

MIT WORLD PEACE UNIVERSITY

Operating Systems
Second Year B. Tech, Semester 3

SIMULATION OF CPU SCHEDULING ALGORITHMS
- FCFS AND ROUND ROBIN

ASSIGNMENT 3
PRACTICAL REPORT

Prepared By
Krishnaraj Thadesar
Cyber Security and Forensics
Batch A2, PA 20
November 3, 2022

1 Code

```
1 #include <stdio.h>
2 #include <math.h>
3 // Turnaround time - completion time - Arrival time
4 // Waiting time = Turn around time - Burst time
5 // Write a program to simulate the first come Non preemptive first come first
   serve cpu scheduling algorithm.
6
7 struct Process
8 {
9     int process_id;
10    int burst_time;
11    int arrival_time;
12    int service_time;
13    int waiting_time;
14    int turnaround_time;
15    int completion_time;
16    int time_quantum;
17 };
18
19 void swap(struct Process *a, struct Process *b)
20 {
21     struct Process temp = *a;
22     *a = *b;
23     *b = temp;
24 }
25
26 void accept_array(struct Process processes[], int number_of_processes)
27 {
28     for (int i = 0; i < number_of_processes; i++)
29     {
30         printf("Enter the Process id : ");
31         scanf("%d", &processes[i].process_id);
32         printf("\nEnter the Arrival Time: ");
33         scanf("%d", &processes[i].arrival_time);
34         printf("Enter the Burst Times: ");
35         scanf("%d", &processes[i].burst_time);
36         printf("\n");
37         // assign others to 0
38     }
39 }
40
41 void display(struct Process processes[], int number_of_processes, int i)
42 {
43
44     printf("
45     -----
46     n");
47     printf("|\\tID\\tBur\\tArr\\tser\\twait\\tCom\\tTAT\\n");
48     for (; i < number_of_processes; i++)
49     {
50         printf("|\\t%d\\t%d\\t%d\\t%d\\t%d\\t%d\\t%d\\n", processes[i].process_id,
51         processes[i].burst_time, processes[i].arrival_time, processes[i].service_time,
52         processes[i].waiting_time, processes[i].completion_time, processes[i].
53         turnaround_time);
54     }
55     printf("
56     -----
57     n");
58 }
```

```
-----  
    n");  
51 }  
52  
53 void displayGanttChartRR(struct Process processes[], int process_id, int cpu_time)  
54 {  
55     printf("|%d|\t", cpu_time);  
56     printf("%d\t", processes[process_id].process_id);  
57 }  
58  
59 void displayGanttChartFSFS(struct Process processes[], int number_of_processes)  
60 {  
61     for (int i = 0; i < number_of_processes * 4; i++)  
62     {  
63         printf("===");  
64     }  
65     printf("\n");  
66  
67     for (int i = 0; i < number_of_processes; i++)  
68     {  
69         printf("| \tP%d\t", processes[i].process_id);  
70     }  
71     printf("\n");  
72  
73     for (int i = 0; i < number_of_processes * 4; i++)  
74     {  
75         printf("----");  
76     }  
77     printf("\n");  
78  
79     for (int i = 0; i < number_of_processes; i++)  
80     {  
81         printf("|%d|\t\t", processes[i].service_time);  
82     }  
83     printf("|%d|\n", processes[number_of_processes - 1].completion_time);  
84     for (int i = 0; i < number_of_processes * 4; i++)  
85     {  
86         printf("===");  
87     }  
88     printf("\n");  
89 }  
90  
91 int insertion_sort(struct Process processes[], int number_of_processes)  
92 {  
93     int j, key;  
94     for (int i = 0; i < number_of_processes; i++)  
95     {  
96         key = processes[i].arrival_time;  
97         j = i - 1;  
98         while (j >= 0 && processes[j].arrival_time > key)  
99         {  
100             swap(&processes[j + 1], &processes[j]);  
101             j--;  
102         }  
103     }  
104 }  
105  
106 void npfcfs(struct Process processes[], int number_of_processes)  
107 {
```

```
108     for (int i = 0; i < number_of_processes; i++)
109     {
110         processes[i].completion_time = 0;
111         processes[i].waiting_time = 0;
112         processes[i].service_time = 0;
113         processes[i].turnaround_time = 0;
114     }
115     processes[0].service_time = processes[0].arrival_time;
116     processes[0].waiting_time = 0;
117
118     for (int i = 1; i < number_of_processes; i++)
119     {
120         processes[i].service_time = processes[i - 1].burst_time + processes[i -
121 1].service_time;
122     }
123     for (int i = 1; i < number_of_processes; i++)
124     {
125         for (int j = 0; j < i; j++)
126         {
127             processes[i].waiting_time += processes[j].burst_time;
128         }
129         processes[i].waiting_time -= processes[i].arrival_time;
130     }
131     for (int i = 0; i < number_of_processes; i++)
132     {
133         processes[i].completion_time = processes[i].service_time + processes[i].
134         burst_time;
135     }
136     for (int i = 0; i < number_of_processes; i++)
137     {
138         processes[i].turnaround_time = processes[i].completion_time - processes[i
139 ].arrival_time;
140     }
141 }
142
143 int find_size_of_ready_queue(struct Process processes[], int number_of_processes,
144 int time_quantum)
145 {
146     int n = 0;
147     for (int i = 0; i < number_of_processes; i++)
148     {
149         n += processes[i].burst_time / time_quantum;
150         if (processes[i].burst_time % time_quantum != 0 && processes[i].burst_time
151 > time_quantum)
152             n++;
153     }
154     // printf("%d", n);
155     return n;
156 }
157
158 void round_robin(struct Process processes[], int number_of_processes, int
159 time_quantum)
160 {
161     int process_counter = 0;
162     // Safely assigning every value we have to calculate to zero.
163     for (int i = 0; i < number_of_processes; i++)
164     {
165         processes[i].completion_time = 0;
166         processes[i].waiting_time = 0;
```

```
161     processes[i].service_time = 0;
162     processes[i].turnaround_time = 0;
163 }
164 // This is like a pre defined thing.
165 processes[process_counter].service_time = processes[process_counter].
arrival_time;
166
167 // we need to put things in the ready queue, so we need its size. This
function calculates that.
168 int ready_queue_size = find_size_of_ready_queue(processes, number_of_processes
, time_quantum);
169 int ready_queue_counter = 0;
170 int current_node = 0;
171 int cpu_time = 0; // to count the cpu clock cycles.
172
173 int ready_queue[ready_queue_size]; // creating the ready queue.
174
175 // Assigning values of that ready queue to -1.
176 for (int i = 0; i < ready_queue_size; i++)
177 {
178     ready_queue[i] = -1;
179 }
180
181 // Let us push the first process to the ready queue. This is the one that has
arrived first as that list is sorted.
182 ready_queue[current_node] = process_counter;
183
184 // Creating a copy of the burst times array, as those values have to change in
preemptive algorithms to keep track of them.
185 int burst_times[number_of_processes];
186 for (int i = 0; i < number_of_processes; i++)
187 {
188     burst_times[i] = processes[i].burst_time;
189 }
190
191 // Drawing the first line of the Gantt Chart
192
193 for (int i = 0; i < ready_queue_size; i++)
194     printf("-----");
195 printf("\n");
196
197 // Drawing the last line of the Gantt Chart
198 // well increment the counter as per as our need, and stop the scheduling when
we are done with the queue.
199 while (current_node <= ready_queue_size)
200 {
201     displayGanttChartRR(processes, ready_queue[current_node], cpu_time);
202
203     int cpu_cycle_begin_state = cpu_time;
204
205     /// Execution of cpu begins
206
207     // increment the cpu time by time quantum only if the burst time of the
current process is greater than the time quantum.
208     if (burst_times[ready_queue[current_node]] >= time_quantum)
209     {
210         cpu_time += time_quantum;
211         burst_times[ready_queue[current_node]] -= time_quantum;
212     }
```

```

213     else
214     {
215         cpu_time += burst_times[ready_queue[current_node]];
216         burst_times[ready_queue[current_node]] = 0;
217     }
218
219     /// Execution complete
220
221     int cpu_cycle_end_state = cpu_time;
222
223     // check if any processes came in the meanwhile when we incremented the
224     cpu_counter
225     for (int i = 0; i < number_of_processes; i++)
226     {
227         // some process arrives during our execution, add to ready_queue
228         if (processes[i].arrival_time > cpu_cycle_begin_state && processes[i].
229         arrival_time <= cpu_cycle_end_state)
230         {
231             ready_queue_counter++;
232             ready_queue[ready_queue_counter] = i;
233             process_counter++;
234         }
235
236         // Check if the current process is done executing its burst cycles, if not
237         then add it to ready queue as well
238         if (burst_times[ready_queue[current_node]] > 0)
239         {
240             ready_queue_counter++;
241             ready_queue[ready_queue_counter] = ready_queue[current_node];
242         }
243
244         // If its done, then calculate everything related to that processes.
245         else if (burst_times[ready_queue[current_node]] == 0)
246         {
247             processes[ready_queue[current_node]].completion_time =
248             cpu_cycle_end_state;
249             processes[ready_queue[current_node]].turnaround_time = processes[
250             ready_queue[current_node]].completion_time -
251             processes[
252             ready_queue[current_node]].arrival_time;
253             processes[ready_queue[current_node]].waiting_time = processes[
254             ready_queue[current_node]].turnaround_time -
255             processes[
256             ready_queue[current_node]].burst_time;
257         }
258
259         // finally increment the current node to point to the next part of the
260         ready queue counter
261         current_node++;
262     }
263
264     // Print the last line of gantt chart
265     printf("|%d|", cpu_time);
266     printf("\n");
267     for (int i = 0; i < ready_queue_size; i++)
268         printf("-----");
269     printf("\n\n");
270 }

```

```
263 float calc_average_waiting_time(struct Process processes[], float
    number_of_processes)
264 {
265     float a = 0;
266     for (int i = 0; i < number_of_processes; i++)
267     {
268         a += processes[i].waiting_time;
269     }
270     printf("%d", a);
271
272     a /= number_of_processes;
273     printf("%f", a);
274
275     return a;
276 }
277
278 float calc_average_tat(struct Process processes[], float number_of_processes)
279 {
280     float a = 0;
281     for (int i = 0; i < number_of_processes; i++)
282     {
283         a += processes[i].turnaround_time;
284     }
285     a /= number_of_processes;
286     return a;
287 }
288
289 int main()
290 {
291     int number_of_processes = 0, time_quantum = 2;
292     float average_waiting_time, average_tat;
293     // printf("How many Processes do you wanna input? \n\n");
294     // scanf("%d", &number_of_processes);
295
296     // if (number_of_processes == 0)
297     // {
298     //     printf("You do not have any processes\n");
299     //     return 0;
300     // }
301     number_of_processes = 4;
302     struct Process processes[4] =
303     {
304         {1, 7, 0, 0, 0, 0, 0},
305         {2, 4, 2, 0, 0, 0, 0},
306         {3, 1, 4, 0, 0, 0, 0},
307         {4, 4, 5, 0, 0, 0, 0}
308
309         // {1, 8, 0, 0, 0, 0, 0},
310         // {2, 2, 5, 0, 0, 0, 0},
311         // {3, 7, 1, 0, 0, 0, 0},
312         // {4, 3, 6, 0, 0, 0, 0},
313         // {5, 5, 8, 0, 0, 0, 0}
314     };
315
316     // accept_array(processes, number_of_processes);
317     insertion_sort(processes, number_of_processes); // sort arrival times.
318
319     // Display Round Robin
320
```

```

321     printf("Here is the Gantt Chart for Round Robin Scheduling done on the Given
Data:\n\n");
322     round_robin(processes, number_of_processes, time_quantum);
323     display(processes, number_of_processes, 0);
324     average_waiting_time = calc_average_waiting_time(processes,
number_of_processes);
325     average_tat = calc_average_tat(processes, number_of_processes);
326     printf("\nAverage waiting time is: %f", average_waiting_time);
327     printf("\nAverage Turnaround time is: %f\n", average_tat);
328
329     printf("\n\n Here is the Gantt Chart for Non Preemptive First Come First
Server Scheduling done on the Given Data:\n\n");
330
331     npfcfs(processes, number_of_processes);
332     displayGanttChartFSFS(processes, number_of_processes);
333     display(processes, number_of_processes, 0);
334     average_waiting_time = calc_average_waiting_time(processes,
number_of_processes);
335     average_tat = calc_average_tat(processes, number_of_processes);
336     printf("\nAverage waiting time is: %f", average_waiting_time);
337     printf("\nAverage Turnaround time is: %f\n", average_tat);
338 }

```

Listing 1: Assignment 3.Cpp

2 Input and Output

```

1 Input Values In the order:
2     int process_id;
3     int burst_time;
4     int arrival_time;
5     int service_time;
6     int waiting_time;
7     int turnaround_time;
8     int completion_time;
9     int time_quantum;
10
11 {1, 7, 0, 0, 0, 0, 0}
12 {2, 4, 2, 0, 0, 0, 0}
13 {3, 1, 4, 0, 0, 0, 0}
14 {4, 4, 5, 0, 0, 0, 0}
15
16
17 Here is the Gantt Chart for Round Robin Scheduling done on the Given Data:
18
19 -----
20 |0|      1      |2|      2      |4|      1      |6|      3      |7|      2
   |9|      4      |11|     1      |13|     4      |15|     1      |16|
21 -----
22
23 -----
24 |      ID|      Bur|      Arr|      ser|      wait|      Com|      TAT
25 |      1|      7|      0|      0|      9|      16|      16|
26 |      2|      4|      2|      0|      3|      9|      7|
27 |      3|      1|      4|      0|      2|      7|      3|
28 |      4|      4|      5|      0|      6|      15|     10|

```



```
29 -----
30 15.000000
31 Average waiting time is: 5.000000
32 Average Turnaround time is: 9.000000
33
34
35 Here is the Gantt Chart for Non Preemptive First Come First Server Scheduling
   done on the Given Data:
36
37 =====
38 |          P1          ||          P2          ||          P3          ||          P4          |
39 -----
40 |0|                  |7|                  |11|                  |12|                  |16|
41 =====
42 -----
43 |          ID|          Bur|          Arr|          ser|          wait|          Com|          TAT
44 |          1|          7|          0|          0|          0|          7|          7|
45 |          2|          4|          2|          7|          5|          11|          9|
46 |          3|          1|          4|          11|          7|          12|          8|
47 |          4|          4|          5|          12|          7|          16|          11|
48 -----
49 14.750000
50 Average waiting time is: 4.750000
51 Average Turnaround time is: 8.750000
```

Listing 2: Input and Output.Cpp

11/10/20

OS ASSIGNMENT - 3

K. Prichrall PT.
1032210808
PA20 ; (A 1)

Q.1. Explain the need for CPU scheduling.

→ Scheduling of processes / work is done to finish the work on time. CPU scheduling is a process that allows one process to use the CPU while another process is delayed (in standby), due to unavailability of any resources such as I/O etc. thus making full use of the CPU.

The purpose of CPU scheduling is to make the system more efficient; faster and fairer.

→ Whenever the ~~OS~~ ^{CPU} operating system becomes idle, OS must select one of the processes ready for launch, and assigns the CPU to one of them.

→ The selection process is done by a temporary CPU scheduler; the scheduler selects between memory processes ready to launch, and assigns the CPU to one of them.

Q.2.

Explain ~~for~~ Pre-emption and pre-emptive decision mode.

→ ① Non-Preemptive : If the CPU is allocated to the process, then it keeps the CPU until it releases it by terminating or switching to waiting state.

② Pre-emptive : If CPU is allocated to a process, it may be released if high priority process needs the CPU.

Q.3

Explain FCFS and Round Robin with examples.

→ 1. FCFS : First come First served.

→ Precision mode : Non-Preemptive.

→ Selection Function : $\max(t)$ - selects the process that is waiting for the maximum time.

→ Response Times : May be very high, especially if there is a large variation in the process execution times.

→ Overhead : Minimum.

→ Effect on process : Penalizes short processes.

→ Starvation : NO.

Process

Arrival time

Burst time

P₁

0

7

P₂

2

4

P₃

4

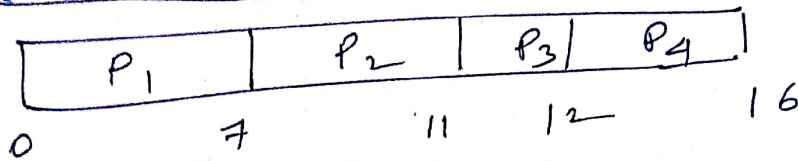
1

P₄

5

4

①

Gantt chart:

②

$$\text{Avg. waiting time} = \frac{(0 + 5 + 7 + 7)}{4}$$

$$= \underline{\underline{4.75}}$$

③

$$\text{Avg. Turnaround Time} = \frac{(7 + 9 + 8 + 11)}{4}$$

$$= \underline{\underline{8.75}}$$

②

Round Robin:

→ Each process gets a small amount of CPU time (time quantum.)

→ After this time has elapsed, the process is preempted and added to the end of the ready time.

→ If there are n processes in the ready queue & the time quantum is q ,

then each process gets $1/n$ of the CPU
in chunks of q units of time at once.

→ No process waits more than $(n-1)q$ time units

- Selection function Constant: constant
- Decision Mode: Pre-emption (at time quantum).
- Response time: provides good response time for good ~~per~~ short processes.
- Overhead: Minimum.

eg. $q = 20$.

Process	Burst Time
P_1	53
P_2	17
P_3	68
P_4	24

$\left(\frac{159}{50}\right)$ v. good

(*)

P_1	P_2	P_3	P_4	P_2	P_3	P_4	P_1	P_3	P_3
0	20	37	57	77	97	117	121	134	154

→ Avg. waiting time: $\frac{(81 + 20 + 94 + 97)}{4} = 78$

→ Avg. Turnaround Time: $\frac{(134 + 37 + 162 + 121)}{4} = 113.5$