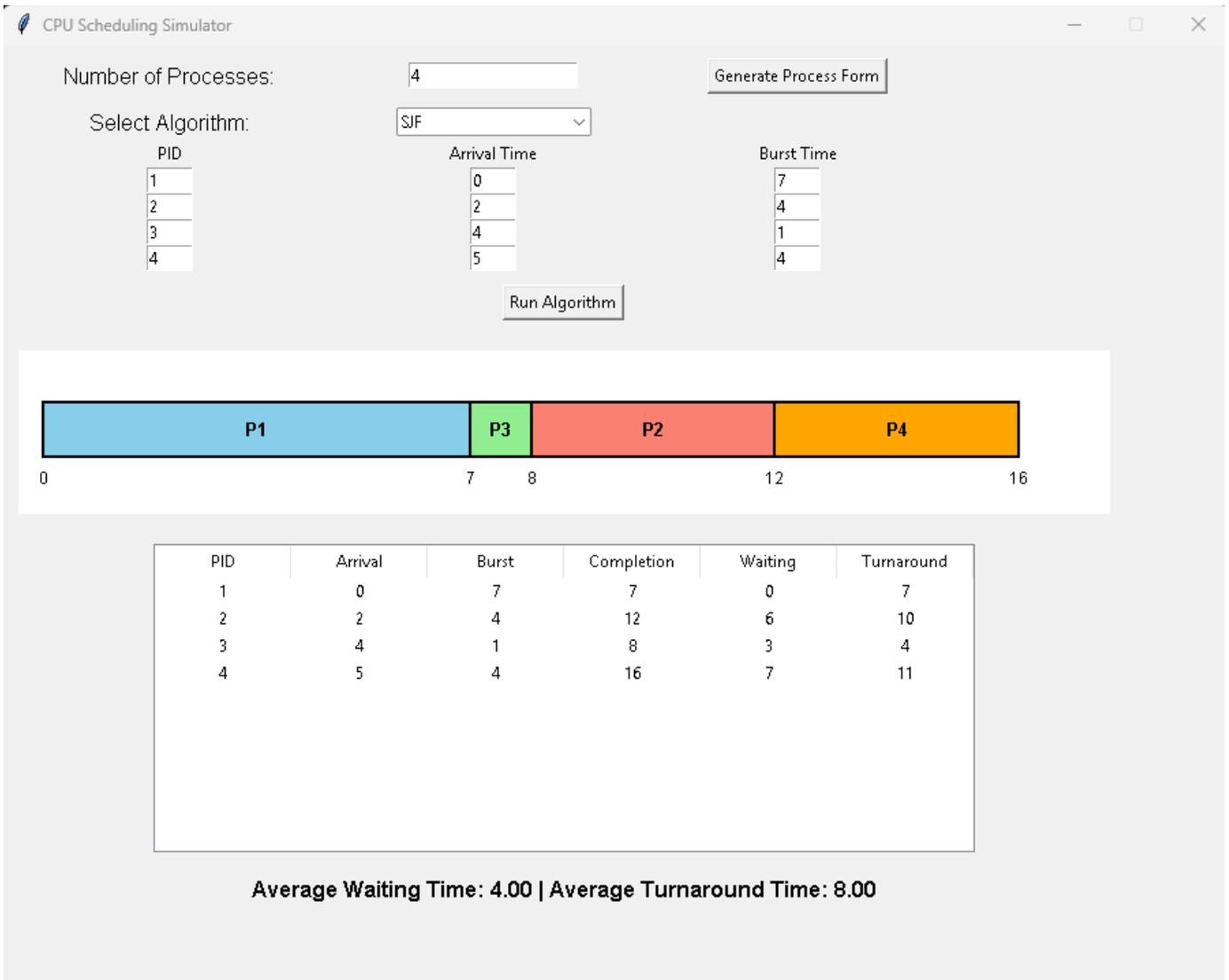


1) تجلیل و مقایسه نتایج الگوریتم‌های SJF و SRTF:

:SJF

الگوریتم SJF به منظور کاهش میانگین زمان انتظار طراحی شده و فرآیند با کوتاه‌ترین زمان اجرا را انتخاب می‌کند. فرآیندها ابتدا بر اساس زمان ورود و در صورت تساوی بر اساس Burst Time مرتب شدند. صف آماده (ready queue) برای نگهداری فرآیندهای رسیده ایجاد گردید و در هر مرحله تصمیم‌گیری، این صف بر مبنای کوتاه‌ترین Burst Time مرتب شد. فرآیند انتخاب‌شده به طور کامل اجرا گردید و معیارهای عملکرد محاسبه شدند. این روش شبیه‌سازی دقیق تصمیم‌گیری پویا را فراهم آورد و از وقفه غیرضروری جلوگیری کرد .

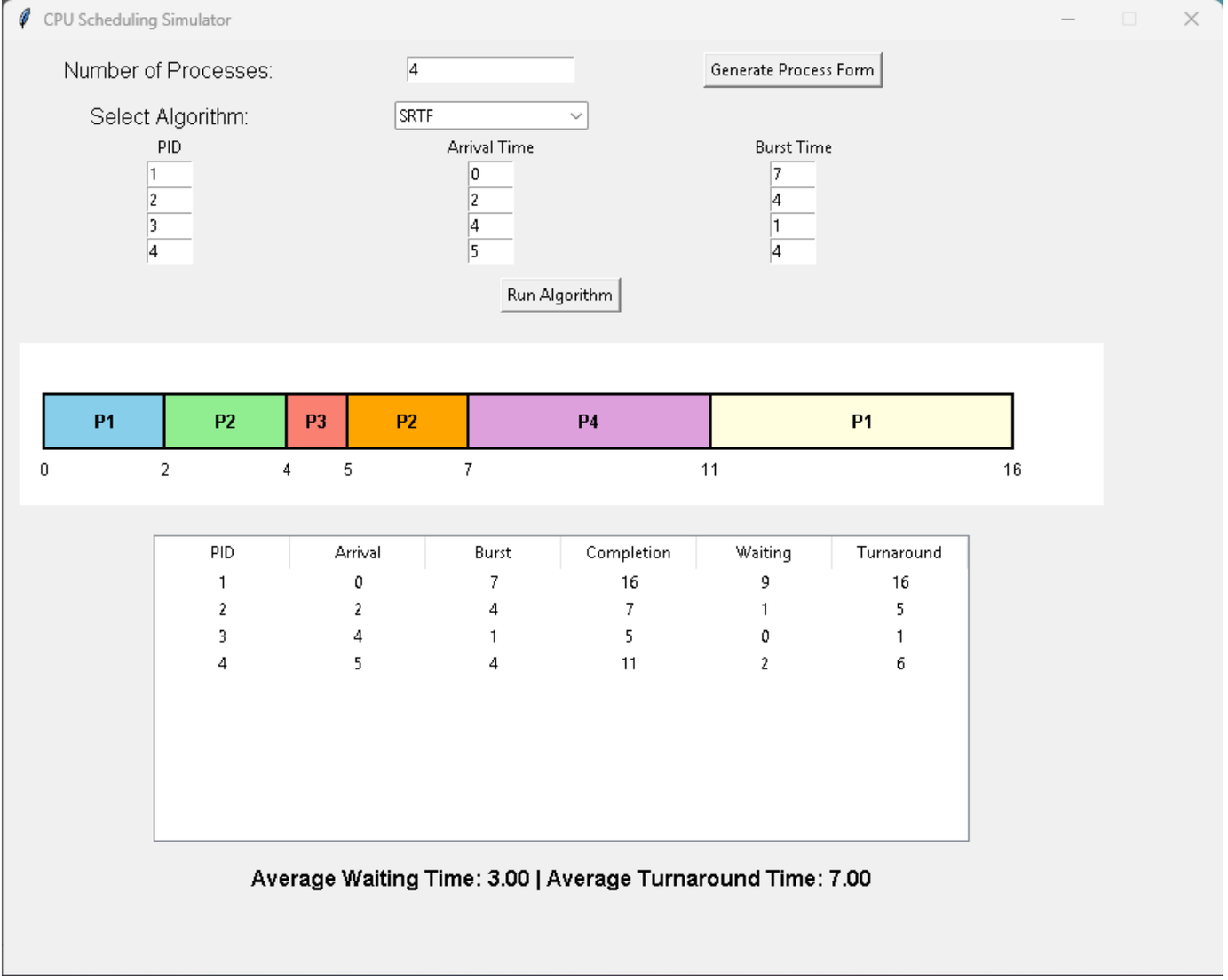
:SRTF

این الگوریتم نسخه قابل‌وقفه SJF بوده و بهینه‌ترین میانگین زمان انتظار را ارائه می‌دهد. زمان به صورت واحد به واحد پیش برده شد و دیکشنری برای نگهداری زمان باقی‌مانده هر فرآیند تعریف گردید. در هر واحد زمانی، فرآیندهای رسیده به صف آماده افزوده شدند و فرآیند با کوتاه‌ترین زمان باقی‌مانده انتخاب گردید. در صورت تغییر فرآیند در حال اجرا، پیش‌قطعی انجام و قطعه جدید در نمودار گانت آغاز شد. این رویکرد شبیه‌سازی دقیق پیش‌قطعی را ممکن ساخت و امکان محاسبه صحیح معیارها را در زمان تکمیل فرآیند فراهم آورد.

تفاوت‌های کلیدی بین SJF وSRTF

ماهیت الگوریتم SJF :غیرقابل‌وقفه است، به این معنا که پس از انتخاب و شروع یک فرآیند، آن را تا پایان اجرا می‌کند، حتی اگر فرآیندهای کوتاه‌تری در حین اجرا وارد شوند. در مقابل، SRTF قابل وقفه است و در هر واحد زمانی، فرآیند با کوتاه‌ترین زمان باقی‌مانده را بررسی و انتخاب می‌کند. این ویژگی اجازه می‌دهد تا فرآیندهای طولانی‌تر برای فرآیندهای کوتاه‌تر ورودی، پیش‌قطعی شوند.

[1]



تصمیم‌گیری در SJF :

تصمیم‌گیری تنها در لحظات پایان فرآیندها یا ورود فرآیندهای جدید ،هنگام خالی بودنCPU ، انجام می‌شود و بر اساس زمان اجرای کامل فرآیند است. اما درSRTF ، تصمیم‌گیری پویا و مداوم است و بر اساس زمان باقی‌مانده (Remaining Time) محاسبه می‌شود، که این امر بهینه‌سازی بیشتری را امکان‌پذیر می‌سازد.

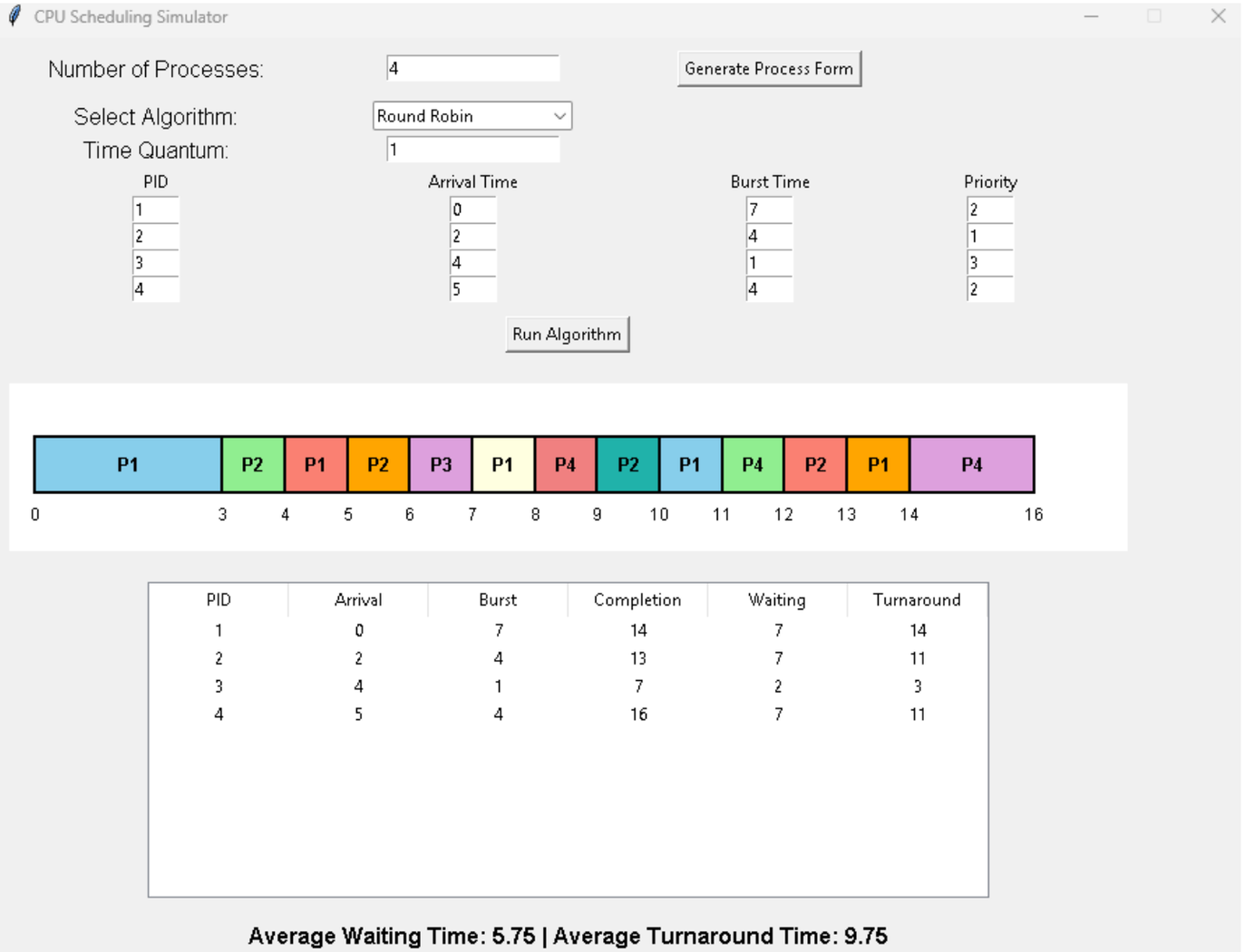
تعداد Context Switches:

SJF معمولاً سوئیچ‌های کمتری دارد (در این تستکیس، ۳ سوئیچ: P1به P3 ، P2به P3 ، P4به P2) ، به دلیل اینکه فرآیندها کامل اجرا می‌شوند .SRTF سوئیچ‌های بیشتری دارد (در این مورد، ۴ سوئیچ: P1 به P2 ، P3به P2 ، P2به P3 ، P4به P2 ، P1به P4) زیرا پیش‌قطعی‌های مکرر رخ می‌دهد.

توجه شود که در بخش مربوطه به هر کدام از تستکیس‌ها برای هر هر مورد توضیحات در خصوص اینکه چه فرایندهایی در الگوریتم رخ می‌دهند آورده شده. با توجه به توضیحاتی که قبلا در خصوص متدولوژی هر الگوریتم آورده شده، به دلایل اصلی تفاوت در نتایج هر الگوریتم می‌پردازیم.

با توجه به نتایج، الگوریتم SRTF میانگین زمان انتظار را به اندازه یک واحد زمانی (از ۴ به ۳) کاهش می‌دهد چرا که پیش‌قطعی P1 در زمان‌های ۲ و ۴ (برای P2 و P3) اجازه می‌دهد فرایندهایی کوتاه مثل P3 بلافاصله اجرا شوند. این امر با اصل کاهش واریانس زمان اجرا در سیستم‌های عامل همخوانی دارد و از پدیده Convoy Effect (که در این مثال در الگوریتم SJF فرایندهای کوتاه پشت فرایند طولانی منتظر می‌مانند) جلوگیری می‌کنند. با این حال، P1 در SRTF زمان انتظار بالاتری دارد (۹ واحد در مقابل ۰)، که می‌تواند به Starvation فرایندهای طولانی منتج شود. SJF ساده‌تر است و سوئیچ‌های کمتری دارد. ولی این الگوریتم زمانی مشکل‌ساز خواهد شد که فضای پردازشی ما پویا باشد. در صورت پویا شدن فضای پردازشی مشاهده می‌کنیم که الگوریتم SJF با افزایش تعداد و تنوع فرایندها غیر بهینه عمل می‌کند.

[2]



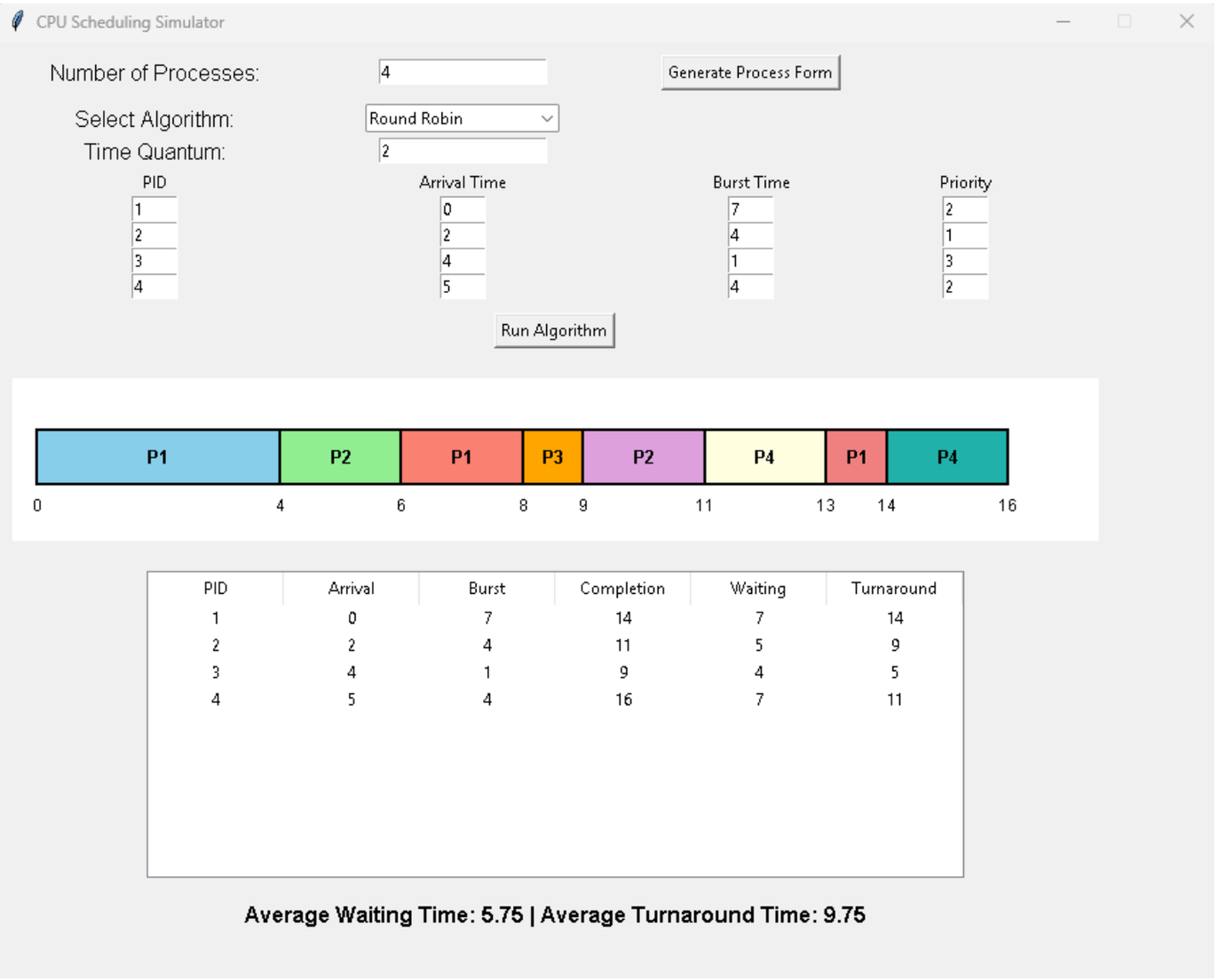
۲) حلیل الگوریتم Round Rubin با چند مقدار مختلف کوانتوم و تأثیر آن بر Context Switch و Average Waiting Time:

الگوریتم Round Rubin یکی از پرکاربردترین الگوریتم‌های زمان‌بندی CPU در سیستم‌های عامل مدرن به حساب می‌آید بخصوص در محیط‌های Time-Sharing و multi-tasking. این الگوریتم از نوع قابل وقفه (preemptive) بوده و بر پایه مفهوم time quantum یا time slice عمل می‌کند. در Round Rubin فرایندهای آماده در یک صف چرخشی نگهداری می‌شوند. هنگامی که نوبت یک فرایند می‌رسد، آن فرایند حداکثر به مدت کوانتوم اجرا می‌شود. اگر زمان اجرای باقی‌مانده فرایند کمتر از کوانتوم باشد، فرایند بطور کامل تکمیل شده و از صف حذف می‌شود. در غیر این صورت پس از اتمام کوانتوم، فرایندپیش‌قطعی شده و به انتهای صف بازگردانده می‌شود تا در دور بعدی فرصت اجرای مجدد پیدا کند. این ماکنیسم چرخشی تضمین می‌کند که هر فرایند بطور منصفانه و بدون Starvation به CPU دسترسی داشته باشد.

یکی از ویژگی‌های کلیدی Round Rubin، عدالت بالا در تخصیص منابع است. هیچ فرایندی نمی‌تواند CPU را برای مدت طولانی انحصاری کند که این امر پاسخدهی سریع سیستم را به ویژه برای برنامه‌های تعاملی بهبود می‌بخشد. با این حال، عملکرد اگلوریتم RR به شدت به اندازه کوانتوم بستگی دارد. کوانتوم کوچک باعث Context Switches زیاد شده و Overhead بالا می‌خواهد. با این‌حال، کوانتوم کوچک، پاسخدهی بهتری ارائه می‌دهد. برعکس، کوانتوم بزرگ تعداد سوئیچ‌ها را کاهشد داده و به FCFS نزدیک‌تر می‌شود که ممکن است دچار پدیده‌ی Convoy Effect شویم.

در مقایسه با الگوریتم‌های دیگر، Round Rubin میانگین زمان انتظاری بالاتری نسبت‌به الگوریتم‌هایی مانند SRTF دارد. اما برتری اصلی آن در عدالت و جلوگیری از انحصار منابع است. به همین دلیل، در سیستم‌های عامل امروزی برای پردازش‌های چند سطحی استفاده می‌شود.

[3]



برای تستکیس‌های Round Rubin، ۴ مورد تست انتخاب شدند که در آنها همه شرایط بجز متغیر مقدار کوانتوم، ثابت درنظر گرفته شده‌اند. در تست کیس‌ها کوانتوم‌هایی با مقدار ۱، ۲، ۳ و ۸ درنظر گرفته شده اند. ملاحضه می‌شود که به دلیل اینکه بیشینه‌ی Burst time در مجموعه‌ی ۴ پردازشی ما، به مقدار ۷ واحد می‌باشد، نیازی به بالا بردن میزان کوانتوم از ۸ نخواهیم داشت. چرا که تغییری در نتیجه‌ی کار به نمایش نمی‌آید.

در تستکیس با شماره کوانتوم ۱، مشاهده می‌کنیم که تا زمانی که برنامه با اولویت بالاتری وارد فضای محاسباتی نشود، پردازنده در بافرهای (واحدهای) ۱ کوانتومی به پردازش یک برنامه می‌پردازد. این موضوع همانطور که در توضیح این الگوریتم آمده است به دلیل تخصیص عادلانه‌ی پردازنده به برنامه است. در تستکیس اول مشاهده می‌کنیم که میانگین زمان انتظاری ۵.۷۵ واحد و میانگین Turnaround ها ۹.۷۵ می‌باشد. این مقادیر در مقایسه با زمانی که کوانتوم مقدار ۲ به خود می‌گیرد محسوس نیست. در حالی که اگر به گراف دقت کنیم، تفاوت میان این دو کوانتوم را در تعداد سوئیچ‌ها می‌یابیم. طبیعتاً با افزایش میزان کوانتوم انتظار سوئیچ‌های کمتری را خواهیم داشت. با اینحال شرایط خاص تستکیس باعث شده که میزان Average Waiting Time برای این دو مورد مشابه گزارش شود.

اما تفاوت اصلی در میزان Average Waiting Time و Context Switches را از مورد سوم و مقایسه‌ی آن با دو مورد قبلی دریافت می‌کنیم. بر اساس آنچه در خلاصه‌ی الگوریتم بیان شد، انتظار می‌رود با افزایش کوانتوم، میانگین زمان انتظار افزایش یافته و تعداد سوئیچ کاهش یابد. حال با ملاحضه‌ی مورد سوم مشاهده می‌کنیم که میانگین زمان انتظاری به اندازه ۸.۶٪ افزایش یافته و تعداد سوئیچ‌ها نیز کاهش پیدا کرده است.

این نتایج برای مورد چهارم نیز برقرار است و زمان انتظاری بطور میانگین افزایش یافته و تعداد سوئیچ‌ها با افزایش کوانتوم کاهش می‌یابد .

[4]

CPU Scheduling Simulator

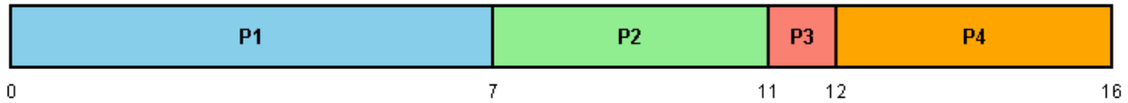
Number of Processes: 4 Generate Process Form

Select Algorithm: Round Robin

Time Quantum: 8

PID	Arrival Time	Burst Time	Priority
1	0	7	2
2	2	4	1
3	4	1	3
4	5	4	2

Run Algorithm



PID	Arrival	Burst	Completion	Waiting	Turnaround
1	0	7	7	0	7
2	2	4	11	5	9
3	4	1	12	7	8
4	5	4	16	7	11

Average Waiting Time: 4.75 | Average Turnaround Time: 8.75

CPU Scheduling Simulator

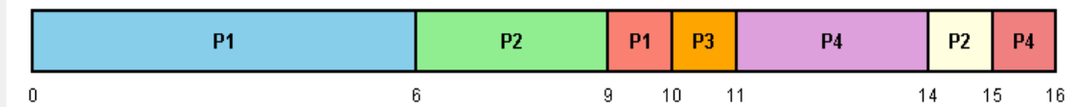
Number of Processes: 4 Generate Process Form

Select Algorithm: Round Robin

Time Quantum: 3

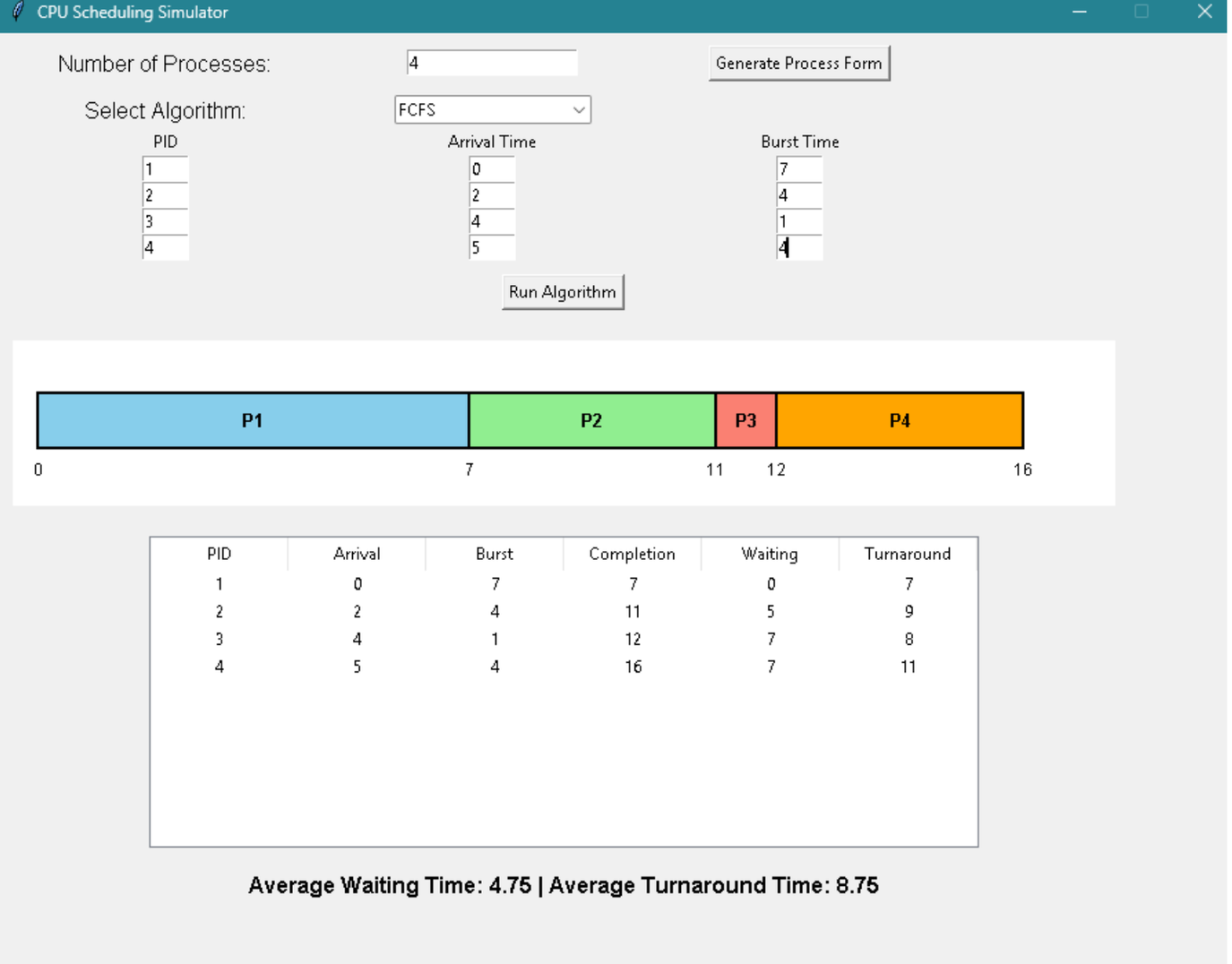
PID	Arrival Time	Burst Time	Priority
1	0	7	2
2	2	4	1
3	4	1	3
4	5	4	2

Run Algorithm



PID	Arrival	Burst	Completion	Waiting	Turnaround
1	0	7	10	3	10
2	2	4	15	9	13
3	4	1	11	6	7
4	5	4	16	7	11

Average Waiting Time: 6.25 | Average Turnaround Time: 10.25



۳) مقایسه‌ی FCFS با Round Rubin در حالتی که کوانتوم بزرگ درنظر گرفته شود؟

الگوریتم First Come First Served (FCFS) یکی از ساده‌ترین و پایه‌ای‌ترین روش‌های زمان‌بندی CPU در سیستم‌های عامل است که بر اساس ترتیب ورود فرآیندها عمل می‌کند. در این الگوریتم، فرآیندها به ترتیب زمانی که وارد صف آماده می‌شوند، اجرا می‌گردند و هیچ اولویتی بر اساس زمان اجرا یا سایر معیارها اعمال نمی‌شود .FCFS غیرقابل‌وقفه (non-preemptive) است، به این معنا که پس از تخصیص CPU به یک فرآیند، آن فرآیند تا پایان زمان اجرای خود (burst time) ادامه می‌یابد و نمی‌توان آن را قطع کرد. این روش شبیه به صف انتظار در دنیای واقعی است و پیاده‌سازی آن آسان می‌باشد، اما معایبی مانند پدیده convoy effect دارد؛ جایی که فرآیندهای کوتاه پشت فرآیند طولانی منتظر می‌مانند و میانگین زمان انتظار (average waiting time) افزایش می‌یابد. از دیدگاه تئوری سیستم‌های عامل، FCFSعدالت را در ترتیب ورود حفظ می‌کند، اما در محیط‌های پویا با فرآیندهای متنوع، کارایی پایینی دارد و ممکن است به starvation فرآیندهای کوتاه منجر شود. در نهایت، این الگوریتم برای سیستم‌های batch مناسب است، اما در سیستم‌های تعاملی کمتر کارآمد می‌باشد.

هنگامی که کوانتوم در الگوریتم Round Robin (RR) به مقداری بالا (برای مثال ۸ یا بالاتر در تست‌کیس‌ها) تنظیم شود، رفتار این الگوریتم به طور قابل توجهی به FCFS نزدیک می‌شود RR .یک روش قابل‌وقفه است که فرآیندها را به صورت چرخشی و با کوانتوم ثابت اجرا می‌کند، اما با کوانتوم بزرگ، پیش‌قطعی رخ نمی‌دهد و هر فرآیند مانند FCFS تا پایان اجرا می‌شود. در تست‌کیس‌ها مشاهده می‌کنیم که عیناً زمان میانگین انتظارها و سوئیچ‌ها برابر هستند. ولی یک نکته مهم وجود دارد و تفاوت این دو مورد است. تفاوت اصلی در پتانسیل عدالت است؛ RR حتی با کوانتوم بالا، امکان پاسخ‌دهی به ورودهای جدید را حفظ می‌کند، در حالی که FCFS کاملاً بر ترتیب ورود وابسته است و ممکن است convoy effect ایجاد کند. از منظر عملکرد، میانگین زمان چرخش (turnaround time)و انتظار در هر دو مشابه است، اما RR overhead بیشتری به دلیل بررسی کوانتوم دارد. در سیستم‌های عامل، RR با کوانتوم بالا برای محیط‌های batch مناسب است، اما FCFS ساده‌تر و بدون overheadاضافی می‌باشد. در نهایت، انتخاب بین آن‌ها بستگی به نیاز به عدالت پویا دارد، جایی که RR انعطاف‌پذیرتر عمل می‌کند.

4) مقایسه‌ی نتایج دو الگوریتم Non-Preemptive و preemptive priority-based priority based و تحلیل نحوه‌ی رسیدگی‌انها به فرایندهای با اولویت‌های مختلف به ویژه تحلیل تعداد سوئیچ‌ها و میانگین زمان انتظار:

الگوریتم‌های زمان‌بندی مبتنی بر اولویت (Priority Scheduling) از جمله روش‌های مهم در سیستم‌های عامل هستند که تصمیم‌گیری تخصیص CPU را بر اساس مقدار اولویت اختصاص‌یافته به هر فرآیند انجام می‌دهند. در این پروژه، اولویت عددی بالاتر به معنای اهمیت بیشتر فرآیند است. دو نسخه اصلی این الگوریتم، یعنی نسخه غیرقابل‌وقفه (Non-Preemptive) و نسخه قابل‌وقفه(Preemptive) ، تفاوت‌های اساسی در نحوه رسیدگی به فرآیندهای با اولویت‌های مختلف، تعداد سوئیچ‌های زمینه و میانگین زمان انتظار دارند.

در الگوریتمNon-Preemptive Priority-Based ، پس از تخصیص CPU به یک فرآیند، آن فرآیند تا پایان زمان اجرای خود ادامه می‌یابد، حتی اگر در حین اجرا فرآیند با اولویت بالاتری وارد شود. انتخاب فرآیند بعدی تنها در زمان‌های خالی شدن CPU (پایان فرآیند جاری یا ورود فرآیندهای جدید) انجام می‌شود. این ویژگی باعث سادگی پیاده‌سازی و کاهش تعداد سوئیچ‌های زمینه می‌گردد، اما ممکن است فرآیندهای با اولویت بالا برای مدت طولانی پشت فرآیندهای طولانی‌تر با اولویت پایین‌تر منتظر بمانند.

در مقابل، الگوریتم Preemptive Priority-Based در هر واحد زمانی، وضعیت صف آماده را بررسی کرده و اگر فرآیند با اولویت بالاتری وارد شود، فرآیند جاری را بلافاصله پیش‌قطع (preempt) کرده و CPU را به فرآیند مهمتر تخصیص می‌دهد. این مکانیسم پاسخ‌دهی سریع‌تری به فرآیندهای حیاتی فراهم می‌آورد و در سیستم‌های real-time بسیار مفید است.

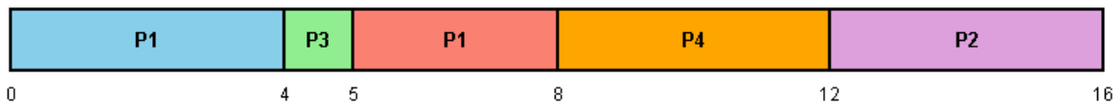
CPU Scheduling Simulator

Number of Processes: 4 Generate Process Form

Select Algorithm: Priority Preemptive

PID	Arrival Time	Burst Time	Priority
1	0	7	2
2	2	4	1
3	4	1	3
4	5	4	2

Run Algorithm



PID	Arrival	Burst	Priority	Completion	Waiting	Turnaround
1	0	7	2	8	1	8
2	2	4	1	16	10	14
3	4	1	3	5	0	1
4	5	4	2	12	3	7

Average Waiting Time: 3.50 | Average Turnaround Time: 7.50

[8]