Department of Computer Science and Engineering

S.G.Shivanirudh , 185001146, Semester IV

2020

UCS1411 - Operating Systems Laboratory

Exercise –3-Implementation of CPU Scheduling Policies: FCFS and SJF (Non-preemptive and Preemptive)

Objective:

Develop a menu driven C program to implement the CPU Scheduling Algorithms FCFS and SJF (Non-Preemptive and Preemptive)

Code:

```
1 /*Develop a menu driven C program to implement the CPU
     Scheduling Algorithms
2 FCFS and SJF (Non-Preemptive and Preemptive)*/
```

```c
#include<stdio.h>
#include <stdlib.h>
#include<string.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 nope;      //Number of pre-emptions
};

typedef struct Job Process;

//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->nope=0;
}

//Accepting data of each process
void acceptProcess(Process *p){
    printf("\n Enter Process ID: ");scanf(" %s",p->PID);
    printf("\n Enter arrival time: ");scanf("%lf",&p->
   arrivalTime);
    printf("\n Enter burst time: ");scanf("%lf",&p->burstTime
   );
    p->dummy=p->burstTime;
}

//Display Processes
void displayProcesses(Process p[],int number_of_processes){
    for(int i=0;i<number_of_processes;i++)
        printf("\t%s",p[i].PID);
    printf("\n");
```

```
}

//Sorting using Insertion sort
void sortOnArrivalTime(Process p[],int start_index,int
    end_index){
     for(int i=start_index;i<end_index;i++){
         Process key=p[i];
         int j=i-1;
         for(;j>=start_index&&key.arrivalTime<p[j].arrivalTime
    ;j--)
              p[j+1]=p[j];
         p[j+1]=key;
     }
}

//Display Gantt Chart
void displayGanttChart(char *Gantt_Chart[],int
    number_of_interval,double start_times[],double end_times
    []){

     //Display top line
     printf("\n Gantt_Chart:\n");
     for(int i=0;i<number_of_interval;i++){
         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");
```

```
87      for(i=0;i<number_of_processes;i++){
88          printf(" %-5.2lf",P[i].waitTime);
89          sum+=P[i].waitTime;
90      }
91      printf("\nAverage: %-5.2lf",sum/number_of_processes);
92      printf("\n");
93  }
94
95  //Print Turnaround Time
96  void printTurnTime(Process P[],int number_of_processes){
97      int i=0;
98      double sum=0.0;
99      printf("\n Turnaround Time:\n");
100     for(i=0;i<number_of_processes;i++){
101         printf(" %-5.2lf",P[i].turnTime);
102         sum+=P[i].turnTime;
103     }
104     printf("\nAverage: %-5.2lf",sum/number_of_processes);
105     printf("\n");
106 }
107
108 //Print Response Time
109 void printRespTime(Process P[],int number_of_processes){
110     int i=0;
111     double sum=0.0;
112     printf("\n Response Time:\n");
113     for(i=0;i<number_of_processes;i++){
114         if(P[i].responseTime<0)
115             P[i].responseTime=0.0;
116         printf(" %-5.2lf",P[i].responseTime);
117         sum+=P[i].responseTime;
118     }
119     printf("\nAverage: %-5.2lf",sum/number_of_processes);
120     printf("\n");
121 }
122
123 //FCFS Scheduling
124 /*Logic:
125 1.Maintain arrays for start and end times of the intervals in
        the Gantt chart
126 2.Sort the processes based on their arrival times
127 3.Insert the processes into the Gantt chart
128 4.Assign start and end times for the intervals
129 5.Compute wait, response and turnaround times
130 */
```

```
131 void FCFS(Process P[],int number_of_processes){
132     //Step 1.
133     char *Gantt_Chart[100];
134     for(int i=0;i<100;i++)
135         Gantt_Chart[i]=(char*)malloc(10*sizeof(char));
136 
137     int interval=0;
138     double start_times[100];
139     double end_times[100];
140 
141     //Step 2.
142     sortOnArrivalTime(P,0,number_of_processes);
143 
144     //Step 3.
145     for(int i=0;i<number_of_processes;i++){
146         strcpy(Gantt_Chart[interval],P[i].PID);
147 
148         //Step 4.
149         if(interval==0){
150             start_times[interval]=0;
151         }
152         else{
153             start_times[interval]=end_times[interval-1];
154         }
155         end_times[interval]=start_times[interval]+P[i].
    burstTime;
156 
157         //Step 5.
158         P[i].waitTime=start_times[interval]-P[i].arrivalTime;
159         P[i].turnTime=P[i].waitTime+P[i].burstTime;
160         P[i].responseTime=P[i].waitTime;
161         interval++;
162     }
163 
164     displayGanttChart(Gantt_Chart,interval,start_times,
    end_times);
165     printWaitTime(P,number_of_processes);
166     printTurnTime(P,number_of_processes);
167     printRespTime(P,number_of_processes);
168 }
169 
170 //Sorting on Burst time
171 void sortOnBurstTime(Process p[],int number_of_processes){
172 
173     for(int i=0;i<number_of_processes;i++){
```

```
174         Process key=p[i];
175         int j=i-1;
176         for(;j>=0&&key.burstTime<p[j].burstTime;j--)
177             p[j+1]=p[j];
178         p[j+1]=key;
179     }
180 
181 }
182 
183 //SJF Non-preemptive Scheduling
184 /*Logic:
185 1.Maintain arrays for start and end times of the intervals in
      the Gantt chart
186 2.Run a timer from 0 to total time taken
187 3.At each iteration, check which processes have arrived and
     store them in a temporary array. Make the processes' burst
      time as zero
188 4.Sort the temporary array on basis of the processes' burst
     time
189 5.Insert the processes in the temporary array into the Gantt
     chart
190 6.Assign start and end times for the intervals
191 7.Move the value of time to the end time of that interval
192 8.Repeat procedure 3. to 7. till timer reaches the end of the
      total time
193 9.Compute wait, response and turnaround times
194 */
195 void Non_PreSJF(Process P[],int number_of_processes){
196     //Total time of execution
197     double sum=0;
198     for(int i=0;i<number_of_processes;i++)
199         sum+=P[i].burstTime;
200 
201     //Gantt chart
202     char *Gantt_Chart[100];
203     for(int i=0;i<100;i++)
204         Gantt_Chart[i]=(char*)malloc(10*sizeof(char));
205 
206     //Start and end times of processes
207     //Step 1.
208     int interval=0;
209     double start_times[100];
210     double end_times[100];
211 
212     //Step 2.
```

```
213     for(int time=0;time<sum;){
214
215         Process tmp[100];
216         for(int i=0;i<100;i++)
217             initialise(&tmp[i]);
218
219         //Step 3.
220         int tctr=0;
221         for(int i=0;i<number_of_processes;i++)
222             if(P[i].arrivalTime<=time&&P[i].burstTime){
223                 tmp[tctr++]=P[i];
224                 P[i].burstTime=0;
225             }
226
227         //Step 4.
228         sortOnBurstTime(tmp,tctr);
229
230         //Step 5.
231         for(int i=0;i<tctr;i++){
232             strcpy(Gantt_Chart[interval],tmp[i].PID);
233
234             //Step 6.
235             if(interval==0){
236                 start_times[interval]=0;
237             }
238             else{
239                 start_times[interval]=end_times[interval-1];
240             }
241             end_times[interval]=start_times[interval]+tmp[i].
    burstTime;
242             int j=0;
243             for(j=0;j<number_of_processes;j++){
244                 if(strcmp(tmp[i].PID,P[j].PID)==0){
245
246                     //Step 8.
247                     P[j].waitTime=start_times[interval]-P[j].
    arrivalTime;
248                     P[j].turnTime=P[j].waitTime+P[j].dummy;
249                     P[j].responseTime=P[j].waitTime;
250                 }
251             }
252             interval++;
253         }
254         //Step 7.
255         time=end_times[interval-1];
```

```
256     }
257     displayGanttChart(Gantt_Chart,interval,start_times,
        end_times);
258     printWaitTime(P,number_of_processes);
259     printTurnTime(P,number_of_processes);
260     printRespTime(P,number_of_processes);
261 }
262
263 //SJF Preemptive Scheduling
264 /*Logic:
265 1.Maintain arrays for start and end times of the intervals in
        the Gantt chart
266 2.Run a timer from 0 to total time taken
267 3.At each iteration, check which processes have arrived and
        store them in a temporary array. Decrement burst time of
        processes properly.
268 4.Sort the temporary array on basis of the processes' burst
        time
269 5.Insert the process at the zeroth index, i.e, the first
        process into the Gantt chart
270     5.1 If the process previously inserted is the same as the
        next one to be inserted, move interval and time
        appropriately
271 6.Assign start and end times for the intervals
272 7.Increment value of time by 1
273 8.Repeat procedure 3. to 7. till timer reaches the end of the
        total time
274 9.Compute wait, response and turnaround times
275 */
276 void PreSJF(Process P[],int number_of_processes){
277     //Total time of execution
278     double sum=0;
279     for(int i=0;i<number_of_processes;i++)
280         sum+=P[i].burstTime;
281
282     //Gantt chart
283     char *Gantt_Chart[100];
284     for(int i=0;i<100;i++)
285         Gantt_Chart[i]=(char*)malloc(10*sizeof(char));
286
287     //Start and end times of processes
288     //Step 1.
289     int interval=0;
290     double start_times[100];
291     double end_times[100];
```

```
292
293     //Step 2.
294     //Step 7. Note time++ instead of time = end_times[
        interval-1] as was the case in non-preemptive SJF
295     for(int time=0;time<sum;time++){
296         int flag=0;
297
298         Process tmp[100];
299         for(int i=0;i<100;i++)
300             initialise(&tmp[i]);
301
302         //Step 3.
303         int tctr=0;
304         for(int i=0;i<number_of_processes;i++)
305             if(P[i].arrivalTime<=time&&P[i].burstTime){
306                 tmp[tctr++]=P[i];
307             }
308
309         //Step 4.
310         sortOnBurstTime(tmp,tctr);
311
312         for(int i=0;i<number_of_processes;i++){
313             if(strcmp(tmp[0].PID,P[i].PID)==0)
314                 P[i].burstTime--;
315         }
316
317         //Step 5.
318         if(interval==0){
319             strcpy(Gantt_Chart[interval],tmp[0].PID);
320             start_times[interval]=0;
321             flag=1;
322             interval++;
323         }
324         else{
325             //Step 5.1
326             //Step 6.
327             if(strcmp(Gantt_Chart[interval-1],tmp[0].PID)!=0)
        {
328                 end_times[interval-1]=time;
329                 strcpy(Gantt_Chart[interval],tmp[0].PID);
330                 start_times[interval]=end_times[interval-1];
331                 flag=1;
332                 interval++;
333             }
334         }
```

```
335         //Step 9.
336         int j=0;
337         for(j=0;j<number_of_processes;j++){
338             if(flag&&strcmp(tmp[0].PID,P[j].PID)==0){
339                 P[j].waitTime+=start_times[interval-1]-P[j].
    arrivalTime;
340                 if(P[j].waitTime>0.0){
341                     P[j].nope++;
342                     P[j].waitTime-=(P[j].dummy-P[j].burstTime
    -P[j].nope);
343                     if(P[j].nope>1){
344                         P[j].waitTime-=P[j].nope;
345                     }
346                 }
347 
348                 P[j].turnTime=P[j].waitTime+P[j].dummy;
349                 if(P[j].responseTime<0.0)
350                     P[j].responseTime=start_times[interval
    -1]-P[j].arrivalTime;
351             }
352         }
353     }
354     end_times[interval-1]=sum;
355     displayGanttChart(Gantt_Chart,interval,start_times,
    end_times);
356     printWaitTime(P,number_of_processes);
357     printTurnTime(P,number_of_processes);
358     printRespTime(P,number_of_processes);
359 }
360 
361 
362 
363 int main(){
364     printf("\n\t\tCPU SCHEDULING ALGORITHMS\n");
365     Process p[100];
366     int number_of_processes;
367 int algo_option;
368     do{
369         printf("\nChoose your scheduling algorithm ");
370         printf("\n1. FCFS\n2. SJF\n0. Exit\n Your Choice: ");
371         scanf("%d",&algo_option);
372 
373         //FCFS Scheduling
374         if(algo_option==1){
375             printf("\nEnter the number_of_processes:");scanf(
```

```
"%d",&number_of_processes);
376             printf("\nEnter the details of the processes:");
377
378             int i;
379             for(i=0;i<number_of_processes;i++){
380                 initialise(&p[i]);
381                 acceptProcess(&p[i]);
382             }
383
384             Process FCFSp[100];
385             for(i=0;i<number_of_processes;i++){
386                 initialise(&FCFSp[i]);
387                 FCFSp[i]=p[i];
388             }
389             printf("\n FCFS Scheduling Output:\n ");
390             FCFS(FCFSp,number_of_processes);
391
392         }
393         //SJF Scheduling
394         else if(algo_option==2){
395             printf("\nEnter the number_of_processes:");scanf(
"%d",&number_of_processes);
396             printf("\nEnter the details of the processes:");
397
398             int i;
399             for(i=0;i<number_of_processes;i++){
400                 initialise(&p[i]);
401                 acceptProcess(&p[i]);
402             }
403
404             char preemp_option;
405             printf("\n Use Pre-emption? y/n ");scanf(" %c",&
preemp_option);
406             //Non preemptive SJF Scheduling
407             if(preemp_option=='n'||preemp_option=='N'){
408
409                 Process NSJFp[100];
410                 for(i=0;i<number_of_processes;i++){
411                     initialise(&NSJFp[i]);
412                     NSJFp[i]=p[i];
413                 }
414
415                 printf("\n Non-preemptive SJF Scheduling
Output:\n ");
416                 Non_PreSJF(NSJFp,number_of_processes);
```

```
            }
            //Preemptive SJF Scheduling
            else if(preemp_option=='y'||preemp_option=='Y'){
                Process SJFp[100];
                for(i=0;i<number_of_processes;i++){
                    initialise(&SJFp[i]);
                    SJFp[i]=p[i];
                }

                printf("\n Preemptive SJF Scheduling Output:\
    n ");
                PreSJF(SJFp,number_of_processes);
            }
            else{
                printf("\n Invalid choice\n");
            }
        }
        else if(algo_option!=0){
            printf("\n Invalid option\n");
        }
        else;
    }while(algo_option);
}

/*
Output:
shivanirudh@shiva-ideapad:~/Desktop/Semester4/OSLAB/
    CPUScheduling$ gcc Scheduling.c -o a
shivanirudh@shiva-ideapad:~/Desktop/Semester4/OSLAB/
    CPUScheduling$ ./a

        CPU SCHEDULING ALGORITHMS

Choose your scheduling algorithm
1. FCFS
2. SJF
0. Exit
 Your Choice: 1

Enter the number_of_processes:5

Enter the details of the processes:
 Enter Process ID: p1

 Enter arrival time: 0
```

```
459
460  Enter burst time: 8
461
462  Enter Process ID: p2
463
464  Enter arrival time: 1
465
466  Enter burst time: 6
467
468  Enter Process ID: p3
469
470  Enter arrival time: 2
471
472  Enter burst time: 1
473
474  Enter Process ID: p4
475
476  Enter arrival time: 3
477
478  Enter burst time: 9
479
480  Enter Process ID: p5
481
482  Enter arrival time: 4
483
484  Enter burst time: 3
485
486  FCFS Scheduling Output:
487
488  Gantt_Chart:
489 ------------------------------------------------------------------------
490 |_____  p1_____|_____  p2_____|_____  p3_____|_____  p4_____|
      _____  p5_____|
491 0              8              14             15
     24             27
492
493
494  Wait Time:
495  0.00  7.00  12.00 12.00 20.00
496 Average: 10.20
497
498  Turnaround Time:
499  8.00  13.00 13.00 21.00 23.00
500 Average: 15.60
```

```
 Response Time:
 0.00   7.00   12.00 12.00 20.00
Average: 10.20

Choose your scheduling algorithm
1. FCFS
2. SJF
0. Exit
 Your Choice: 2

Enter the number_of_processes:5

Enter the details of the processes:
 Enter Process ID: p1

 Enter arrival time: 0

 Enter burst time: 8

 Enter Process ID: p2

 Enter arrival time: 1

 Enter burst time: 6

 Enter Process ID: p3

 Enter arrival time: 2

 Enter burst time: 1

 Enter Process ID: p4

 Enter arrival time: 3

 Enter burst time: 9

 Enter Process ID: p5

 Enter arrival time: 4

 Enter burst time: 3

 Use Pre-emption? y/n n
```

```
 Non-preemptive SJF Scheduling Output:

 Gantt_Chart:
--------------------------------------------------------------------------------
|_____  p1_____|_____  p3_____|_____  p5_____|_____  p2_____|_____  p4_____|
0               8              9              12             18             27


 Wait Time:
 0.00  11.00 6.00  15.00 5.00
Average: 7.40

 Turnaround Time:
 8.00  17.00 7.00  24.00 8.00
Average: 12.80

 Response Time:
 0.00  11.00 6.00  15.00 5.00
Average: 7.40

Choose your scheduling algorithm
1. FCFS
2. SJF
0. Exit
 Your Choice: 2

Enter the number_of_processes:5

Enter the details of the processes:
 Enter Process ID: p1

 Enter arrival time: 0

 Enter burst time: 8

 Enter Process ID: p2

 Enter arrival time: 1

 Enter burst time: 6
```

```
 Enter Process ID: p3

 Enter arrival time: 2

 Enter burst time: 1

 Enter Process ID: p4

 Enter arrival time: 3

 Enter burst time: 9

 Enter Process ID: p5

 Enter arrival time: 4

 Enter burst time: 3

 Use Pre-emption? y/n y

 Preemptive SJF Scheduling Output:

 Gantt_Chart:
--------------------------------------------------------------------------------
|_____  p1_____|_____  p2_____|_____  p3_____|_____  p2_____|
   _____  p5_____|_____  p2_____|_____  p1_____|_____
   p4_____|
0              1              2              3              4
                 7              11             18
     27


 Wait Time:
 10.00 4.00  0.00  15.00 0.00
Average: 5.80

 Turnaround Time:
 18.00 10.00 1.00  24.00 3.00
Average: 11.20

 Response Time:
 0.00  0.00  0.00  15.00 0.00
Average: 3.00
```

```
628 Choose your scheduling algorithm
629 1. FCFS
630 2. SJF
631 0. Exit
632  Your Choice: 0
633 */
```