

```
1 #include<bits/stdc++.h>
2 using namespace std;
3 struct process {
4
5     pid_t p_no = 0;
6     time_t start_AT = 0, AT = 0,
7         BT_left = 0, BT = 0, temp_BT = 0,
8         CT = 0, TAT = 0, WT = 0, RT = 0;
9     int priority = 0;
10
11     void set_CT(time_t time)
12     {
13         CT = time;
14         set_TAT();
15         set_WT();
16     }
17
18     void set_TAT()
19     {
20         TAT = CT - start_AT;
21     }
22
23     void set_WT()
24     {
25         WT = TAT - BT;
26     }
27     void P_set()
28     {
29         start_AT = AT;
30         BT_left = BT;
31     }
32     void set_RT(time_t time)
33     {
34         RT = time - start_AT;
35     }
36
37
38     friend bool operator<(const process& a, const process& b)
39     {
40         return a.AT > b.AT;
41     }
42 };
43
44 process pop_index(priority_queue<process>* main_queue,
45                 ,
46                 int index)
47 {
48     priority_queue<process> rm_index;
49     int i;
50     process p;
51
52     switch (index) {
53     case 0:
54         p = (*main_queue).top();
55         (*main_queue).pop();
56         break;
57     default:
58         for (i = 0; i < index; i++) {
59             rm_index.push((*main_queue).top());
60             (*main_queue).pop();
61         }
62         p = (*main_queue).top();
63     }
```

```
62     (*main_queue).pop();
63
64     while (!(*main_queue).empty()) {
65         rm_index.push((*main_queue).top());
66         (*main_queue).pop();
67     }
68     (*main_queue) = rm_index;
69     break;
70 }
71 return p;
72 }
73
74
75 int max_priority(priority_queue<process> main_priority_queue,
76                 int limit, bool high)
77 {
78     int max = -1;
79     if (high == 1) {
80         while (!main_priority_queue.empty()
81             && main_priority_queue.top().AT <= limit) {
82             if (main_priority_queue.top().priority > max)
83                 max = main_priority_queue.top().priority;
84             main_priority_queue.pop();
85         }
86     }
87     else {
88         while (!main_priority_queue.empty()
89             && main_priority_queue.top().AT <= limit) {
90             if (max == -1 || main_priority_queue.top().priority < max)
91                 max = main_priority_queue.top().priority;
92             main_priority_queue.pop();
93         }
94     }
95     return max;
96 }
97
98 int max_priority_index(priority_queue<process> main_queue, int limit, bool high)
99 {
100     int max = -1, i = 0, index = 0;
101     if (high == 1) {
102         while (!main_queue.empty() && main_queue.top().AT <= limit) {
103             if (main_queue.top().priority > max) {
104                 max = main_queue.top().priority;
105                 index = i;
106             }
107             main_queue.pop();
108             i++;
109         }
110     }
111     else {
112         while (!main_queue.empty()
113             && main_queue.top().AT <= limit) {
114             if (max == -1 || main_queue.top().priority < max) {
115                 max = main_queue.top().priority;
116                 index = i;
117             }
118             main_queue.pop();
119             i++;
120         }
121     }
122     return index;
123 }
```

```

123 }
124
125 priority_queue<process> Priority_P_run(priority_queue<process> ready_queue,
queue<process>* gantt, bool high)
126 {
127     int temp;
128     priority_queue<process> completion_queue;
129     process p;
130     time_t clock = 0;
131     if (high == 1) {
132         while (!ready_queue.empty()) {
133             while (clock < ready_queue.top().AT) {
134                 p.temp_BT++;
135                 clock++;
136             }
137             if (p.temp_BT > 0) {
138                 p.p_no = -1;
139                 p.CT = clock;
140                 (*gantt).push(p);
141             }
142             p = pop_index(&ready_queue,
143                         max_priority_index(ready_queue, clock, high));
144             if (p.AT == p.start_AT)
145                 p.set_RT(clock);
146             while (p.BT_left > 0
147                 && (ready_queue.empty()
148                     || clock < ready_queue.top().AT
149                     || p.priority >= max_priority(ready_queue, clock, high))) {
150                 p.temp_BT++;
151                 p.BT_left--;
152                 clock++;
153             }
154             if (p.BT_left == 0) {
155                 p.AT = p.start_AT;
156                 p.set_CT(clock);
157                 (*gantt).push(p);
158                 p.temp_BT = 0;
159                 completion_queue.push(p);
160             }
161             else {
162                 p.AT = clock;
163                 p.CT = clock;
164                 (*gantt).push(p);
165                 p.temp_BT = 0;
166                 ready_queue.push(p);
167             }
168         }
169     }
170     else {
171         while (!ready_queue.empty()) {
172             while (clock < ready_queue.top().AT) {
173                 p.temp_BT++;
174                 clock++;
175             }
176             if (p.temp_BT > 0) {
177                 p.p_no = -1;
178                 p.CT = clock;
179                 (*gantt).push(p);
180             }
181             p = pop_index(&ready_queue,
182                         max_priority_index(ready_queue,

```

```

183                                     clock, high));
184
185         if (p.AT == p.start_AT)
186             p.set_RT(clock);
187         temp = max_priority(ready_queue, clock, high);
188
189         while (p.BT_left > 0 && (ready_queue.empty()
190                                 || clock < ready_queue.top().AT
191                                 || p.priority <= max_priority(ready_queue,
clock, high))) {
192             p.temp_BT++;
193             p.BT_left--;
194             clock++;
195         }
196         if (p.BT_left == 0) {
197             p.AT = p.start_AT;
198             p.set_CT(clock);
199             (*gantt).push(p);
200             p.temp_BT = 0;
201             completion_queue.push(p);
202         }
203         else {
204             p.AT = clock;
205             p.CT = clock;
206             (*gantt).push(p);
207             p.temp_BT = 0;
208             ready_queue.push(p);
209         }
210     }
211 }
212
213 return completion_queue;
214 }
215
216 priority_queue<process> set_sample_data()
217 {
218     priority_queue<process> ready_queue;
219     int n;
220     cout<<"\nEnter the number of processes: "; cin>>n;
221
222     for(int i=0;i<n;i++)
223     {
224         cout<<"\nEnter arrival time ,burst time and priority of process "<<i+1<<"
: ";
225         process temp;
226         cin>>temp.AT>>temp.BT>>temp.priority;
227         temp.p_no = i+1;
228         temp.P_set();
229         ready_queue.push(temp);
230     }
231     cout<<"\n";
232
233     return ready_queue;
234 }
235
236
237 double get_total_WT(priority_queue<process> processes)
238 {
239     double total = 0;
240     while (!processes.empty()) {
241         total += processes.top().WT;
242         processes.pop();

```

```

243     }
244     return total;
245 }
246
247 double get_total_TAT(priority_queue<process> processes)
248 {
249     double total = 0;
250     while (!processes.empty()) {
251         total += processes.top().TAT;
252         processes.pop();
253     }
254     return total;
255 }
256
257 double get_total_CT(priority_queue<process> processes)
258 {
259     double total = 0;
260     while (!processes.empty()) {
261         total += processes.top().CT;
262         processes.pop();
263     }
264     return total;
265 }
266
267 double get_total_RT(priority_queue<process> processes)
268 {
269     double total = 0;
270     while (!processes.empty()) {
271         total += processes.top().RT;
272         processes.pop();
273     }
274     return total;
275 }
276
277 void disp(priority_queue<process> main_queue, bool high)
278 {
279     int i = 0, temp, size = main_queue.size();
280     priority_queue<process> tempq = main_queue;
281     double temp1;
282     cout << "+-----+-----+";
283     cout << "+-----+-----+";
284     cout << "+-----+-----+-----+";
285     if (high == true)
286         cout << "-----+" << endl;
287     else
288         cout << endl;
289     cout << "| Process No. | Arrival Time ";
290     cout << "| Burst Time | Completion Time ";
291     cout << "| Turnaround Time | Waiting Time | Response Time |";
292     if (high == true)
293         cout << " Priority |" << endl;
294     else
295         cout << endl;
296     cout << "+-----+-----+";
297     cout << "+-----+-----+";
298     cout << "+-----+-----+-----+";
299     if (high == true)
300         cout << "-----+" << endl;
301     else
302         cout << endl;
303     while (!main_queue.empty()) {

```

```

304     temp = to_string(main_queue.top().p_no).length();
305     cout << '|' << string(6 - temp / 2 - temp % 2, ' ')
306           << main_queue.top().p_no << string(7 - temp / 2, ' ');
307     temp = to_string(main_queue.top().start_AT).length();
308     cout << '|' << string(7 - temp / 2 - temp % 2, ' ')
309           << main_queue.top().start_AT << string(7 - temp / 2, ' ');
310     temp = to_string(main_queue.top().BT).length();
311     cout << '|' << string(6 - temp / 2 - temp % 2, ' ')
312           << main_queue.top().BT << string(6 - temp / 2, ' ');
313     temp = to_string(main_queue.top().CT).length();
314     cout << '|' << string(8 - temp / 2 - temp % 2, ' ')
315           << main_queue.top().CT << string(9 - temp / 2, ' ');
316     temp = to_string(main_queue.top().TAT).length();
317     cout << '|' << string(8 - temp / 2 - temp % 2, ' ')
318           << main_queue.top().TAT << string(9 - temp / 2, ' ');
319     temp = to_string(main_queue.top().WT).length();
320     cout << '|' << string(7 - temp / 2 - temp % 2, ' ')
321           << main_queue.top().WT << string(7 - temp / 2, ' ');
322     temp = to_string(main_queue.top().RT).length();
323     cout << '|' << string(7 - temp / 2 - temp % 2, ' ')
324           << main_queue.top().RT << string(8 - temp / 2, ' ');
325     if (high == true) {
326         temp = to_string(main_queue.top().priority).length();
327         cout << '|' << string(5 - temp / 2 - temp % 2, ' ')
328               << main_queue.top().priority << string(5 - temp / 2, ' ');
329     }
330     cout << "|\n";
331     main_queue.pop();
332 }
333 cout << "+-----+-----+";
334 cout << "+-----+-----+";
335 cout << "+-----+-----+-----+";
336 if (high == true)
337     cout << "-----+";
338 cout << endl;
339 temp1 = get_total_CT(tempq);
340 cout << "\nTotal completion time :- " << temp1 << endl;
341 cout << "Average completion time :- " << temp1 / size << endl;
342 temp1 = get_total_TAT(tempq);
343 cout << "\nTotal turnaround time :- " << temp1 << endl;
344 cout << "Average turnaround time :- " << temp1 / size << endl;
345 temp1 = get_total_WT(tempq);
346 cout << "\nTotal waiting time :- " << temp1 << endl;
347 cout << "Average waiting time :- " << temp1 / size << endl;
348 temp1 = get_total_RT(tempq);
349 cout << "\nTotal response time :- " << temp1 << endl;
350 cout << "Average response time :- " << temp1 / size << endl;
351 }
352
353 void disp_gantt_chart(queue<process> gantt)
354 {
355     int temp, prev = 0;
356     queue<process> spaces = gantt;
357     cout << "\n\nGantt Chart (IS indicates ideal state) :- \n\n";
358     while (!spaces.empty()) {
359         cout << string(to_string(spaces.front().p_no).length() +
360 (spaces.front().p_no != -1) + 2 * spaces.front().temp_BT, '-') << "+";
361         spaces.pop();
362     }
363     cout << "\n|";
364     spaces = gantt;

```

```
364     while (!spaces.empty()) {
365         cout << string(spaces.front().temp_BT, ' ');
366         if (spaces.front().p_no == -1)
367             cout << "IS" << string(spaces.front().temp_BT, ' ') << '|';
368         else
369             cout << "P" << spaces.front().p_no
370                 << string(spaces.front().temp_BT, ' ') << '|';
371         spaces.pop();
372     }
373     spaces = gantt;
374     cout << "\n+";
375     while (!spaces.empty()) {
376         cout << string(to_string(spaces.front().p_no).length() +
377 (spaces.front().p_no != -1) + 2 * spaces.front().temp_BT, '-')
378             << "+";
379         spaces.pop();
380     }
381     spaces = gantt;
382     cout << "\n0";
383     while (!spaces.empty()) {
384         temp = to_string(spaces.front().CT).length();
385         cout << string(to_string(spaces.front().p_no).length() +
386 (spaces.front().p_no != -1) + 2 * spaces.front().temp_BT - temp / 2 - prev, '-')
387             << spaces.front().CT;
388         prev = temp / 2 - temp % 2 == 0;
389         spaces.pop();
390     }
391     cout << "\n\n";
392 }
393
394 int main()
395 {
396     priority_queue<process> ready_queue, completion_queue;
397
398     queue<process> gantt;
399
400     ready_queue = set_sample_data();
401
402     completion_queue = Priority_P_run(ready_queue, &gantt, true);
403
404     disp(completion_queue, true);
405
406     disp_gantt_chart(gantt);
407     return 0;
408 }
```