مریم سعیدمهر
ش.د. :۹۶۲۹۳۷۳
استاد مربوطه : دکتر زالی

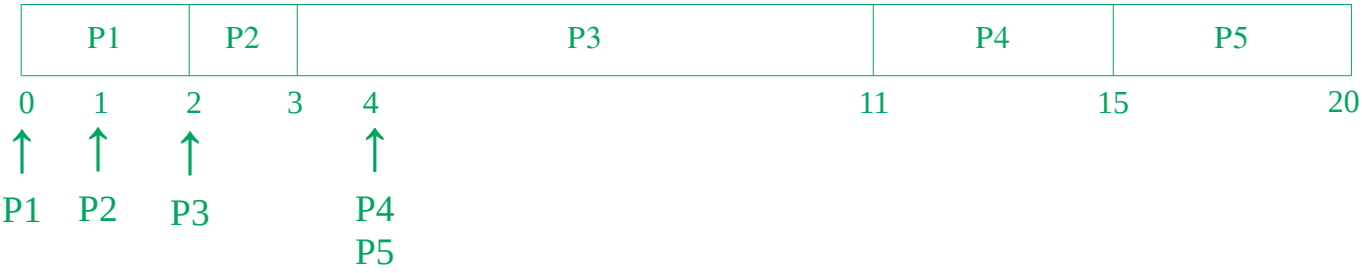
تکلیف سری چهارم سیستم عامل
دانشگاه صنعتی اصفهان
ترم اول سال تحصیلی ۹۸ – ۹۹

---

۱. پروسس های زیر را در نظر بگیرید .جدول گانت را برای الگوریتم های RR , FCFS , SJF , SRF با کوانتوم ۲ رسم کنید ومقادیر متوسط زمان بازگشت و متوسط زمان انتظار را برای هر یک محاسبه کنید .

0rocess	Arrival Time	Burst Time
P1	0	2
P2	1	1
P3	2	8
P4	4	4
P5	4	5

: FCFS

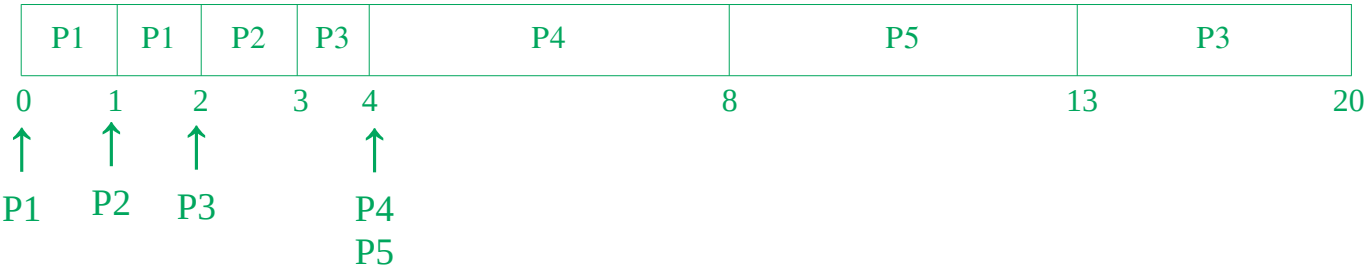


: *FCFS*

$$average\ waiting\ time = \frac{0+(2-1)+(3-2)+(11-4)+(15-4)}{5} = 4$$

$$align laverage\ turn\ arround = \frac{(2-0)+(3-1)+(11-2)+(15-4)+(20-4)}{5} = 8$$

: SRF



	t=0	t=1	t=2	t=3	t=4	t=8	t=13
P1	2	1	0	0	0	0	0
P2	X	1	1	0	0	0	0
P3	X	X	8	8	7	7	7
P4	X	X	X	X	4	0	0
P5	X	X	X	X	5	5	0

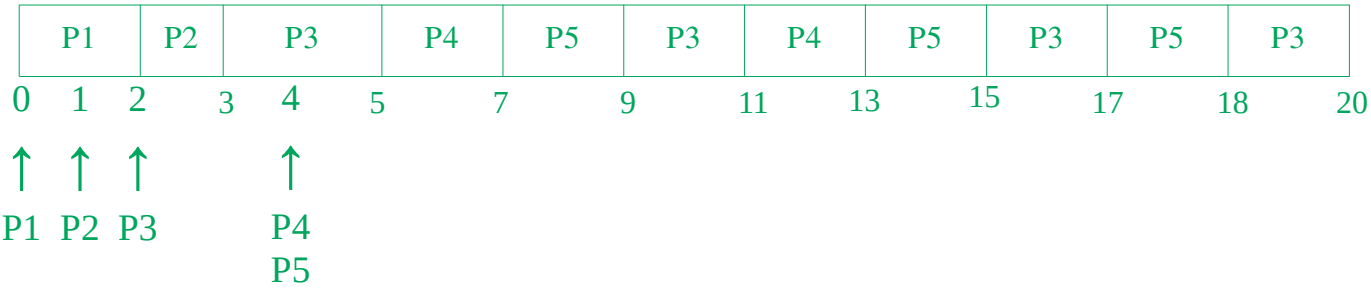
:SRF

$$average\ waiting\ time = \frac{0+(2-1)+(3-2)+(13-4)+(4-4)+(8-4)}{5} = 3$$

$$alignlaverage\ turn\ arround = \frac{(2-0)+(3-1)+(20-2)+(8-4)+(13-4)}{5} = 7$$

الگوریتم SJF دقیقا مشابه FCFS است

: RR , q=2



:RR,q=2

$$average\ waiting\ time = \frac{0+(2-1)+(3-2+9-5+15-11+18-17)+(5-4+11-7)+(7-4+13-9+17-15)}{5} = 5$$

$$alignlaverage\ turn\ arround = \frac{2+2+(20-2)+(13-4)+(18-4)}{5} = 9$$

۲. یک مدل زمان بندی Round Robin بدین صورت کار میکند :

این زمان بند به هر پروسس ، یک کوانتوم زمانی و یک اولویت اختصاص میدهد .مقدار کوانتوم اولیه برای ۵۰ میلی ثانیه است ، با این حال اگر به یک پروسس ، CPU تخصیص داده شود و آن پروسس از تمام کوانتوم زمانی اختصاص داده شده استفاده کند(در بین اجرا به دلیل نیاز به I/O بلاک نشود) و هنوز به پایان نرسیده باشد ، ۱۰ میلی ثانیه به کوانتوم زمانی آن اضافه میشود و اولویت آن ارتقا میابد(اولویتش بالاتر میرود) افزایش زمان کوانتوم تا ۱۰۰ میلی ثانیه ادامه میابد و اولویتش تغییری نمیکند.

پروسس I/O-Bound از این روش سود بیشتری میبرد یا CPU-Bound ؟ توضیح دهید.

پروسس CPU-Bound از این مدل بیشتر سود میبرد زیرا CPU-Burst آن بیشتر از I/O-Burst آن است در نتیجه بیشتر تمایل دارد در CPU بماندتا کامل یا حداکثر اجرایش را انجام بدهد. به علاوه مرتب کوانتوم در حال افزایش است یعنی به پروسس CPU بیشتری اختصاص میدهد به علاوه اولویت آن بیشتر خواهد شد یعنی scheduler پروسس با اولویت بیشتر را برای تخصیص CPU انتخاب میکند پس در نهایت پروسس CPU-Bound از این مدل بیشتر سود میبرد.

۳. منظور از مفاهیم زیر را شرح دهید.

الف) Dispatcher یا توزیع کننده : یک ماژول است که بعد از اینکه scheduler تصمیم گرفت به کدام پروسس CPU اختصاص دهد ، این ماژول عمل context switch را انجام میدهد و عملا این ماژول است که کنترل را به پروسس جدید میدهد.

ب)Process Affinity یا وابستگی پردازنده : اگر سیستم مولتی کوری را درنظر بگیریم ، به وابستگی هر پروسس به هسته ای که در آن اجرا میشده ، affinity گویند ازاین جهت که هر هسته ، کش مخصوص دارد و اگر به هر دلیلی پروسس نتوانست اجرایش را به شکل متوالی کامل کند و باید مدتی در وضعیت انتظار برای گرفتن پردازنده باشد ، موقعی که میخواهد مجددا در یک هسته لود شود ، به صرفه تر است که در همان هسته ای که قبلا بوده لود شود . دو نوع affinity داریم :

soft : یعنی در صورتی که سربار هسته ی قبلی این پروسس زیاد باشد و پروسس توانایی migration به هسته ی دیگری را دارد

hard :توانایی migration ندارند.

ج) Virtual Run time در الگوریتم CFS : برابر با nicevalue + real runtime یعنی مدت زمانی که تا این لحظه اجرا شده و CPU در اختیار داشته به علاوه اولویتی که برای آن پروسس در نظر گرفته شده.

۴. با بیان مثال ، موقعیت هایی را مشخص کنید که هر یک از زوج پارامترهای زیر با هم در تضاد باشند.

الف) بهره وری از CPU و زمان پاسخ (response time و CPU utilization) : مثلا در حالتی که تعداد زیادی پروسس بخواهیم اجرا کنیم و همه آنها طولانی باشند الگوریتم خود را اینگونه در نظر بگیریم به صورتی که غیر از رعایت کوانتوم (کوچک) پروسه تازه رسیده را به پروسس در حال اجرا اولویت دهد در این صورت مثلا ۱۰ پروسس با فاصله میانگین ۱ وارد و هر کدام میانگین ۱۵ واحد زمانی برای اجرا نیاز دارند به هرکدام کوانتوم ۵ را اختصاص میدهیم اما اگر پروسس جدیدی وارد شد قبلی را متوقف میکند و خودش اجرا میشود (قبضه شدنی) در این حالت چون پروسس ها زیاد و سریع وارد میشوند و سریع هم پاسخ داده میشوند به شکل متداوم در حال context switch است و utilization کاهش میابد.

ب) متوسط زمان برگشت و ماکزیمم زمان انتظار : این حالت زمانی رخ میدهد که یک پروسس خیلی طولانی به نسبت بقیه پروسس ها وجود داشته باشد فرض کنیم میانگین زمان اجرای پروسس ها ۸ باشد این پروسس ۳۲ واحد زمانی نیاز دارد و کوانتوم ۴ در نظر گرفته شود و الگوریتم RR را پیاده کنیم. با فرض اینکه ۸ پروسس قرار است اجرا شود و در ابتدا پروسس ۳۲ ثانیه وارد شود و با میانگین ۲ یک پروسس جدید وارد شود اکنون بعد از ۴ واحد زمانی CPU از آن گرفته شده و به پروسس بعدی داده میشود. در این حالت میانگین زمان بازگشت کل فرایند کم میشود اما ماکزیمم زمان انتظار که متعلق به همین پروسس ۳۲ ثانیه ای است زیاد است یعنی در این حالت میانگین زمان بازگشت به طرز قابل توجهی کمتر از ماکزیمم زمان انتظار است.

۵. کدام یک از الگوریتم های زیر منجر به گرسنگی میشود ؟

الف) FCFS : اصلا گرسنگی ندارد زیرا هر پروسس به محض ورود به صف ready ها CPU میگیرد.

ب) SJF : در صورتی دچار گرسنگی میشویم که یک یا چند پروسس سی پی یو برست بالایی داشته باشند و همیشه پروسس های با سی پی یو برست کمتر بیایند ، در این صورت ممکن است به پروسس هایی با سی پی یو برست بالا هرگز سی پی یو نرسد.

ج) RR : هرگز گرسنگی نداریم به دلیل وجود مفهوم کوانتوم

د) Priority : در صورتی دچار گرسنگی میشویم که پروسس هایی با کمترین اولویت به نسبت بقیه ی پروسس هایی که می آیند ، وجود داشته باشد در این صورت به این پروسس ها با اولویت های کم ، هرگز سی پی یو نمی رسد.

۶. بررسی کنید کدام یک از الگوریتم های multi-level feedback queues و FCFS، RR برای پروسس های کوچک تبعیض قائل میشود؟(نفع یا ضرر)

در الگوریتم RR اگر کوانتوم خیلی کوچک باشد ، به ضرر پروسس های کوچک(با CPU-Burst کم) میشود چون زمان انتظار و بازگشت طولانی تر میشود . اگر کوانتوم خیلی بزرگ هم باشد باز به ضرر پروسس های کوچک است زیرا زمان بازگشت شان بیشتر میشود.

در الگوریتم FCFS اگر پروسس های کوچک دیرتر به صف ready آمده باشند ، متضرر خواهند شد در حالی که میتوانستند زودتر CPU بگیرند و اجرایشان را کامل کنند ولی اکنون زمان انتظار و در نتیجه زمان بازگشت شان طولانی تر میشود که خوب نیست.

در الگوریتم multi-level feedback queues بستگی به این دارد که پروسس های کوچک در صف ready با چه اولویتی و بر اساس چه الگوریتمی قرار داشته باشند و اینکه اگر این پروسس طبق بازخورد ها ، باید بین صف ها جابه جا شود دچار سربار اضافه خواهیم شد که نتیجتاً زمان انتظار و بازگشت را هم طولانی تر میکند پس نمیتوان به طور قطعی گفت که آیا به ضرر پروسس های کوچک است یا نه ولی میتوان با قطعیت گفت که این الگوریتم به شکل ۱۰۰٪ به نفع پروسس های کوچک نیست.

۷. میدانیم برای جلوگیری از بن بست باید یکی از شرایط ایجاد آن را نقض کنیم. یکی از این قاعده ها Hold and Wait است . دو راهکار برای نقض آن در کتاب مطرح شده :

- معایب هرکدام از راهکارها را بیان کنید .

- استفاده از این دو راهکار را برای سیستم عامل های تعاملی (Interactive) و سیستم عامل های بی درنگ (Realtime) بررسی کنید و بیان کنید کدام راهکار بهینه تر است .

برای اینکه شرط Hold & Wait نقض شود باید تضمین شود که هر گاه یک ترد یا پروسس یک منبع را درخواست کرد از قبل هیچ منبع دیگری را در اختیار نداشته باشد یعنی تمام منابع مورد نیازش را همزمان با هم دریافت کند

پس روش اول : به هر منبعی که نیاز داشت قبل از شروع اجرایش درخواست دهد تا زمانی که تمام منابع موردنیازش را دراختیار گرفت ، شروع به اجرا کند .

روش دوم : زمانی بتواند منبع مورد نیازش را دریافت کند که هیچ منبع دیگری را از قبل در اختیار نداشته باشد یعنی تمام منابع مورد نیازش را با هم به شکل همزمان دریافت کند.

در هر دو روش resource utilization کم میشود چون ممکن است است منبع توسط ترد اخذ شده باشد و هیچ استفاده ای از آن نشود همچنین ممکن است دچار گرسنگی شویم زیرا ممکن است حالتی رخ دهد که همه منابع مورد نیاز برای یک پروسس یا ترد به شکل همزمان فراهم نشود.

از آنجایی که در سیستم های بی درنگ زمان پاسخگویی بسیار مهم است بهتر است از روش دوم استفاده شود زیرا در روش اول سیستم بسیار زیاد با مشکل گرسنگی مواجه میشود البته حتی در این حالت هم زمان پاسخگویی طولانی میشود که مطلوب نیست .در روش اول هم زمان پاسخگویی طولانی میشود و ....

۸. الف) سیستمی را در نظر بگیرید که ۴ واحد از یک منبع یکسان را دارد که بین ۳ پروسس مشترک است .هر یک از پروسس ها حداکثر نیاز به ۲ واحد از منبع مورد نظر را دارند. نشان دهید که این سیستم آزاد از بن بست است.

چون اگر هر کدام (حداکثر نیاز – ۱) بگیرن بدترین حالته چون هیچ کس هیچ کدام از منابعی که در اختیارش است را آزاد نمیکنه به علاوه همگی منبع گرفته اند و در این حالت باز هم یه واحد از منبع باقیمانده و یکی از پروسس ها بالاخره منبع را قبضه میکند و بعد که اجرایش تمام شد هر دو منبع در اختیارش را آزاد میکند و در آن زمان هر دو پروسس دیگر میتوانند همزمان یه واحد دیگر از منبع که نیاز دارند را اخذ کرده و اجرا شوند.

ب) سیستمی با m منبع یکسان که بین n نخ به اشتراک گذاشته شده است در نظر بگیرید. هر نخ میتواند در هر لحظه فقط یک منبع درخواست دهد یا آزاد کند. نشان دهید که اگر دو شرط زیر برقرارباشد ، سیستم آزاد از بن بست است .

۱: ماکزیمم نیاز هر نخ بین یک و m منبع باشد.

۲: جمع ماکزیمم نیازها کوچکتر از m+n باشد.

اگر بازم هم بدترین وضعیت را درنظر بگیریم که هر پروسس به اندازه ی حداکثر نیازش منهای یک واحد از منابع موجود را قبضه کند پس فقط یک واحد دیگر منبع نیاز دارد تا اجرا شود از آنجایی که مجموع ماکزیمم نیازها کمتراز m+n است و n پروسس داریم پس در این شرایط کمتر از m واحد از منبع گرفته شده و حداقل یک واحد دیگر باقیمانده تا بالاخره یکی از پروسس ها آن را گرفته و اجرا شود و بعد از اتمام کارش تمام منابع در اختیارش را آزاد کرده و ما بقیه پروسس ها هم منبع مورد نیازشان که اکنون اماده هم هست را گرفته و اجرا میشوند و در نتیجه به بن بست نمیخوریم .