گزارش دستورکار هشتم آزمایشگاه سیستم‌های عامل

نگار موقتیان، ۹۸۳۱۰۶۲

پیاده‌سازی الگوریتم FCFS

در این قسمت از آزمایش می‌خواهیم الگوریتم FCFS را برای زمانبندی تعدادی پردازه پیاده‌سازی کنیم.

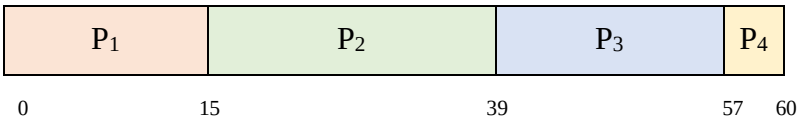
برای این کار مطابق دستور کار ابتدا تعداد پردازه‌ها و سپس burst time متناظر با هر یک دریافت شده‌است. سپس دقیقاً به ترتیبی که پردازه‌ها به عنوان ورودی دریافت شده‌اند اجرا خواهند شد. به این صورت مقدار waiting time و turnaround time برای پردازهٔ اول به ترتیب برابر با صفر و burst time پردازهٔ اول بوده و برای پردازه‌های بعدی برابر با turnaround time پردازهٔ قبل و مجموع waiting time و burst time پردازهٔ فعلی می‌باشد.

در نهایت مقدار میانگین waiting time و turnaround time برای تمام پردازه‌ها محاسبه شده‌است.

خروجی برنامهٔ فوق مانند زیر می‌باشد.

```
E:\Projects\DevC++\OS Lab\OSLab08_FCFS.exe
Number of processes: 4

Please enter the 'burst time' corresponding to each process:
15 24 18 3

Process 1:
  Burst Time = 15 - Waiting Time = 0 - Turnaround Time = 15

Process 2:
  Burst Time = 24 - Waiting Time = 15 - Turnaround Time = 39

Process 3:
  Burst Time = 18 - Waiting Time = 39 - Turnaround Time = 57

Process 4:
  Burst Time = 3 - Waiting Time = 57 - Turnaround Time = 60


Average Waiting Time: 27.75
Average Turnaround Time: 42.75
```

که با Gantt Chart پردازه‌ها که برای الگوریتم FCFS مانند زیر خواهد بود تطابق دارد.



پیاده‌سازی الگوریتم SJF

در این قسمت از آزمایش می‌خواهیم الگوریتم SJF را برای زمانبندی تعدادی پردازه پیاده‌سازی کنیم.

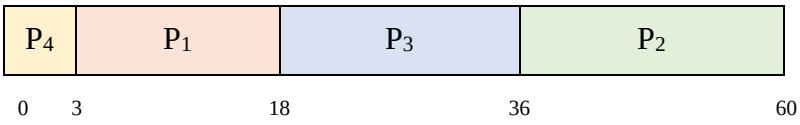
برای این کار مطابق دستور کار ابتدا تعداد پردازه‌ها و سپس burst time متناظر با هر یک دریافت شده‌است. سپس با استفاده از تابع sort پردازه‌ها با استفاده از الگوریتم bubble sort و بر حسب burst time شان مرتب شده‌اند و بر همین اساس به ترتیب اجرا خواهند شد. پس از مرتب‌سازی پردازه‌ها مقدار waiting time و turnaround time برای هر یک از آن‌ها دقیقاً مانند الگوریتم FCFS محاسبه می‌شود.

در نهایت مقدار میانگین waiting time و turnaround time برای تمام پردازه‌ها محاسبه شده‌است.

خروجی برنامهٔ فوق مانند زیر می‌باشد.

```
E:\Projects\DevC++\OS Lab\OSLab08_SJF.exe
Number of processes: 4

Please enter the 'burst time' corresponding to each process:
15 24 18 3

Process 4:
  Burst Time = 3 - Waiting Time = 0 - Turnaround Time = 3

Process 1:
  Burst Time = 15 - Waiting Time = 3 - Turnaround Time = 18

Process 3:
  Burst Time = 18 - Waiting Time = 18 - Turnaround Time = 36

Process 2:
  Burst Time = 24 - Waiting Time = 36 - Turnaround Time = 60


Average Waiting Time: 14.25
Average Turnaround Time: 29.25
```

که با Gantt Chart پردازه‌ها که برای الگوریتم SJF مانند زیر خواهد بود تطابق دارد.



پیاده‌سازی الگوریتم مبتنی بر الویت

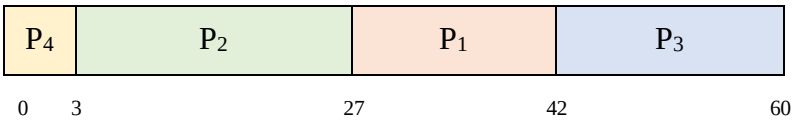
در این قسمت از آزمایش می‌خواهیم الگوریتمی مبتنی بر الویت را برای زمانبندی تعدادی پردازه پیاده‌سازی کنیم.

برای این کار یک فیلد pri که نشان‌دهندهٔ الویت پردازه است به struct process اضافه شده‌است تا بتوانیم الویت پردازه را نیز ذخیره‌سازی کنیم. ادامهٔ برنامه دقیقاً مشابه الگوریتم SJF می‌باشد، با این تفاوت که پردازه‌ها بر اساس الویتشان (و نه burst time آن‌ها) و به صورت نزولی مرتب سازی می‌شوند (در این برنامه فرض شده‌است که هر چه عدد الویت بالاتر باشد پردازه الویت بیش‌تری خواهد داشت).

خروجی برنامهٔ فوق مانند زیر می‌باشد.

```
E:\Projects\DevC++\OS Lab\OSLab08_Priority.exe
Number of processes: 4

Please enter the 'burst time' and 'priority' corresponding to each process:
15 4
24 7
18 1
3 10

Process 4:
  Burst Time = 3 - Priority = 10 - Waiting Time = 0 - Turnaround Time = 3

Process 2:
  Burst Time = 24 - Priority = 7 - Waiting Time = 3 - Turnaround Time = 27

Process 1:
  Burst Time = 15 - Priority = 4 - Waiting Time = 27 - Turnaround Time = 42

Process 3:
  Burst Time = 18 - Priority = 1 - Waiting Time = 42 - Turnaround Time = 60


Average Waiting Time: 18.00
Average Turnaround Time: 33.00
```

که با Gantt Chart پردازه‌ها که برای الگوریتم مبتنی بر الویت مانند زیر خواهد بود تطابق دارد.



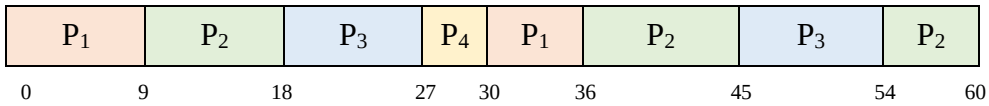
پیاده‌سازی الگوریتم RR

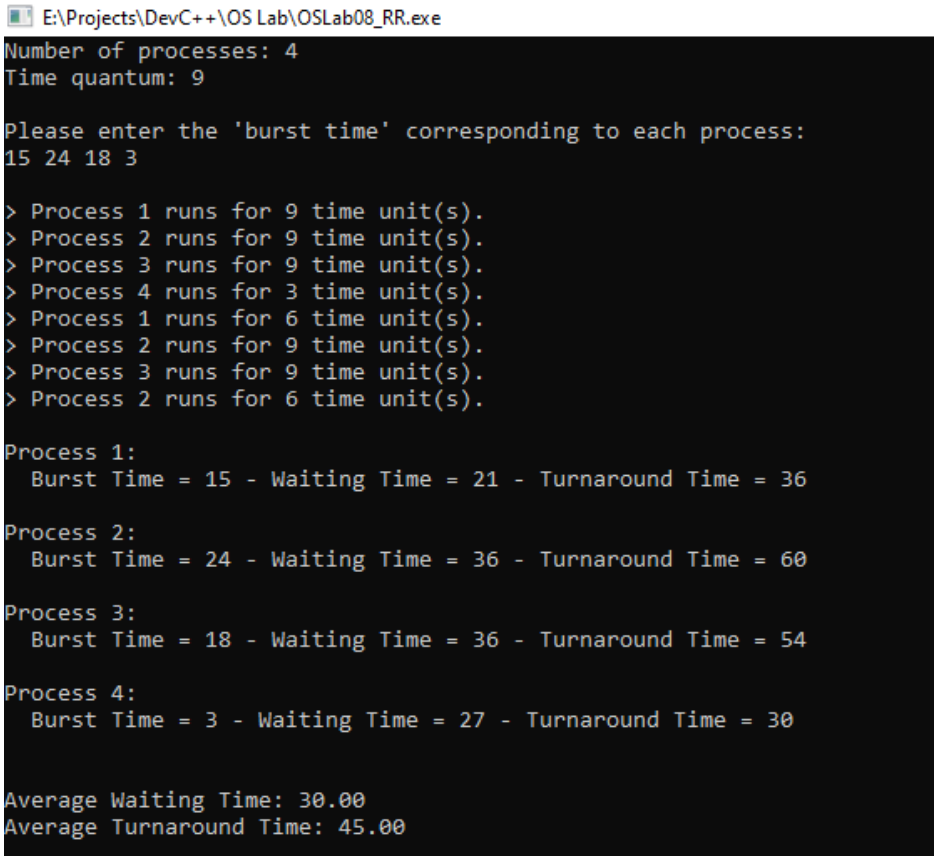
در این قسمت از آزمایش می‌خواهیم الگوریتم RR را برای زمانبندی تعدادی پردازه پیاده‌سازی کنیم.

برای این کار مطابق دستور کار ابتدا تعداد پردازه‌ها، time quantum و سپس burst time متناظر با هر یک از پردازه‌ها دریافت شده‌است. در این برنامه از یک آرایهٔ remainingJob استفاده شده که مقدار کاری که از هر یک از پردازه‌ها باقی مانده را نگهداری می‌کند. در ادامه یک حلقهٔ while با شرط flag داریم. flag مشخص می‌کند آیا پردازه‌ای در سیستم باقی مانده که به اتمام نرسیده باشد یا خیر. سپس با استفاده از متغیرهای curr و index به صورت چرخشی بر روی لیست پردازه‌ها پیمایش می‌کنیم تا پردازه‌ای بیابیم که مقداری از کار آن باقی مانده. در صورتی که چنین پردازه‌ای وجود داشت آن را به اندازهٔ مینیمم time quantum و مقدار کار باقی‌مانده اجرا می‌کنیم، به زمان فعلی اضافه کرده و آرایهٔ remainingJob را آپدیت می‌کنیم. پس از آن بررسی می‌کنیم که آیا کار این پردازه به اتمام رسیده یا خیر (در حقیقت remainingJob آن صفر شده یا خیر) و در صورتی که به اتمام رسیده بود waiting time و turnaround time آن را بر حسب زمان فعلی تنظیم می‌کنیم.

در نهایت نیز مقدار میانگین waiting time و turnaround time برای تمام پردازه‌ها محاسبه شده‌است.

خروجی برنامهٔ فوق مانند زیر می‌باشد که با Gantt Chart پردازه‌ها که برای الگوریتم RR و time quantum برابر با ۹ مانند زیر خواهد بود تطابق دارد.





مقایسهٔ الگوریتم‌ها

با توجه به نتایج قبل برای الگوریتم‌های استفاده شده می‌توان نوشت:

Algorithm	Average Waiting Time	Average Turnaround Time
FCFS	27.75	42.75
SJF	14.25	29.25
Priority Based	18	33
RR	30	40

به طور کلی الگوریتم FCFS برای زمانی مناسب است که پردازه‌هایی با burst time کوتاه و نسبتاً یکنواخت داریم. اگر طول پردازه‌ها بسیار متفاوت باشد این خطر وجود دارد که پردازه‌های کوتاه مجبور شوند برای اجرا منتظر

پردازه‌های طولانی مانده و در نتیجه اثر کاروان به وجود آمده و متوسط زمان اجرای برنامه‌ها زیاد شود (همانطور که مشاهده می‌شود در این جدول نیز ماکسیمم turnaround time را برای این الگوریتم داریم). در عوض پیاده-سازی این الگوریتم ساده است.

برای الگوریتم SJF در مقابل اثبات می‌شود که می‌توان کمترین میانگین زمان انتظار را داشت و این الگوریتم از این نظر به خوبی عمل می‌کند (این اتفاق در جدول بالا نیز مشهود است). اما مشکل این الگوریتم این است که با این روش ممکن است پردازه‌های طولانی دچار قحطی شده و پردازنده به آن‌ها اختصاص نیابد.

الگوریتم مبتنی بر الویت بر خلاف الگوریتم‌های دیگر الویت پردازه‌ها را نیز در نظر می‌گیرد و از این جهت برای سیستم‌های معمول (که در آن‌ها کارها الویت‌های متفاوتی دارند) برتری دارد. اما این روش نیز خطر قحطی برای پردازه‌های کم الویت را در پی دارد.

الگوریتم RR نیز از این جهت اهمیت دارد که با چرخش نوبت میان پردازه‌ها تضمین می‌کند هیچ یک از آن‌ها دچار قحطی نخواهند شد. در عوض عملکرد این الگوریتم (از نظر میانگین زمان انتظار و اجرای پردازه‌ها) ممکن است به خوبی الگوریتم‌هایی مانند SJF نباشد.