Department of Computer Science and Engineering

S.G.Shivanirudh , 185001146, Semester IV

2020

UCS1411 - Operating Systems Laboratory

Exercise –4 -Implementation of CPU Scheduling Policies: Priority (Non-preemptive and Preemptive) and Round Robin

Objective:

Develop a menu driven C program to implement the CPU Scheduling Algorithms Priority (Non-Preemptive and Preemptive) and Round Robin

Code:

```
/*Develop a menu driven C program to implement the CPU
    Scheduling Algorithms
Priootity (Non-Preemptive and Preemptive) and Round Robin */
```

```c
#include<stdio.h>
#include <stdlib.h>
#include<string.h>
#include<ctype.h>

//Struture representing each process
struct Job{
    char *PID;
    double arrivalTime;
    double burstTime;
    double dummy;     //Copy of burst time
    double waitTime;
    double turnTime;
    double responseTime;
    int priority;
    int nope;         //Number of pre-emptions
    int chance;       //Keep track of chance of process in Round Robin scheduling
};

typedef struct Job Process;

void initialise(Process *p);

void acceptProcess(Process *p);

void sortOnArrivalTime(Process p[],int start_index,int end_index);
void sortOnPriority(Process p[],int number_of_processes);

void displayGanttChart(char *Gantt_Chart[],int number_of_interval,double start_times[],double end_times[]);

void printWaitTime(Process P[],int number_of_processes);
void printTurnTime(Process P[],int number_of_processes);
void printRespTime(Process P[],int number_of_processes);

void NonPriority(Process P[],int number_of_processes);
void Priority(Process P[],int number_of_processes);
void RoundRobin(Process P[],int number_of_processes,int tq);

//Queue used for implementing Round Robin scheduling
//Used as the Ready Queue
typedef struct{
```

```
44      int front,rear;
45      Process data[100];
46      int capacity,size;
47  }Queue;
48
49  //Intialise data members of the queue object
50  void initialiseQueue(Queue *q);
51
52  //Check if queue is full
53  int isFull(Queue *q);
54
55  //Check if queue is empty
56  int isEmpty(Queue *q);
57
58  //Enqueue operation
59  void enqueue(Queue *q,Process x);
60
61  //Dequeue operation
62  Process dequeue(Queue *q);
63
64  //Display queue
65  void display(Queue *q);
66
67  //Check if process is in the queue
68  int checkQueue(Process p,Queue *q);
69
70  void initialiseQueue(Queue *q){
71      q->front=q->rear=-1;
72      q->capacity=100;
73      q->size=0;
74  }
75
76  int isFull(Queue *q){
77      if((q->rear==q->capacity-1&&q->front==0)||(q->rear==q->
        front-1))
78          return 1;
79      else
80          return 0;
81  }
82
83  int isEmpty(Queue *q){
84      if(q->front==-1)
85          return 1;
86      else
87          return 0;
```

```
88 }
89
90 void enqueue(Queue *q,Process x){
91     if(isFull(q))
92         printf("Queue is full ");
93     else{
94         if(q->front==-1)
95             q->front++;
96
97         if(q->rear==q->capacity-1)
98             q->rear=0;
99         else
100            q->rear++;
101
102        q->size++;
103        q->data[q->rear]=x;
104    }
105 }
106
107 Process dequeue(Queue *q){
108     Process x;
109     initialise(&x);
110     if(isEmpty(q))
111         printf("Queue is empty");
112     else{
113         x=q->data[q->front];
114         q->size--;
115         if(q->front==q->rear)
116             q->front=q->rear=-1;
117         else if(q->front==q->capacity-1)
118             q->front=0;
119         else
120             q->front++;
121     }
122     return x;
123 }
124
125 void display(Queue *q){
126     if(isEmpty(q))
127         printf("\nQueue is empty\n");
128     else{
129             int i=q->front;
130             while(i!=q->rear){
131                 printf("%s ",q->data[i].PID);
132
```

```
                if(i==q->capacity-1)
                    i=0;
                else
                    i++;
            }

            printf("%s ",q->data[i].PID);
    }
}

int checkQueue(Process p,Queue *q){
    if(isEmpty(q))
        printf("\nQueue is empty\n");
    else{
            int i=q->front;
            while(i!=q->rear){
                if(strcmp(p.PID,q->data[i].PID)==0)
                    return 1;

                if(i==q->capacity-1)
                    i=0;
                else
                    i++;
            }

            if(strcmp(p.PID,q->data[i].PID)==0)
                return 1;
            return 0;
    }
}


//Initialising the data members of each process
void initialise(Process *p){
    p->PID=(char*)malloc(100*sizeof(char));
    p->arrivalTime=0.0;
    p->burstTime=0.0;
    p->dummy=0.0;
    p->waitTime=0.0;
    p->turnTime=0.0;
    p->responseTime=-1.0;
    p->priority=0;
    p->nope=0;
    p->chance=0;
}

```

```
16 //Accepting data of each process
17 void acceptProcess(Process *p){
18     printf("\n Enter Process ID: ");scanf(" %s",p->PID);
19     printf("\n Enter arrival time: ");scanf("%lf",&p->
       arrivalTime);
20     printf("\n Enter burst time: ");scanf("%lf",&p->burstTime
       );
21     printf("\n Enter priority: ");scanf("%d",&p->priority);
22     p->dummy=p->burstTime;
23 }
24
25 //Sorting on arrival time using Insertion sort
26 void sortOnArrivalTime(Process p[],int start_index,int
       end_index){
27     for(int i=start_index;i<end_index;i++){
28         Process key=p[i];
29         int j=i-1;
30         for(;j>=start_index&&key.arrivalTime<p[j].arrivalTime
       ;j--)
31             p[j+1]=p[j];
32         p[j+1]=key;
33     }
34 }
35
36 //Sorting based on Priority
37 void sortOnPriority(Process p[],int number_of_processes){
38
39     for(int i=0;i<number_of_processes;i++){
40         Process key=p[i];
41         int j=i-1;
42         for(;j>=0&&key.priority<p[j].priority;j--)
43             p[j+1]=p[j];
44         p[j+1]=key;
45     }
46 }
47
48 //Display Gantt Chart
49 void displayGanttChart(char *Gantt_Chart[],int
       number_of_interval,double start_times[],double end_times
       []){
50
51     //Display top line
52     printf("\n Gantt_Chart:\n");
53     for(int i=0;i<number_of_interval;i++){
54         printf("_____________");
```

```
    }

    //Display order of processes
    printf("\n|");
    for(int i=0;i<number_of_interval;i++)
        printf("_____%4s_____|",Gantt_Chart[i]);
    printf("\n");

    //Display time line
    int i=0;
    for(i=0;i<number_of_interval;i++)
        printf("%-15.0lf",start_times[i]);
    printf("%-15.0lf",end_times[i-1]);
    printf("\n\n");
}

//Print Wait Time
void printWaitTime(Process P[],int number_of_processes){
    int i=0;
    double sum=0.0;
    printf("\n Wait Time:\n");
    for(i=0;i<number_of_processes;i++){
        printf(" %-5.2lf",P[i].waitTime);
        sum+=P[i].waitTime;
    }
    printf("\nAverage: %-5.2lf",sum/number_of_processes);
    printf("\n");
}

//Print Turnaround Time
void printTurnTime(Process P[],int number_of_processes){
    int i=0;
    double sum=0.0;
    printf("\n Turnaround Time:\n");
    for(i=0;i<number_of_processes;i++){
        printf(" %-5.2lf",P[i].turnTime);
        sum+=P[i].turnTime;
    }
    printf("\nAverage: %-5.2lf",sum/number_of_processes);
    printf("\n");
}

//Print Response Time
void printRespTime(Process P[],int number_of_processes){
    int i=0;
```

```
100     double sum=0.0;
101     printf("\n Response Time:\n");
102     for(i=0;i<number_of_processes;i++){
103         if(P[i].responseTime<0)
104             P[i].responseTime=0.0;
105         printf(" %-5.2lf",P[i].responseTime);
106         sum+=P[i].responseTime;
107     }
108     printf("\nAverage: %-5.2lf",sum/number_of_processes);
109     printf("\n");
110 }
111 
112 //Non-Preemptive Priority Scheduling
113 /*
114 Logic:
115 1.Maintain arrays for start and end times of the intervals in
        the Gantt chart
116 2.Run a timer from 0 to maximum time elapsed
117 3.In each iteration, add the processes that have arrived to a
        temporary array, provided they are incomplete
118 4.Sort the processes in the temporary array based on their
      priorities
119 5.Insert the processes in the temporary array into the gantt
      chart. Set the inserted processes as complete
120 6.Assign start and end times for the intervals.
121 7.Move time to the end time of the intervals.
122 8.Repeat steps 3 to 7.
123 9.Compute wait, response and turnaround times
124 */
125 void NonPriority(Process P[],int number_of_processes){
126     //Total time of execution
127     double sum=0;
128     for(int i=0;i<number_of_processes;i++)
129         sum+=P[i].burstTime;
130 
131     //Gantt chart
132     char *Gantt_Chart[100];
133     for(int i=0;i<100;i++)
134         Gantt_Chart[i]=(char*)malloc(10*sizeof(char));
135 
136     //Step 1.
137     //Start and end times of processes
138     int interval=0;
139     double start_times[100];
140     double end_times[100];
```

```
141
142     //Step 2.
143     for(int time=0;time<sum;){
144
145         Process tmp[100];
146         for(int i=0;i<100;i++)
147             initialise(&tmp[i]);
148
149         //Step 3.
150         int tctr=0;
151         for(int i=0;i<number_of_processes;i++)
152             if(P[i].arrivalTime<=time&&P[i].priority){
153                 tmp[tctr++]=P[i];
154             }
155
156         //Step 4.
157         sortOnPriority(tmp,tctr);
158
159         if(tctr==0){
160             strcpy(Gantt_Chart[interval],"////");
161             if(interval==0){
162                 start_times[interval]=0;
163             }
164             else{
165                 start_times[interval]=end_times[interval-1];
166             }
167         }
168         else{
169             for(int i=0;i<tctr;i++){
170                 //Step 5.
171                 strcpy(Gantt_Chart[interval],tmp[i].PID);
172
173                 //Step 6.
174                 if(interval==0){
175                     start_times[interval]=0;
176                 }
177                 else{
178                     start_times[interval]=end_times[interval
    -1];
179                 }
180                 end_times[interval]=start_times[interval]+tmp
    [i].burstTime;
181                 int j=0;
182                 for(j=0;j<number_of_processes;j++){
183                     if(strcmp(tmp[i].PID,P[j].PID)==0){
```

```
184                         //Step 9.
185                         P[j].priority=0;
186                         P[j].waitTime=start_times[interval]-P
    [j].arrivalTime;
187                         P[j].turnTime=P[j].waitTime+P[j].
    burstTime;
188                         P[j].responseTime=P[j].waitTime;
189                     }
190                 }
191                 interval++;
192             }
193         }
194         //Step 7.
195         time=end_times[interval-1];
196     }
197     displayGanttChart(Gantt_Chart,interval,start_times,
    end_times);
198     printWaitTime(P,number_of_processes);
199     printTurnTime(P,number_of_processes);
200     printRespTime(P,number_of_processes);
201 }
202
203 //Preemptive Priority Scheduling
204 /*
205 Logic:
206 1.Maintain arrays for start and end times of the intervals in
      the Gantt chart
207 2.Run a timer from 0 to maximum time elapsed
208 3.In each iteration, add the processes that have arrived to a
      temporary array, provided they are incomplete
209 4.Sort the processes in the temporary array based on their
    priorities
210 5.Insert the process at the zeroth index of the temporary
    array into the gantt chart.
211     5.1 Decrement the burst time of the inserted process by
    the necessary amount
212 6.Assign start and end times for that interval
213 7.Increment value of time by 1
214 8.Repeat steps 3 to 7.
215 9.Compute wait, response and turnaround times
216 */
217 void Priority(Process P[],int number_of_processes){
218     //Total time of execution
219     double sum=0;
220     for(int i=0;i<number_of_processes;i++)
```

```
221         sum+=P[i].burstTime;
222 
223     //Gantt chart
224     char *Gantt_Chart[100];
225     for(int i=0;i<100;i++)
226         Gantt_Chart[i]=(char*)malloc(10*sizeof(char));
227 
228     //Step 1.
229     //Start and end times of processes
230     int interval=0;
231     double start_times[100];
232     double end_times[100];
233 
234     //Step 2.
235     //Step 7. Note time++ instead of time = end_times[
    interval-1] in Non-preemptive Priority
236     for(int time=0;time<sum;time++){
237         int flag=0;
238 
239         Process tmp[100];
240         for(int i=0;i<100;i++)
241             initialise(&tmp[i]);
242 
243         //Step 3.
244         int tctr=0;
245         for(int i=0;i<number_of_processes;i++)
246             if(P[i].arrivalTime<=time&&P[i].burstTime){
247                 tmp[tctr++]=P[i];
248             }
249 
250         //Step 4.
251         sortOnPriority(tmp,tctr);
252 
253         //Step 5.1
254         for(int i=0;i<number_of_processes;i++){
255             if(strcmp(tmp[0].PID,P[i].PID)==0)
256                 P[i].burstTime--;
257         }
258 
259         //Step 5.
260         //Step 6.
261         if(interval==0){
262             strcpy(Gantt_Chart[interval],tmp[0].PID);
263             start_times[interval]=0;
264             flag=1;
```

```
265             interval++;
266         }
267         else{
268             //Step 6.
269             if(strcmp(Gantt_Chart[interval-1],tmp[0].PID)!=0)
    {
270                 end_times[interval-1]=time;
271                 strcpy(Gantt_Chart[interval],tmp[0].PID);
272                 start_times[interval]=end_times[interval-1];
273                 flag=1;
274                 interval++;
275             }
276         }
277         //Step 9.
278         int j=0;
279         for(j=0;j<number_of_processes;j++){
280             if(flag&&strcmp(tmp[0].PID,P[j].PID)==0){
281                 P[j].waitTime+=start_times[interval-1]-P[j].
    arrivalTime;
282                 if(P[j].waitTime>0.0){
283                     P[j].nope++;
284                     P[j].waitTime-=(P[j].dummy-P[j].burstTime
    -P[j].nope);
285                     printf("\n%s %d",P[j].PID,P[j].nope);
286                     if(P[j].nope>1){
287                         P[j].waitTime-=P[j].nope;
288                     }
289                 }
290 
291                 P[j].turnTime=P[j].waitTime+P[j].dummy;
292                 if(P[j].responseTime<0.0)
293                     P[j].responseTime=start_times[interval
    -1]-P[j].arrivalTime;
294             }
295         }
296     }
297     end_times[interval-1]=sum;
298     displayGanttChart(Gantt_Chart,interval,start_times,
    end_times);
299     printWaitTime(P,number_of_processes);
300     printTurnTime(P,number_of_processes);
301     printRespTime(P,number_of_processes);
302 }
303 
304 //Round Robin Scheduling
```

```
/*
Logic:
1. Maintain arrays for the gantt chart, start and end times
   of the intervals
2. Maintain a queue object to serve as the Ready queue
3. Run a timer from 0 to maximum time elapsed
4. At each iteration, enqueue all the processed that have
   arrived, if they are incomplete, to the ready queue
5. If queue is empty, break the loop
6. Dequeue and add the process to the gantt chart.
    6.1 Decrement the burst time of that process by the time
   quantum or remaining burst time, whichever is smaller.
7. If the process is still incomplete, add it back to the
   Ready Queue.
    7.1 Ensure no process has more than one instance in the
   ready queue at any given point of time
8. Allot start and end times for that interval.
9. Move time to the end of that interval
10. Repeat steps 4. to 9.
11. Compute wait times, turnaround times and response times.
*/
void RoundRobin(Process P[],int number_of_processes,int tq){
    //Total time of execution
    double sum=0;
    for(int i=0;i<number_of_processes;i++)
        sum+=P[i].burstTime;

    //Gantt chart
    char *Gantt_Chart[100];
    for(int i=0;i<100;i++)
        Gantt_Chart[i]=(char*)malloc(10*sizeof(char));

    //Step 1.
    //Start and end times of processes
    int interval=0;
    double start_times[100];
    double end_times[100];

    //Step 2.
    Queue RQ;
    initialiseQueue(&RQ);

    //Step 3.
    for(int time=0;time<sum;){

```

```
345         //Step 4.
346         for(int i=0;i<number_of_processes;i++){
347             if(isEmpty(&RQ)){
348                 if(P[i].arrivalTime<=time&&P[i].burstTime>0&&
    P[i].chance==0){
349                     enqueue(&RQ,P[i]);
350                     P[i].chance=1;
351                 }
352             }
353             else{
354                 if(P[i].arrivalTime<=time&&P[i].burstTime>0&&
    P[i].chance==0&&!checkQueue(P[i],&RQ)){
355                     enqueue(&RQ,P[i]);
356                     P[i].chance=1;
357                 }
358             }
359         }
360 
361         Process DQ;
362         initialise(&DQ);
363 
364         //Step 5.
365         if(isEmpty(&RQ))
366             break;
367         while(isEmpty(&RQ)==0){
368             //Step 6.
369             DQ=dequeue(&RQ);
370             strcpy(Gantt_Chart[interval],DQ.PID);
371             //Step 8.
372             if(interval==0){
373                 start_times[interval]=0;
374             }
375             else{
376                 start_times[interval]=end_times[interval-1];
377             }
378 
379             //Step 6.1
380             //Step 8.
381             for(int i=0;i<number_of_processes;i++){
382                 if(strcmp(P[i].PID,Gantt_Chart[interval])==0)
    {
383                     if(P[i].burstTime<tq)
384                         end_times[interval]=start_times[
    interval]+P[i].burstTime;
385                     else
```

```
386                         end_times[interval]=start_times[
    interval]+tq;
387                     P[i].burstTime-=tq;
388                     P[i].burstTime=(P[i].burstTime<0)?0:P[i].
    burstTime;
389                 }
390             }
391 
392             //Step 9.
393             time=end_times[interval];
394             interval++;
395 
396             //Step 11.
397             int j=0;
398             for(j=0;j<number_of_processes;j++){
399                 if(strcmp(DQ.PID,P[j].PID)==0){
400 
401                     P[j].waitTime+=start_times[interval-1]-P[
    j].arrivalTime;
402 
403                     if(P[j].burstTime>0)
404                         P[j].nope++;
405                     if(P[j].waitTime>0.0){
406                         P[j].waitTime-=(P[j].dummy-P[j].
    burstTime-P[j].nope);
407                         if(P[j].waitTime<0.0)
408                             P[j].waitTime+=(P[j].dummy-P[j].
    burstTime);
409                         if(P[j].nope>=1){
410                             P[j].waitTime-=P[j].nope;
411                         }
412                     }
413 
414                     P[j].turnTime=P[j].waitTime+P[j].dummy;
415                     if(P[j].responseTime<0.0)
416                         P[j].responseTime=start_times[
    interval-1]-P[j].arrivalTime;
417                 }
418             }
419 
420             //Step 7.1 Note chance keeps track of number of
    instances of a process in the ready queue
421             //Chance takes value 1 if the process is already
    in the ready queue, and 0 otherwise.
422             for(int i=0;i<number_of_processes;i++){
```

```
                if(isEmpty(&RQ)){
                    if(P[i].arrivalTime<=time&&P[i].burstTime>0&&P[i].chance==0){
                        enqueue(&RQ,P[i]);
                        P[i].chance=1;
                    }
                }
                else{
                    if(P[i].arrivalTime<=time&&P[i].burstTime>0&&P[i].chance==0&&!checkQueue(P[i],&RQ)){
                        enqueue(&RQ,P[i]);
                        P[i].chance=1;
                    }
                }

            }
            //Step 7.
            for(int i=0;i<number_of_processes;i++){
                if(strcmp(DQ.PID,P[i].PID)==0)
                    if(P[i].burstTime>0)
                        enqueue(&RQ,P[i]);
            }

        }

    }
    end_times[interval-1]=sum;
    displayGanttChart(Gantt_Chart,interval,start_times,end_times);
    printWaitTime(P,number_of_processes);
    printTurnTime(P,number_of_processes);
    printRespTime(P,number_of_processes);
}

int main(){
    printf("\n\t\tCPU SCHEDULING ALGORITHMS\n");
    Process p[100];
    int number_of_processes;
int algo_option;
    do{
        printf("\nChoose your scheduling algorithm ");
        printf("\n1. Round Robin\n2. Priority\n0. Exit\n Your Choice: ");
        scanf("%d",&algo_option);

```

```
12          //RoundRobin Scheduling
13          if(algo_option==1){
14              printf("\nEnter the number_of_processes:");scanf(
    "%d",&number_of_processes);
15              printf("\nEnter the details of the processes:");
16
17              int i;
18              for(i=0;i<number_of_processes;i++){
19                  initialise(&p[i]);
20                  acceptProcess(&p[i]);
21              }
22              int tq;
23              printf("\nEnter the time quantum: ");scanf("%d",&
    tq);
24
25              Process RRp[100];
26              for(i=0;i<number_of_processes;i++){
27                  initialise(&RRp[i]);
28                  RRp[i]=p[i];
29              }
30              printf("\n Round Robin Scheduling Output:\n ");
31              RoundRobin(RRp,number_of_processes,tq);
32
33          }
34          //Priority Scheduling
35          else if(algo_option==2){
36              printf("\nEnter the number_of_processes:");scanf(
    "%d",&number_of_processes);
37              printf("\nEnter the details of the processes:");
38
39              int i;
40              for(i=0;i<number_of_processes;i++){
41                  initialise(&p[i]);
42                  acceptProcess(&p[i]);
43              }
44
45              char preemp_option;
46              printf("\n Use Pre-emption? y/n ");scanf(" %c",&
    preemp_option);
47              //Non preemptive Priority Scheduling
48              if(preemp_option=='n'||preemp_option=='N'){
49
50                  Process NPrip[100];
51                  for(i=0;i<number_of_processes;i++){
52                      initialise(&NPrip[i]);
```

```
53                      NPrip[i]=p[i];
54                  }
55
56                  printf("\n Non-preemptive SJF Scheduling
    Output:\n ");
57                  NonPriority(NPrip,number_of_processes);
58              }
59              //Preemptive Priority Scheduling
60              else if(preemp_option=='y'||preemp_option=='Y'){
61                  Process PPrip[100];
62                  for(i=0;i<number_of_processes;i++){
63                      initialise(&PPrip[i]);
64                      PPrip[i]=p[i];
65                  }
66
67                  printf("\n Preemptive Priority Scheduling
    Output:\n ");
68                  Priority(p,number_of_processes);
69              }
70              else{
71                  printf("\n Invalid choice\n");
72              }
73          }
74          else if(algo_option!=0){
75              printf("\n Invalid option\n");
76          }
77          else;
78      }while(algo_option);
79  }
```