.

Ans

I have not performed any extra algorithm .

6)Explain the boundary conditions of the implemented code?

Ans

1)The no of process entered by the user should not be greater than 5.

7)Explain all the test cases applied on the solution of assigned problem?

Ans

Test case:1

Process	Arrival time	Burst time
P1	0	12

P2	6	17
P3	8	8
P4	15	9

Test case:2

Process	Arrival time	Burst time
P1	2	8
P2	5	12
P3	7	16
P4	15	23

```

1  #include<stdio.h>
2  #include<conio.h>
3
4  struct process
5  {
6      int arrivaltime;
7      int bursttime;
8  }p[5];
9
10 void main()
11 {
12     int t = 0; int wait[5];
13     printf("Enter the number of processes you want to execute\n");
14     int n;
15     scanf("%d\n",&n);
16     printf("Enter the details of %d processes",n);
17     int i;
18     for(i=0;i<n;i++)
19     {
20         printf("Enter the arrival time of process P%d\n",i);
21         scanf("%d\n",&p[i].arrivaltime);
22         printf("Enter the burst time of process P%d\n",i);
23         scanf("%d\n",&p[i].bursttime);
24     }
25     printf("-----");
26     for(i=0;i<n;i++)
27     {
28         printf("\n\t\tProcess\t\tArrival Time\t\tBurst Time\t\t\n\n");
29         printf("\n\t\tP%d\t\t%d\t\t%d\t\t\n",i,p[i].arrivaltime,p[i].bursttime);
30     }
31 }
32
33

```

Jusers\Srinjay\Desktop\Untitled1.cpp - Dev-C++ 5.11

Edit Search View Project Execute Tools AStyle Window Help



t Classes Debug [*] Untitled1.cpp

```
34 printf("Initial iteration\n");
35 for(i=0;i<n;i++)
36 {
37     bool completed;
38
39     if(p[i].arrivaltime<=t){
40         if (p[i].bursttime>0)
41         {
42             completed = false;
43             if (p[i].bursttime > 3)
44             {
45
46                 t += 3;
47
48                 p[i].bursttime -= 3;
49             }
50
51             else
52             {
53
54                 t = t + p[i].bursttime;
55
56                 wait[i] = t - p[i].bursttime;
57
58                 p[i].bursttime = 0;
59             }
60         }
61     }
62
63     if (completed == true)
64         break;
65 }
66
67
68 }
```

```

69 printf("Final Iteration.....");
70 for(i=0;i<n;i++)
71 {
72     bool completed1;
73     if(p[i].arrivalttime<=t)
74     {
75         if (p[i].bursttime>0)
76         {
77             completed1 = false; // completed = false indicates that there is an incomplete process
78             if (p[i].bursttime > 6)
79             {
80                 t += 6; //t shows how much time a process has executed
81                 p[i].bursttime -= 6; //decrease the burst time by 6
82             }
83             else
84             {
85                 t = t + p[i].bursttime; //t shows how much time a process has executed
86                 wait[i] = t - p[i].bursttime; //wait time is current time minus burst time
87                 p[i].bursttime = 0;
88             }
89         }
90         if (completed1 == true) //when all processes are done
91             break;
92     }
93 }
94 printf("All processes are done in second iteration\n\n");
95 }

```

QUESTION 22:

1) Explain the problem in terms of operating system concept?

ANS:

In question 22 where we are given a table where the respective columns are AVAILABLE , PROCESSES , ALLOCATION , MAX . We are supposed to find out the safe sequence for the system. This scenario looks like Bankers Algorithm. The banker's algorithm is a resource allocation and deadlock avoidance algorithm that tests for safety by simulating the allocation for predetermined maximum possible amounts of all resources, then makes a "safe state" check to test for possible activities, before deciding whether allocation should be allowed to continue.

2)Write the algorithm for the proposed solution of the assigned problem?

Ans

1)In main function we declare all the variables associated with the algorithm.

We ask the user to enter the number of processes , resources , number of instances available for each resource and maximum demand for each process which fills up the table of the question.

2)We calculate the need by subtracting maximum matrix and allocate matrix.

3) We check if available process meets the necessary conditions which is it should be greater than need. If the condition is not satisfied then break the process else continue.

We further check the iteration with the available resource, if both the iteration and available resource are equal then further execution can be done.

By running another loop we calculate the remaining available by adding the allocation matrix. We further update of the safe sequence matrix after every process execution.

4) After execution of the processes safe sequence is updated and then it is further checked whether the system is really in safe state or not.

3) Calculate the complexity of implemented algorithm?

Ans

1) Complexity for accepting values from user is $O(n^2)$.

2) Complexity for calculating NEED section is $O(n^2)$

3) Complexity of the available and allocation and safe sequence section is $O(n^3)$

Overall Complexity of the program is $O(n^3)$

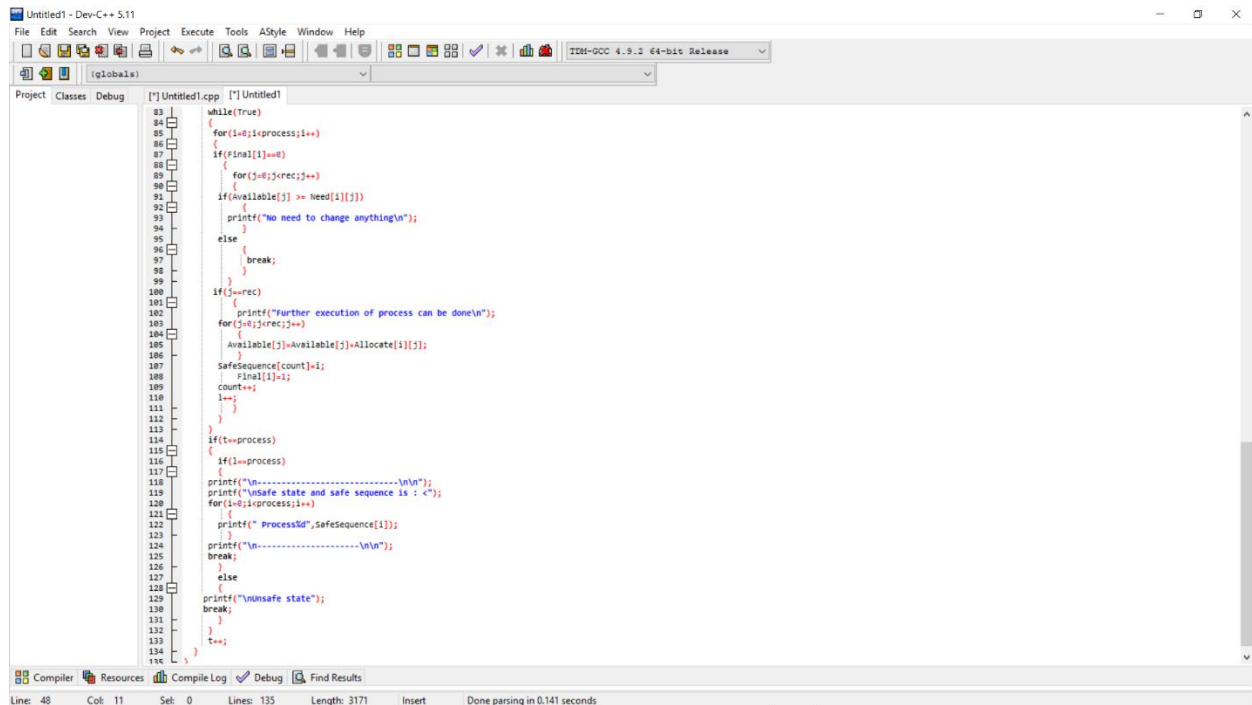
4) Explain all the constraints given in the problem. Attach the code snippet of the implemented constraints.

Calculation of Need column by subtracting Maximum need and allocate of the particular process :

```
66 |
67 | □
68 |
69 |
70 |
71 | □
72 |
73 |
74 | □
75 |
76 |
77 |
```

```
for(i=0;i<rec;i++)
{
    printf(" Resource%d",i);
}
for(i=0;i<process;i++)
{
    printf("\nProcess%d ",i);
    for(j=0;j<rec;j++)
    {
        Need[i][j]=Maximum[i][j]-Allocate[i][j];
        printf(" %d",Need[i][j]);
    }
}
```

Calculation of Available and Safe sequence state :



```
83 while(true)
84 {
85     for(i=0;i<process;i++)
86     {
87         if(final[i]==0)
88         {
89             for(j=0;j<rec;j++)
90             {
91                 if(Available[j] >= need[i][j])
92                 {
93                     printf("No need to change anything\n");
94                 }
95                 else
96                 {
97                     break;
98                 }
99             }
100             if(j==rec)
101             {
102                 printf("Further execution of process can be done\n");
103                 for(j=0;j<rec;j++)
104                 {
105                     Available[j]=Available[j]+Allocate[i][j];
106                 }
107                 SafeSequence[count]=i;
108                 final[i]=1;
109                 count++;
110                 i++;
111             }
112             }
113             if(t==process)
114             {
115                 if(i==process)
116                 {
117                     printf("\n-----\n\n");
118                     printf("Unsafe state and safe sequence is : <");
119                     for(i=0;i<process;i++)
120                     {
121                         printf(" Process%d",SafeSequence[i]);
122                     }
123                     printf("\n-----\n\n");
124                     break;
125                 }
126                 else
127                 {
128                     printf("Unsafe state");
129                     break;
130                 }
131             }
132             t++;
133         }
134     }
135 }
```

5)If you have implemented any additional algorithm to support the solution , explain the need and usage of the same.

Ans

I have not used any extra algorithm for this question.

6)Explain the boundary conditions of the implemented code?

Ans

1)Program will crash when the array size is exceeded when the user makes input in the program.

7)Explain all the test cases applied on the solution of assigned problem?

Ans

Test case:1

Process	Allocation			Max			Available		
	A	B	C	A	B	C	A	B	C
P ₀	0	1	0	7	5	3	3	3	2
P ₁	2	0	0	3	2	2			
P ₂	3	0	2	9	0	2			
P ₃	2	1	1	2	2	2			
P ₄	0	0	2	4	3	3			

Test case:2

Process	Allocated			Maximum			Available			Need(Maximum-allocated)		
	A	B	C	A	B	C	A	B	C	A	B	C
P1	0	1	0	7	5	3	5	3	2	7	4	3
P3	4	0	1	9	0	4				5	0	3
P4	2	1	1	2	2	2				0	1	1

CODE :

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

```
#include<conio.h>
```

```
int main()
```

```
{
```

```
    int
```

```
    rec,process,Allocate[30][30],Need[30][30],SafeSequence[30],Final[30],
    Available[30];
```

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

```

scanf("%d",&process);

printf("Enter the number of resources : ");

scanf("%d",&rec);

printf("-----\n");

printf("\nEnter the number of instances available for each
resource:\n");

int i;

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

{

    printf("Resource %d :",i);

    scanf("%d",&Available[i]);

}

printf("\nAvailable <");

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

{

    printf(" %d",Available[i]);

}

printf("\n-----\n\n");

printf("Enter maximum demand of each process(Max) and resource
currently allocated to it :\n");

```

```

int Maximum[30][30];

int j;

for(i=0;i<process;i++)
{
    printf("\nFor process P%d :\n",i);
    Final[i]=0;
    for(j=0;j<rec;j++)
    {
        printf("\tCurrently R%d allocated : ",j);
        scanf("%d",&Allocate[i][j]);
        printf("\tMaximum R%d required : ",j);
        scanf("%d",&Maximum[i][j]);
    }
}

printf("Maximum is :\n  ");

for(i=0;i<rec;i++)
{
    printf(" Resource%d",i);
}

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

```

```

{
    printf("Process%d ",i);
    for(j=0;j<rec;j++)
    {
        printf(" %d",Maximum[i][j]);
    }
}
printf("\nAllocated slot is :\n  ");
for(i=0;i<rec;i++)
{
    printf("Resource%d",i);
}
for(i=0;i<process;i++)
{
    printf("\nProcess%d ",i);
    for(j=0;j<rec;j++)
    {
        printf(" %d",Allocate[i][j]);
    }
}

```

```

printf("\nNeed :\n  ");
for(i=0;i<rec;i++)
{
    printf(" Resource%d",i);
}
for(i=0;i<process;i++)
{
    printf("\nProcess%d ",i);
    for(j=0;j<rec;j++)
    {
        Need[i][j]=Maximum[i][j]-Allocate[i][j];
        printf("  %d",Need[i][j]);
    }
}
printf("\n-----\n\n");
int t=0;
int count=0;
int l=0;
while(True)
{

```

```
for(i=0;i<process;i++)
{
    if(Final[i]==0)
    {
        for(j=0;j<rec;j++)
        {
            if(Available[i][j] >= Need[i][j])
            {
                printf("No need to change anything\n");
            }
            else
            {
                break;
            }
        }
        if(j==rec)
        {
            printf("Further execution of process can be done\n");
            for(j=0;j<rec;j++)
            {
```



```

        Available[j]=Available[j]+Allocate[i][j];

    }

    SafeSequence[count]=i;

    Final[i]=1;

    count++;

    l++;

    }

}

}

if(t==process)
{
    if(l==process)
    {
        printf("\n-----\n\n");
        printf("\nSafe state and safe sequence is : <");
        for(i=0;i<process;i++)
        {
            printf(" Process%d",SafeSequence[i]);
        }

        printf("\n-----\n\n");
    }
}

```

```
        break;
    }
    else
    {
        printf("\nUnsafe state");
        break;
    }
}
t++;
}
}
```