Preemptive Priority Scheduling

هنگامی که process های مختلف قرار است روی CPU اجرا شود باید برای اینکه کدام یک زودتر و کدام دیرتر اجرا شود برنامه ریزی انجام شود. برای این کار نیاز به الگوریتمی است که این process ها را ترتیب دهی کند. یکی از این الگوریتم ها الگوریتم Priority Scheduling است. این الگوریتم را میتوان به صورت preemptive یا nonpreemptive اجرا کرد که در این پروژه به صورت preemptive پیاده سازی شده است. Process ها وارد میشوند، یک زمان اجرا دارند و هرکدام اولویتی دارند که این مشخصات به الگوریتم داده میشود. هنگامی که process ای با اولویت بالاتر وارد صف میشود باید فرآیندی که درحال اجراست متوقف شود و فرایند با اولویت بالاتر اجرا شود و پس از اتمام آن فرایند های با اولویت پایین تر به کار خود ادامه دهند.

در اینجا به توضیح کد میپردازیم و همچنین برای یک مثال آن را اجرا میکنیم:

```
4   struct process {                    // structure of each process
5       int pid;
6       int arrive_time;
7       int burst_time;
8       int remaining_burst_time;
9       int priority;
10      bool completed;
11      int finishing_time;
12      int turnaround_time;
13      int waiting_time;
14      int respond_time;
15  };
```

در ابتدا ساختار هر process را مشخص کرده ایم. هر فرایند یک pid دارد که شناسه ای است که با آن این فرآیند توسط CPU و ... شناخته میشود. Arrive_time زمان ورود فرایند به صف را نشان میدهد. Burst_time مدت زمانی که اجرای یک cycle فرایند طول میکشد را نمایش میدهد.

Remaining_burst_time نشاندهنده ی مدت زمان باقیمانده از اجرای آن فرایند است. مثلا اگر فرایند 20 ثانیه برای اجرا نیاز داشته باشد و پس از 5 ثانیه توسط فرایندی دیگر متوقف شود remaining_burst_time آن برابر 15 ثانیه است. Priority اولویتی است که به یک فرایند اختصاص دارد. Completed یک متغیر boolean است که صرفا نشان میدهد که آیا اجرای یک فرایند تمام شده است یا هنوز به CPU نیاز دارد. Finishing_time نشان دهنده ی زمانی است که اجرای یک فرایند بطور کامل خاتمه می یابد و remaining_burst_time آن برابر 0 میشود. Turn_around_time برابر فاصله ی بین زمان رسیدن و زمان خاتمه ی یک فرایند است. Waiting_time مجموع کل مدت زمانی است که یک فرایند در انتظار اختصاص CPU است. Respond_time زمانی است که طول میکشد تا یک فرایند برای اولین بار شروع به اجراشدن کند.

```
17  class priority_scheduling {
18      private:
19          int clock;                // stores current time
20          int total_time;           // total time that is needed for termination
21          int finished_processes;   // number of processes that their execution is finished by current time
22          int number_of_processes;  // number of all existing processes
23          struct process p[1000];   // stores all processes
24          int* scheduled_list;      // saves the runtimes
25
26      public:
27          priority_scheduling(int);
28          void get_input();
29          void set_input(int);
30          void show_details();
31          void scheduling();
32          void run(int);
33          void gantt();
34          double calculate_turnaround();
35          double calculate_waiting();
36          double calculate_respond();
37          double calculate_utilization();
38          double calculate_throughput();
39  };
```

در این قسمت یک کلاس برای انجام عملیات روی فرایند ها تعریف کرده ایم. خصوصیات این کلاس بدین صورت است:

Clock : **زمان کنونی سیستم**

Total_time : **مجموع کل زمانی که برای خاتمه ی کل فرایند ها نیاز است**

Finished_processes : **تعداد فرایند های خاتمه یافته**

Number_of_processes : **تعداد کل فرایند های موجود**

P : **آرایه ای از همه ی فرایند های موجود**

Scheduled_list : **لیست نشاندهنده ی ترتیب اجرا طبق الگوریتم**

Priority_scheduling : **تابع** initializer **که مقادیر اولیه را ست میکند**

Get_input : **تابعی که فرایند ها و ویژگی های انها را از ورودی میگیرد**

Set_input : **تابعی که ویژگی های فرایند ها را ذخیره میکند و به انها اختصاص میدهد**

Show_details : **تابعی که فرایند های موجود و ویژگی های انها از جمله مقدار زمان اجرای باقیمانده شان را نشان میدهد.**

Scheduling : **تابع اصلی برای انجام عملیات زمانبندی فرایند ها**

Run : **تابعی که فرایندی که باید اجرا شود را به** CPU **اعلام میکند و مقادیر و ویژگی های فرایند اجرا شده را اپدیت میکند.**

Gantt : **تابعی که نمودار** gantt **فرایند ها را پس از زمانبندی نمایش میدهد.**

Calculate_turnarond : **متوسط زمان** turnaround **فرایند ها را محاسبه میکند.**

Calculate_waiting : **متوسط زمان** waiting **فرایند ها را محاسبه میکند.**

Calculate_respond : **متوسط زمان** respond **فرایند ها را محاسبه میکند.**

Calculate_utilization : **درصد بهره وری** CPU **را محاسبه میکند.**

Calculate_troughput : **تعداد فرایندی که در یک واحد زمانی اجرا میشود را محاسبه میکند.**

```
41  priority_scheduling::priority_scheduling(int n) {    // initializer
42      clock = 0;                          // set the time to 0
43      finished_processes = 0;
44      number_of_processes = n;
45      total_time = 0;
46  }
```

این تابع initializer **کلاس است. در این تابع مقادیر اولیه ست میشود. زمان سیستم به 0 ست میشود. تعداد فرایند های خاتمه یافته و زمان کل اجرا هم مقادیر اولیه ی 0 میگیرند و تعداد کل فرایند ها هم از ورودی گرفته میشوند.**

```
48  void priority_scheduling::get_input() {
49      for (int i=1; i<=number_of_processes; i++) {
50          set_input(i);             // set the properties of each process
51      }
52      for (int i=0; i<number_of_processes; i++) {
53          total_time += p[i].burst_time;
54      }
55      scheduled_list = new int[total_time];
56  }
```

در این تابع برای تعداد کل فرایند های موجود تابع SET **فراخوانی میشود. پس از مقدار دهی کردن ویژگی های فرایند ها مقدار کل زمان اجرا محاسبه میشود و ارایه ی نتیجه برای این مقدار زمان ساخته میشود.**

```
58  void priority_scheduling::set_input(int process) {
59      p[process-1].pid = process;      // set the pid of the process to its number
60      p[process-1].completed = false;
61      cout << "Enter the arrival time of process " << process << " : ";
62      cin >> p[process-1].arrive_time;
63      cout << "Enter the burst time of process " << process << " : ";
64      cin >> p[process-1].burst_time;
65      p[process-1].remaining_burst_time = p[process-1].burst_time;
66      cout << "Enter the priority of process " << process << " : ";
67      cin >> p[process-1].priority;
68  }
```

در این تابع مقدار و ویژگی های هر فرایند ذخیره میشوند. در خط 59، pid فرایند به شماره ی ورود ان فرایند ست میشود. سپس ان را خاتمه نیافته مقدار دهی میکنیم. پس از آن مقادیر زمان رسیدن، زمان انفجار و اولویت آن فرایند را از سیستم ورودی میگیریم و هرکدام را در ساختار مربوطه مقدار دهی میکنیم.

```
70  void priority_scheduling::show_details() {
71      // represent all informations of processes
72      cout << "Arrival time    Burst time    Priority" << endl;
73      for (int i=0; i<number_of_processes; i++) {
74          cout <<  "     " << p[i].arrive_time << "              " << p[i].burst_time << "            " << p[i].priority << endl;
75      }
76  }
```

در این تابع برخی مقادیر و ویژگی های فرایند ها را نمایش میدهیم. این تابع برای راحتی کد زدن است و در الگوریتم بکار نمیرود اما کاربر ممکن است بخواهد همه ی اطلاعات را بدست اورد.

```
77  void priority_scheduling::gantt() {    // print gantt chart for schedulled processes
78      for (int i=0; i<total_time*5; i++) {
79          cout << "_";
80      }
81      cout << endl;
82      int prev = 0;
83      for (int i=0; i<total_time; i++) {
84          if (prev != scheduled_list[i]) {
85              cout << "| P" << scheduled_list[i] << " ";
86              prev = scheduled_list[i];
87          }
88          else {
89              cout << "     ";
90          }
91      }
92      cout << '|' << endl;
93      prev = 0;
94      for (int i=0; i<total_time; i++) {
95          if (scheduled_list[i] != prev) {
96              cout << i << "____";
97              prev = scheduled_list[i];
98          }
99          else {
100             cout << "_____";
101         }
102     }
103     cout << total_time;
104 }
```

در این تابع نمودار gantt برای فرایند های برنامه ریزی شده ی نهایی رسم میشود. در خطوط 78 تا 80 خط بالای جدول نمایش داده میشود. سپس در خطوط 82 تا 92 فرایند انتخابی به نسبت زمان اجرایی که دارد فضای نمودار را به خود اختصاص میدهد. اگر در دو کلاک متوالی یک فرایند اجرا شود از نوشتن مجدد اسم آن خودداری میشود. سپس در خطوط 93 تا 103 خط پایین نمودار نشان داده میشود که در زمان هایی که فرایند اجرایی تغییر کرده است تایم نمایش داده شده است.

```
106  void priority_scheduling::scheduling() {
107      // this is the main function that specifies the process that should be run in current time clock
108      while (finished_processes != number_of_processes) {  // until the time that every processes are finished
109          int selected_process = 0;
110          int max_priority = 10000;
111          for (int i=0; i<number_of_processes; i++) {      // check all processes and find the process with maximum
112                                                           // priority by the current time
113              if (p[i].arrive_time<=clock && !p[i].completed && p[i].priority<max_priority) {
114                  max_priority = p[i].priority;
115                  selected_process = p[i].pid;
116              }
117          }
118          run(selected_process);                  // execute the selected process
119          scheduled_list[clock] = selected_process;
120          clock += 1;                             // go for next time clock
121      }
122
123  }
```

سپس به سراغ تابع اصلی زمانبندی میرویم. این تابع تا زمانی که همه ی فرایند ها خاتمه یابند اجرا میشود تا همه زمانبندی شوند. در حلقه ی for هربار روی همه ی فرایند های موجود حرکت میکنیم. سپس برای هر فرایند چک میکنیم. اگر به زمان رسیدن ان فرایند نرسیده باشیم یعنی هنوز ان فرایند وارد صف انتخاب نشده است پس از ان چشم پوشی کرده و به جست و جو ادامه میدهیم. اگر زمان رسیدن آن فرارسیده باشد اما فرایند پیش از ان خاتمه یافته باشد هم ان فرایند را در نظر نمیگیریم. سپس به سراغ شرط اخر میرویم. تنها درصورتی فرایند جدید جایگزین فرایند قبلی میشود که اولویت آن از قبلی بالاتر باشد. در غیر اینصورت کاری انجام نمیدهد. اما اگر فرایند عوض شود در نهایت فرایند خاتمه نیافته ی موجود با بیشترین اولویت انتخاب میشود و به تابع run داده میشود تا عملیات های لازم را روی آن اجرا کند. سپس برای آن بازه زمانی فرایند انتخاب شده را در لیست نهایی درج میکنیم. و در آخر زمان سیستم 1 واحد افزایش می یابد که به سراغ مرحله ی بعد میرود. از این روش چک کردن در هر واحد زمانی برای اجرای شرط preemptive بودن استفاده میشود. چون بین بازه های زمانی هیچ فرایند جدیدی وارد نمیشود.

```
126 void priority_scheduling::run(int process) {    // execute the process for current time clock and update properties
127     cout << "Process " << process << " is executing..." << endl;
128     int index = process - 1;            // location of the process in the array of processes
129     if (p[index].remaining_burst_time == p[index].burst_time) {
130         p[index].respond_time = clock - p[index].arrive_time;
131     }
132     p[index].remaining_burst_time -= 1;
133     if (p[index].remaining_burst_time == 0) {
134         p[index].completed = true;
135         finished_processes += 1;
136         p[index].finishing_time = clock + 1;
137         p[index].turnaround_time = p[index].finishing_time - p[index].arrive_time;
138         p[index].waiting_time = p[index].turnaround_time - p[index].burst_time;
139     }
140     cout << "Finished. remaining time: " << p[index].remaining_burst_time << " completed or not: " << p[index].completed << endl;
141 }
```

در این قسمت تابع run را توضیح میدهیم. این تابع به صورت واقعی قرار نیست فرایند را اجرا کند چون اجرا کار CPU است و مستقل از این الگوریتم زمانبندی است اما پس از اجرای هر فرایند لازم است عملیاتی روی آن انجام شود که این عملیات ها در این تابع هندل میشوند. ابتدا به کاربر نشان میدهد که کدام فرایند هم اکنون در حال اجراست. سپس اندیس اصلی فرایند را با توجه به pid آن در صف بدست می آورد. در خطوط 128 تا 130 چک میکند که اگر فرایند برای بار اول است که دارد اجرا میشود زمان پاسخ آن را به زمان حاضر مقدار دهی کند. سپس چون در این واحد زمانی این فرایند اجرا شده است یک واحد از زمان انفجار آن کم میکند. سپس در خطوط 132 تا 138 چک میشود که اگر فرایند پس از این بازه خاتمه یافته است متغیر completed آن را برابر true قرار میدهیم و تعداد فرایند های خاتمه یافته را 1 عدد اضافه میکنیم و زمان پایان آن فرایند را به کلاک بعدی ست میکنیم. در این قسمت میتوان turnaround time و waiting time نهایی را نیز برای فرایند خاتمه یافته مقدار دهی کرد. سپس به کاربر اعلام میکنیم که این بازه زمان با اجرای فرایند x خاتمه یافت و از آن y مقدار دیگر باقیمانده است و آیا تمام شده است یا خیر.

```
142  double priority_scheduling::calculate_turnaround() {
143      double sum = 0.0;
144      for (int i=0; i<number_of_processes; i++) {
145          sum += p[i].turnaround_time;
146      }
147      return sum / number_of_processes;
148  }
```

در این تابع برای همه ی فرایند ها مقدار turnaround time **ای که هنگام خاتمه فرایند ست شده است را باهم جمع میکنیم و میانگین گیری را انجام میدهیم.**

```
150  double priority_scheduling::calculate_waiting() {
151      double sum = 0.0;
152      for (int i=0; i<number_of_processes; i++) {
153          sum += p[i].waiting_time;
154      }
155      return sum / number_of_processes;
156  }
```

در این تابع برای همه ی فرایند ها مقدار waiting time **ای که هنگام خاتمه فرایند ست شده است را باهم جمع میکنیم و میانگین گیری را انجام میدهیم.**

```
158   double priority_scheduling::calculate_respond() {
159       double sum = 0.0;
160       for (int i=0; i<number_of_processes; i++) {
161           sum += p[i].respond_time;
162       }
163       return sum / number_of_processes;
164   }
```

در این تابع برای همه ی فرایند ها مقدار respond time **ای که هنگام شروع اجرای فرایند ست شده است را باهم جمع میکنیم و میانگین گیری را انجام میدهیم.**

```
166   double priority_scheduling::calculate_utilization() {
167
168       return (total_time / clock) * 100;
169   }
```

در این تابع بهره وری CPU **را محاسبه می کنیم. بهره وری به مقدار زمانی که** CPU **در حال انجام اجرای فرایند ها بوده در طول زمانی که تا خاتمه ی همه ی فرایند ها است گویند. در متغیر** total_time **مجموع زمان انفجار همه ی فرایند ها را ذخیره کرده ایم. سپس تقسیم بر زمان میکنیم و درصد گیری میکنیم.**

```
171  double priority_scheduling::calculate_throughput() {
172      int latest_finished = 0, earliest_arrive = 10000;
173      for (int i=0; i<number_of_processes; i++) {
174          if (p[i].finishing_time > latest_finished) {
175              latest_finished = p[i].finishing_time;
176          }
177          if (p[i].arrive_time < earliest_arrive) {
178              earliest_arrive = p[i].arrive_time;
179          }
180      }
181      double throughput = double(number_of_processes) / (latest_finished - earliest_arrive);
182      return throughput;
183  }
```

در این تابع throughput را محاسبه میکنیم. باید مشخص کنیم در یک واحد زمانی چند فرایند اجرا شده اند. این موضوع کاملا به فرایند ها و زمان اجرای آنها بستگی دارد. هرچقدر زمان اجرا کوتاه باشد این عدد بزرگتر خواهد شد. برای محاسبه از بین همه ی فرایند های اجرا شده اولین زمان رسیدن و اخرین زمان تمام شدن را پیدا میکنیم. سپس تعداد کل فرایند ها را بر بازه ی رسیدن اولین فرایند تا اتمام اخرین فرایند تقسیم میکنیم.

```
185  int main() {
186      int n;
187      cout << "Enter the number of total processes: ";
188      cin >> n;
189      priority_scheduling obj(n);
190      obj.get_input();
191      obj.show_details();
192      obj.scheduling();
193      cout << "Average TurnAround time: " << obj.calculate_turnaround() << endl;
194      cout << "Average Waiting time: " << obj.calculate_waiting() << endl;
195      cout << "Average Respond time: " << obj.calculate_respond() << endl;
196      cout << "CPU utilization: " << obj.calculate_utilization() << "%" << endl;
197      cout << "Throughput: " << obj.calculate_throughput() << endl;
198      obj.gantt();
199  }
```

در اینجا تابع main دیده میشود. تعداد کل فرایند ها از ورودی گرفته میشود. سپس آبجکتی از کلاس زمانبندی ساخته میشود که کل عملیات ها روی آن انجام میشود. ساختن کلاس برای این است که میتوان آبجکت های متفاوتی ساخت و زمانبندی را روی صف فرایند های مختلفی به طور همزمان اجرا کرد. سپس فرایند ها را برای آن ابجکت ورودی میگیریم و مقدار دهی میکنیم. پس از آن اطلاعات فرایند های داده شده را نمایش میدهیم و سپس زمانبندی را روی آنها انجام میدهیم. بعد از آن میانگین turn around time, waiting time, respond time و بهره وری CPU و بازده را محاسبه کرده و نمایش میدهیم. در آخر هم نمودار گانت را برای زمانبندی انجام شده نمایش میدهیم.

مثال:

گرفتن ورودی(main):

```
Enter the number of total processes: 3
Enter the arrival time of process 1 : 0
Enter the burst time of process 1 : 2
Enter the priority of process 1 : 3
Enter the arrival time of process 2 : 1
Enter the burst time of process 2 : 4
Enter the priority of process 2 : 2
Enter the arrival time of process 3 : 5
Enter the burst time of process 3 : 1
Enter the priority of process 3 : 1
```

نمایش ویژگی ها(show_details):

Arrival time	Burst time	Priority
0	2	3
1	4	2
5	1	1

اجرای فرایند انتخابی(run):

```
Process 1 is executing...
Finished. remaining time: 1 completed or not: 0
Process 2 is executing...
Finished. remaining time: 3 completed or not: 0
Process 2 is executing...
Finished. remaining time: 2 completed or not: 0
Process 2 is executing...
Finished. remaining time: 1 completed or not: 0
Process 2 is executing...
Finished. remaining time: 0 completed or not: 1
Process 3 is executing...
Finished. remaining time: 0 completed or not: 1
Process 1 is executing...
Finished. remaining time: 0 completed or not: 1
```

نمایش نتایج توابع محاسبات:

```
Average TurnAround time: 4
Average Waiting time: 1.66667
Average Respond time: 0
CPU utilization: 100%
Throughput: 0.428571
```

نمایش نمودار گانت(gantt):