**گزارش دستورکار هشتم آزمایشگاه سیستم­های عامل**

نگار موقتیان، 9831062

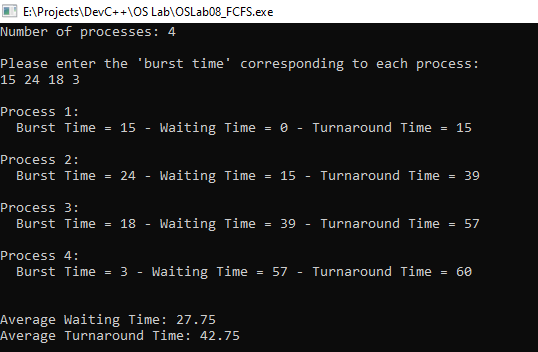
**پیاده­سازی الگوریتم FCFS**

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 دقیقاٌ به ترتیبی که پردازه­ها به عنوان ورودی دریافت شده­اند اجرا خواهند شد. به این صورت مقدار waiting time و turnaround time برای پردازۀ اول به ترتیب برابر با صفر و burst time پردازۀ اول بوده و برای پردازه­های بعدی برابر با turnaround time پردازۀ قبل و مجموع waiting time و burst time پردازۀ فعلی می­باشد.

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

خروجی برنامۀ فوق مانند زیر می­باشد.



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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | P4 | P3 | P2 | P1 |
| 0 15 39 57 60 | | | | |

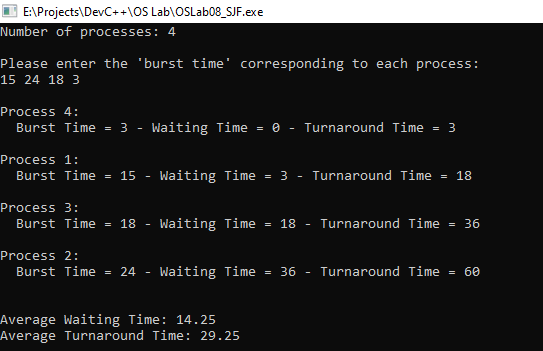
**پیاده­سازی الگوریتم SJF**

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 برنامۀ فوق مانند زیر می­باشد.



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

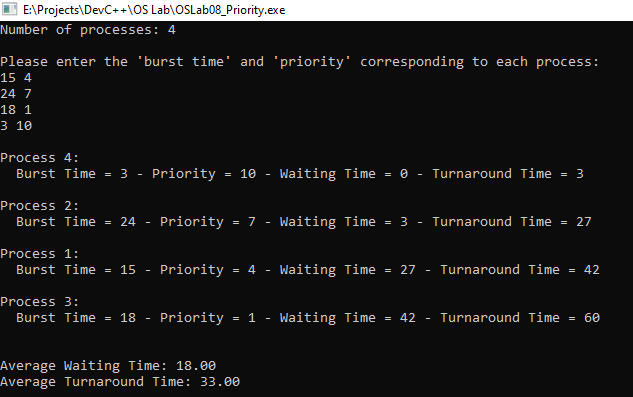
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | P2 | P3 | P1 | P4 |
| 0 3 18 36 60 | | | | |

**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 نشان­دهندۀ الویت پردازه است به struct process اضافه شده­است تا بتوانیم الویت پردازه را نیز ذخیره­سازی کنیم. ادامۀ 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 برنامۀ فوق مانند زیر می­باشد.



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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | P3 | P1 | P2 | P4 |
| 0 3 27 42 60 | | | | |

**پیاده­سازی الگوریتم RR**

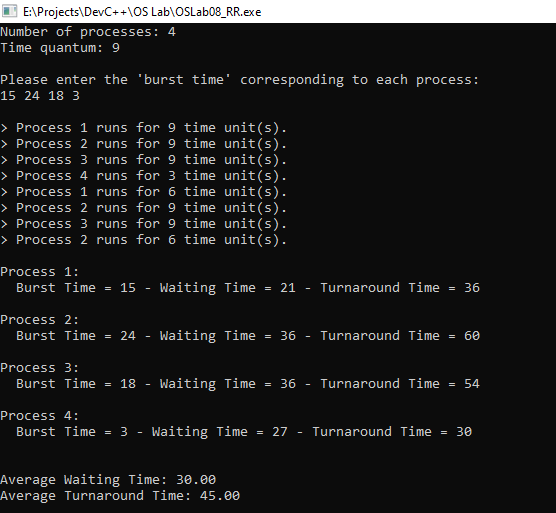
در این قسمت از آزمایش می­خواهیم الگوریتم RR را برای زمانبندی تعدادی پردازه پیاده­سازی کنیم.

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 آرایۀ remainingJob استفاده شده که مقدار کاری که از هر یک از پردازه­ها باقی مانده­ را نگهداری می­کند. در ادامه یک حلقۀ 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 اندازۀ مینیمم time quantum و مقدار کار باقی­مانده اجرا می­کنیم، به زمان فعلی اضافه کرده و آرایۀ 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 برنامۀ فوق مانند زیر می­باشد که با Gantt Chart پردازه­ها که برای الگوریتم RR و time quantum برابر با 9 مانند زیر خواهد بود تطابق دارد.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | P2 | P3 | P2 | P1 | P4 | P3 | P2 | P1 |
| 0 9 18 27 30 36 45 54 60 | | | | | | | | |



**مقایسۀ الگوریتم­ها**

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

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

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 پیاده­سازی 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.