**LOVELY PROFESSIONAL UNIVERSITY**

****

**PROJECT FOR CSE 316**

**Submitted By -**

**Sobhan Bhuyan**

**Registration No.: - 11807058**

**Section: - K18GT**

**Roll – 60**

**Project Link :** <https://github.com/sobhan-bhuyan/CA-3-Assignment-OS>

Question 8) The following processes are being scheduled using a pre-emptive, round robin scheduling algorithm. Each process is assigned a numerical priority, with a higher number indicating a higher relative priority. In addition to the processes listed below, the system also has an idle task (which consumes no CPU resources and is identified as P idle). This task has priority 0 and is scheduled whenever the system has no other available processes to run. The length of a time quantum is 10 units. If a process is pre-empted by a higher-priority process, the pre-empted process is placed at the end of the queue.

| **Thread** | **Priority** | **Burst** | **Arrival** |
| --- | --- | --- | --- |
| P1 | 40 | 20 | 0 |
| P2 | 30 | 25 | 25 |
| P3 | 30 | 25 | 30 |
| P4 | 35 | 15 | 60 |
| P5 | 5 | 10 | 100 |
| P6 | 10 | 10 | 105 |

Write a C code to

a. Show the scheduling order of the processes using a Gantt chart.

b. What is the turnaround time for each process?

c. What is the waiting time for each process?

**Pre-emptive Scheduling: -**  is a CPU scheduling technique that works by dividing time slots of CPU to a given process. The time slot given might be able to complete the whole process or might not be able to it. When the burst time of the process is greater than CPU cycle, it is placed back into the ready queue and will execute in the next chance. This scheduling is used when the process switch to ready state.

**Round Robin Scheduling: -**  is a scheduling algorithm used by the system to schedule CPU utilization. This is a pre-emptive algorithm. There exists a fixed time slice associated with each request called the quantum. The job scheduler saves the progress of the job that is being executed currently and moves to the next job present in the queue when a particular process is executed for a given time quantum.

C code for the above problem: -

|  |
| --- |
|  |
| 1. #include<stdio.h> 2. #include<conio.h> 3. **int** main() 4. { 5. **int** x,n,p[10],pp[10],bt[10],w[10],t[10],awt,atat,i,at[10],tq;         //pp\_> process  ;  bt-> burst time  ;  at-> arrival time 7. printf("Enter total no. of processes   :   "); 8. scanf("%d",&n); 9. printf("Enter time quantum variable as mentioned in question    :  "); 10. scanf("%d",&tq); 11. printf("\n\t Enter Priority : Burst : Arrival  \n"); 13. **for**(i=0;i<n;i++) 14. { 16. printf("\n Process %d   :  ",i+1); 17. scanf("%d %d %d",&pp[i],&bt[i],&at[i]); 18. p[i]=i+1; 19. } 21. **int** j; 22. **for**(i=0;i<n-1;i++) 23. { 24. **for**(j=i+1;j<n;j++) 25. { 26. **if**(pp[i]<pp[j]) 27. { 28. x=pp[i]; 29. pp[i]=pp[j]; 30. pp[j]=x; 31. x=bt[i]; 32. bt[i]=bt[j]; 33. bt[j]=x; 34. x=p[i]; 35. p[i]=p[j]; 36. p[j]=x; 37. } 39. } 40. } 41. w[0]=0; 42. awt=0; 43. t[0]=bt[0]; 44. atat=t[0]; 45. **for**(i=1;i<n;i++) 46. { 47. **if**(tq<n || tq>n) { 49. w[i]=t[i-1]; 50. awt+=w[i]; 51. t[i]=w[i]+bt[i]; 52. atat+=t[i]; 53. } 54. } 56. printf("\n\n\n\n\n\n"); 57. printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*      Answers    \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"); 58. printf("\n\nA )  Gantt chart for the processes\n"); 59. **for**(i=0;i<n;i++) 60. { 61. printf("P %d ",p[i]); 62. }  65. printf("\n\nProcess \t Burst Time \t Wait Time \t Turn Around Time   Priority \tArrival time \n"); 66. **for**(i=0;i<n;i++){ 67. printf("\n  %d",p[i]); printf("\t\t %d",bt[i]); printf("\t\t %d",w[i]); printf("\t\t %d",t[i]); printf("\t\t %d",pp[i]); printf("\t\t %d",at[i]); 69. } 70. awt/=n; 71. atat/=n; 72. printf("\nB  )   Average Wait Time : %d ",awt); 73. printf("\nC  )   Average Turn Around Time : %d",atat); 74. getch(); |
| **ALGORITHM:-**  Complexity of the following code snippet is **O(n2)**   1. **for**(i=0;i<n-1;i++) 2. { 3. **for**(j=i+1;j<n;j++) 4. { 5. **if**(pp[i]<pp[j]) 6. { 7. x=pp[i]; 8. pp[i]=pp[j]; 9. pp[j]=x; 10. x=bt[i]; 11. bt[i]=bt[j]; 12. bt[j]=x; 13. x=p[i]; 14. p[i]=p[j]; 15. p[j]=x; 16. }   Complexity of the following code snippet is **O(n)** |
| 1. **for**(i=1;i<n;i++) 2. { 3. **if**(tq<n || tq>n) { 5. w[i]=t[i-1]; 6. awt+=w[i]; 7. t[i]=w[i]+bt[i]; 8. atat+=t[i]; 9. } 10. }   Hence, Complexity of the total program is **O(n2)** |
|  |
|  |
| **DESCRIPTION: -**  First the total number of processes are entered that is 6 as mentioned in the question  Then the time quantum is entered which is 10  Then the priority of the problems along with the burst time and arrival time is entered  With enter button pressed on every input.  Data is taken from the table provided in the question.  **Boundary conditions**: - are that if a very high number is entered which is greater than the max capacity of int data type then the program can’t process that. A solution to such a problem would be to take long int or float as a data type to increase the range.  Variables used in the program  p[ ] - processes  pp[ - process priority  bt [] - burst time  w[ ] - wait time  at[] - time of arrival  awt - average wait time  atat - average turnaround time  tq - time quantum  **Output: -** |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
| Question 19) Ten students (s1, s2, s3, s4, s5, s6, s7, s8, s9, s10) 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 program to define order of billed students? |  |
|  |  |
|  |  |
|  |  |
| C program for the above problem: -   1. #include <stdio.h> 3. **int** i,t,j,num\_students; //n = number of students; t=temp variable for swapping 4. **int** num\_gifts[10],students[10],bill[10]; 5. **float** cost;  8. **void** get() 9. { 11. printf("\n\n"); 12. printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Data Entered by User \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n"); 13. printf( "\nSTUDENT NO.\t\tNO.OF GIFTS PURCHASED \n"); 15. **for** (i = 0; i < num\_students; i++) 16. { 17. printf("STUDENT NO. = %d \t\t" ,students[i]); 18. printf("%d \t\t\t",num\_gifts[i]); 19. printf("\n\n"); 20. } 21. }  24. **void** billfn() 25. { 26. printf( "STUDENT\_NO\t\tTOTAL GIFTS PURCHASED\t\tBILL\n\n"); 28. **for** (i = 0; i < num\_students; i++) 29. { 30. printf("STUDENT NO. = %d \t\t" ,students[i]); 31. printf("%d \t\t\t",num\_gifts[i]); 32. bill[i]= cost \* num\_gifts[i]; 33. printf("%d \t\t\t",bill[i]); 34. printf("\n"); 35. } 36. }  39. **void** sorting() 40. { 42. printf("\*\*\*\*\*\*\*\*\*\*  Order for Billing Students on the basis of number of gifts  \*\*\*\*\*\*\*\*\*\*\*\*\* \n\n"); 43. **for**(i=0;i<num\_students;i++) 44. { 45. **for**(j=i+1;j<num\_students;++j) 46. { 47. **if**(num\_gifts[i]<num\_gifts[j]) 48. { 49. t=num\_gifts[i]; 50. num\_gifts[i]=num\_gifts[j]; 51. num\_gifts[j]=t; 52. t=students[i]; 53. students[i]=students[j]; 54. students[j]=t; 55. } 56. } 57. }          }  60. **int** main() 61. { 63. printf("Enter the number of students who have purchased gifts from shops        :  "); 64. scanf("%d",&num\_students); 65. printf("Cost of each gift       :  "); 66. scanf("%f",&cost); 67. printf("\n\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Enter the Number of Gifts Corresponding Students \*\*\*\*\*\*\*\*\*\*\*\*\*"); 68. **for**(i=0;i<num\_students;i++) 69. { 70. printf("\nStudent\_Number[%d] \n",i+1); 71. printf("Enter the number of gifts purchased        :  "); 72. scanf("%d",&num\_gifts[i]); 73. students[i]=i+1; 74. }  77. get(); 78. sorting(); 79. billfn(); 81. } |  |
|  |  |
| **Algorithm: -**  There are mainly 3 functions in this code   1. **Void get()**   This takes in the data that has to entered by the user that is student id and the number of gifts purchased by each student.   1. **void billfn()**   this function calculates the total bill and the order of billing of the students based on the max number of gifts and gives us the total bill of each student. |  |
| 1. **Void sorting()**   This function sorts the student ids on the basis of number of gifts, i.e. the more number of gifts the better is the priority. Students with highest number of gifts is billed first.  Bubble sort is used in this case an easy method to understand the code.  Different sorting algorithms can be used to reduce time complexity in future builds as time taken by the program is not a major concern now. |  |
|  |  |
|  |  |
|  |  |
| Time complexity of the following code snippet is **O(n2)** |  |
|  |  |
| 1. **for**(i=0;i<num\_students;i++) 2. { 3. **for**(j=i+1;j<num\_students;++j) 4. { 5. **if**(num\_gifts[i]<num\_gifts[j]) 6. { 7. t=num\_gifts[i]; 8. num\_gifts[i]=num\_gifts[j]; 9. num\_gifts[j]=t; 10. t=students[i]; 11. students[i]=students[j]; 12. students[j]=t; 13. } 14. }        } |  |
|  |  |
|  |  |
| Hence, the total time complexity of the entire program is **O(n2)** |  |
|  |  |
|  |  |
| **Output: -** |  |
|  |  |
|  |  |
| **C:\Users\sobha\AppData\Local\Microsoft\Windows\INetCache\Content.Word\q192.png** |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |