**SIMULATION BASED ASSIGNMENT ASSESMENT RUBRIC**

**BACHELOR OF TECHNOLOGY**

**in**

**COMPUTER SCIENCE AND ENGINEERING**

**By**

**NAME OF THE STUDENT: P. VIKAS REDDY**

**Mail ID: peddireddysai23@gmail.com**

**Registration number: 11603494**

**Roll No: A-28**

**Section: K1626**



**School of Computer Science and Engineering**

Lovely Professional University

Phagwara, Punjab (India)

**DESCRIPTION:**

A preemptive scheduling program to implements a queue with two levels. Level 1 consists of Fixed priority preemptive scheduling and Level 2 consists of Round robin scheduling. For a Fixed priority preemptive Scheduling (Queue 1), the Priority 0 is highest priority. If one process P1 is scheduled and running, another process P2 with higher priority comes. The New process (high priority) process P2 preempts currently running process P1 and process P1 will go to second level queue. Time for which process will strictly execute must be considered in the multiples of 2. All the processes in second level queue will complete their execution according to round robin scheduling. Consider Queue 2 will be processed after Queue 1 becomes empty and Priority of Queue 2 has lower priority than in Queue 1.

**ALGORITHM:**

A process can move between various queues. Processes can be any algorithm. Multilevel queue scheduling divides ready queue into several queues. Processes are permanently assigned to one queue on some property like memory size, process priority, process type. Each queue has its own scheduling algorithm. As different type of processes are there so all can’t be put into same queue and apply same scheduling algorithm.

In this assignment there is use of two algorithms priority scheduling and round robin scheduling.

**Preemptive Priority Scheduling:**

* Priority is associated with each process.
* CPU is allocated to the process with highest priority.
* If 2 processes have same priority à FCFS.

**Round robin Scheduling:**

* It is simple, easy to implement, and starvation-free as all processes get fair share of CPU.
* A Time Quantum is associated to all processes.
* Time Quantum: Maximum amount of time for which process can run once it is scheduled.
* RR scheduling is always Pre-emptive.

**PURPOSE OF USE:**

The purpose is to schedule the processes in different queues with different algorithms like priority scheduling in one queue and round robin scheduling in another queue.

**CODE SNIPPET:**

#include<stdio.h>

1. #include<conio.h>
2. #define N 100
3. int ct=0;
4. int ppt=0;
6. int nk=100;
7. int tick=0;
8. struct process{
9. int at,bt,prt,pid,copy\_array[N],cbt;
10. };

13. struct process2{
14. int at2,bt2,prt2,pid2;
15. int wt,ta;
16. int flage;
17. }proc2[N];
19. void copyfun(struct process record){

22. proc2[ppt].bt2=[record.bt](http://record.bt/);
23. proc2[ppt].at2=[record.at](http://record.at/);
24. proc2[ppt].prt2=record.prt;
25. proc2[ppt].pid2=record.pid;

28. ppt++;
30. }
31. //-------------------------------------------------------
32. void RR(){
34. int i, limit, total = 0, x, counter = 0, time\_quantum;
35. int wait\_time = 0, turnaround\_time = 0, arrival\_time[10], burst\_time[10], temp[10];
36. float average\_wait\_time, average\_turnaround\_time;
37. limit=ct;
38. x = limit;
39. for(i = 0; i <limit ; i++)
40. {
42. arrival\_time[i]=proc2[i].at2;
43. burst\_time[i]=proc2[i].bt2;
45. temp[i] = burst\_time[i];
46. }
47. printf("\nEnter Time Quantum:\t");
48. scanf("%d", &time\_quantum);
49. printf("\nProcess ID\t\tBurst Time\t Turnaround Time\t Waiting Time\n");
50. for(total = 0, i = 0; x != 0;)
51. {
52. if(temp[i] <= time\_quantum && temp[i] > 0)
53. {
54. total = total + temp[i];
55. temp[i] = 0;
56. counter = 1;
57. }
58. else if(temp[i] > 0)
59. {
60. temp[i] = temp[i] - time\_quantum;
61. total = total + time\_quantum;
62. }
63. if(temp[i] == 0 && counter == 1)
64. {
65. x--;
66. printf("\nProcess[%d]\t\t%d\t\t %d\t\t\t %d", proc2[i].pid2, burst\_time[i], total - arrival\_time[i], total - arrival\_time[i] - burst\_time[i]);
67. wait\_time = wait\_time + total - arrival\_time[i] - burst\_time[i];
68. turnaround\_time = turnaround\_time + total - arrival\_time[i];
69. counter = 0;
70. }
71. if(i == limit - 1)
72. {
73. i = 0;
74. }
75. else if(arrival\_time[i + 1] <= total)
76. {
77. i++;
78. }
79. else
80. {
81. i = 0;
82. }
83. }
84. average\_wait\_time = wait\_time \* 1.0 / limit;
85. average\_turnaround\_time = turnaround\_time \* 1.0 / limit;
86. printf("\n\nAverage Waiting Time:\t%f", average\_wait\_time);
87. printf("\nAvg Turnaround Time:\t%f\n", average\_turnaround\_time);


91. }
93. //-----------------------------------------------------


97. int main(){
99. int nump,j=0,i=0,k=1,count=0,flag=0,time;
100. printf("Enter the number of processes :");
101. scanf("%d",&nump);
102. struct process proc[nump];
103. for(i=0;i<nump;i++)
104. {       printf("Enter the process id here\n ");
105. scanf("%d",&proc[i].pid);
106. printf("Enter the arrival time of %d :",proc[i].pid);
107. scanf("%d",&proc[i].at);
108. printf("Enter the burst time of %d :",proc[i].pid);
109. scanf("%d",&proc[i].bt);
110. proc[i].cbt=proc[i].bt;
111. printf("Enter the priority time of %d :",proc[i].pid);
112. scanf("%d",&proc[i].prt);}
113. int fl=0;
115. for(int i=0;i<nump;i++){  //1.1
116. int btd=proc[i].cbt;
118. int scale=0;
119. k=1;
120. for(int j=0;j<btd;j++){  //1.2
121. //  printf("\n  the burst time= %d of  process :%d  of value j:%d",proc[i].bt,proc[i].pid,j);
123. if(i==0 && proc[i].at==0 && scale==0){
124. for(int l=0;l<proc[i+1].at;l++){
125. count++;
126. proc[i].bt--;
128. }
129. fl=1;
130. printf("\n%d --------- %d",proc[i].at,count);
131. time=proc[i].at+count;
133. }

136. if(i==(nump-1)){
137. if(tick==1)count--;
138. for(int m=0;m<proc[i].bt;m++){
139. count++;
140. proc[i].bt--;
142. }
143. printf("\n%d --------- %d",time,count);
144. break;}


148. if( count!=0 && ((count%2)==0) && (proc[k+i].prt>proc[i].prt)){
150. printf("\n%d --------- %d",time,count);
151. time=count;
153. count++;
154. proc[i].bt--;

157. flag++;
158. k++;
159. scale=1;
160. continue;

163. }

166. else if( count!=0 && ((count%2)==0) && (proc[i+k].prt < proc[i].prt)){
167. if(fl!=1)
168. printf("\n%d --------- %d",time,count);//printing the first quque
169. fl=0;
171. if(proc[i].bt>0){
172. ct++;
173. copyfun(proc[i]);}
174. scale=1;
175. time=count;
176. count++;


180. proc[i+1].bt--;
181. tick=1;
182. break;
183. }
185. else
186. {
188. flag++;
189. proc[i].bt--;
190. count++;
191. scale=1;

                }

**Boundary condition of implemented code:**

* Number of processes should be positive.
* Arrival time should be positive
* Burst time should be positive and multiples of two.
* Priority should be positive starting from zero.

**Test cases:**

**Case 1:**

If there is no process in the queue-1 then the process which came first to the queue-2 will be allocated to CPU. But it will in CPU only for time quantum that is given to the process.

**Case 2:**

If any processes come into ready state which belongs to first queue when a process which belong to queue-2 is in CPU, then queue-2 processes will be stop and queue-1 processes will be allocate to the CPU

**Case 3:**

If the CPU is allocated to a process which belongs to queue-1, but if any process that belongs to queue-2 comes into ready state then there will no switching of process

**Case 4:**

If there are processes in both queue-1 and queue-2 at starting then the processes in queue-1 will allocated to CPU first