سوال ۶ تکلیف دوم OS

الف) برش‌های کوتاه ← q کوتاه ← زمانی می‌تواند به نفع سیستم باشد که پروسه‌ها IO bound باشند یا سیستم real time باشد در واقع زمانی مفید است که burst پروسه‌ها کم باشد

↙ پروسه‌ها می‌توانند سریع اجرا شوند و response time کمتری دارند.

البته این کار اگر بیشتر به زمان context switch هم دارد اگر زمان cs بیشتر از زمان q باشد ← بهینه نیست و چون تعداد cs ها با q ↓ ، q ↑ می‌یابد

→ بهینه نمی‌شود ← باید به زمان cs هم توجه کرد

ب) اگر q بزرگ باشد ← پروسه‌ها توی همان زمان q می‌توانند کامل اجرا شوند

برای سیستم‌هایی که cpu intensive یا cpu bound هستند مناسب است

چون این پروسه‌ها burst زیادی دارند ← برای اجرا به زمان بیشتری

نیاز دارند ← اگر q بزرگ باشد context switch هایی که بیهوده هستند را نداریم ←

بهینه‌تر است برای cpu bound ها و overhead هم کم می‌شود ←

↑ cpu utilization

ج) زمانی که q ثابت باشد ← اولویت پروسه‌ها اهمیتی نخواهد داشت ← به همه پروسه‌ها یک زمان برای استفاده از cpu می‌دهد.

د) برای ایجاد یک ترکیب خوب بین IO bound ها و cpu bound ها می‌شه مثل الگوریتم multilevel queue عمل کرد و مثلاً ۲ تا صف ایجاد کرد و در صف اول پروسه‌هایی را که IO bound هستند و burst کمتری دارند قرار داد و در صف دوم cpu bound هایی که burst بیشتری دارند را بگذاریم. ← این طوری از مزیت‌های هر دو الگو استفاده کردیم و ترکیبی از هردو داریم ← به هر دو گروه می‌توانیم پاسخ دهیم با این روش.

سوال ۶

(c) اگر q کوتاه باشد ← زمان response کوتاه می‌شود چون مدام context switch اتفاق می‌افتد ← **↑responsiveness** ← cpu در فاصله‌های کمی به پردازه‌ها می‌رسد.

اگر q بلند باشد ← زمان response هم بیشتر می‌شود و بیشتر طول می‌کشد که cpu به پردازه‌ها جواب بده ← **↓responsiveness**

بازدهی throughput هم اگر q کوتاه باشد ← throughput کم می‌شود

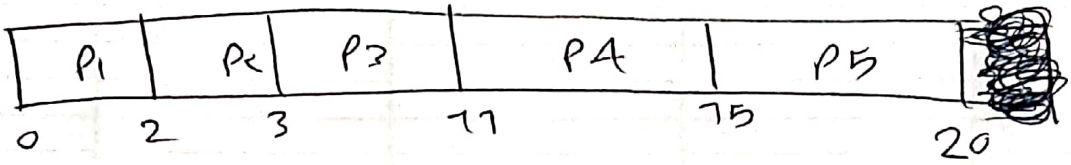
چون throughput یعنی تعداد پردازه‌هایی که اجراشان را در واحد زمان کامل می‌کنند ← یک بازده را نشان می‌دهد. وقتی q کم باشد پردازه‌ها فرصت نمی‌کنند تا در آن واحد زمانی داده شده (q) اجراشان را تمام کنند ← بازده یا throughput کم می‌شود

اما اگر q بزرگ باشد ← پردازه‌ها زمان کافی دارند تا اجرا شوند ← **↑throughput**

اگر q کم باشد ← تعداد context switch بیشتر می‌شود ← **↑overhead** ← **کاهش بهره‌وری cpu**

اما وقتی q زیاد باشد ← برای برنامه‌های cpu bound مثال الف ← به طور کلی از cpu خیلی بیشتر استفاده می‌شود ← **بهره‌وری cpu زیادتر می‌شود**

FCFS → first come first serve



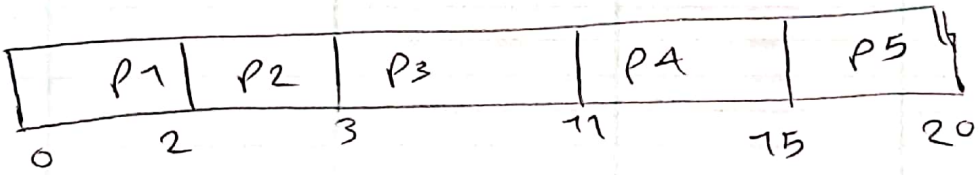
avg wt = $\frac{0 + (2-1) + (3-2) + (11-4) + (15-4)}{5} =$

$\frac{20}{5} = 4$

avg TT = $\frac{(2) + (3-1) + (11-2) + (15-4) + (20-4)}{5}$

$= \frac{40}{5} =$ 8

SJF → shortest job first → non preemptive



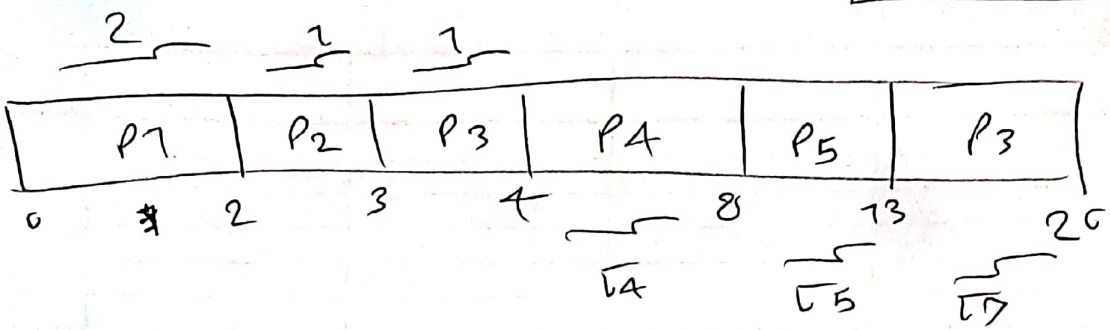
avg wt = $\frac{0 + (2-1) + (3-2) + (11-4) + (15-4)}{5} =$

$\frac{20}{5} = 4$

avg TT = $\frac{2 + (3-1) + (11-2) + (15-4) + (20-4)}{5} =$

$\frac{40}{5} = 8$

SRF → shortest remaining first → preemptive

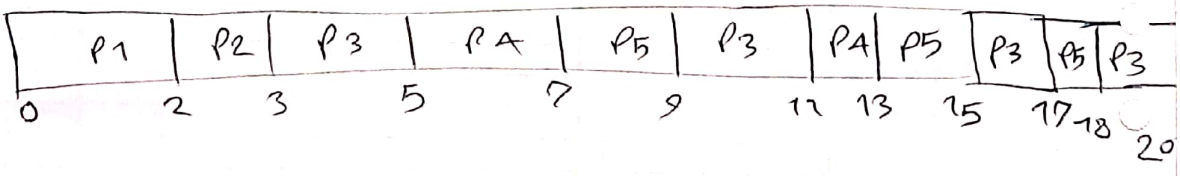


avg wt = $\frac{0 + (2-1) + (3-2) + (13-3) + 0 + 5}{5} = \frac{15}{5} = 3$

avg turnaround = $\frac{2 + (3-1) + (8-2) + (18-3) + (13-4) + 9}{5} = \frac{50}{5} = 10$

RR → round Robin q = 2

P3 → 8 → 6 → 4 → 2 → 0



P4 → 4 → 2 → 0 P1 → 2 → 0 ✓

P5 → 5 → 3 → 1 P2 → 1 → 0 ✓

avg wt = $\frac{0 + (2-1) + (3-2) + (9-5) + (15-11) + (18-17) + (5-4) + (11-7) + (7-4) + (13-9) + (17-15)}{5} = \frac{25}{5} = 5$

RR

avg TT = $\dfrac{2 + (3-1) + (20-2) + (73-4) + (78-4)}{5}$

(corrections written above: 2, 18, 69, 74)

$\dfrac{78 + 27}{5}$ (45 written above) $= \boxed{91}$

سوال ۳

q = 50ms ← پروسس اگر از تمام q استفاده کند و به پایان نرسد ← به 70ms q ها افزوده می‌شود

الف) کدام سود بیش‌تری می‌برد؟ چون در این روش به زمانی که به پروسه‌ی cpu داده می‌شود این پروسه از تمام q اش استفاده می‌کند و بلاک هم نمی‌شود یعنی پروسه cpu bound است ← هر چه زمان بیش‌تری بگیرد q بزرگ‌تر می‌شود براش) و بعد می‌تواند عملیات cpu را انجام دهد. همچنین اولویت بالاتری هم می‌گیرد ← احتمال اینکه cpu را زودتر بیش‌تر اختصاص داده شود هم بالاتر می‌شود ← در کل با این شرایط cpu bound سود می‌کند.

I/O bound چون اولویتش تغییری نمی‌کند سود نمی‌کند (نسبت به cpu bound) و q آن هم کمتر می‌شود ← امکان اینکه کل پروسه در q اجرا نشود هم هست.

ب) هدف در زمان‌بندی: اینکه زمان بیش‌تری در اختیار پروسه‌های cpu bound بگذاریم و q را افزایش دهیم و بلاک نشوند پروسه‌ها ← باعث می‌شود تعداد context switch ها خیلی کمتر شود ← تمرکز cpu بیش‌تر روی پروسه‌های cpu bound قرار می‌گیرد مدت زمان طولانی‌تری busy است ← ↑ cpu utilization

در کل هدف این سیستم اجرای پروسه‌های cpu bound در کنار I/O bound است طوری که اولویت پردازشی cpu bound ها بالاتر باشد و اجراشون هم کوتاه‌تر باشد. مثلاً در محاسبات data mining یا محاسباتی که باید cpu عملیات زیادی انجام دهد و تمرکز کمتری روی I/O داشته باشد.

سوال ۲

الف) Dispatcher و وقتی scheduler تصمیم می‌گیرد که CPU را از یک پروسه بگیرد و به یک پروسه دیگر بدهد، یک ماژول سیستمی به اسم dispatcher وکنترل CPU را از پروسه اول می‌گیرد و به پروسه بعدی که scheduler انتخاب کرده.

در واقع وظیفه‌ی dispatcher جابه‌جایی CPU است ← context switch صدا را انجام می‌دهد.

ب) affinity یا وابستگی

cache ← پراسسورها دائم با اطلاعات حافظه‌ی threadهایی که دارند روی اون core اجرا می‌شوند، پر می‌شود چون دفعه‌ی بعد سریع‌تر از خود memory است.

پس وقتی یک تردی (یا thread) داشته توی یک core اجرا می‌شود و به اون core وابسته شده و دوست نداره از اون core جدا بشه چون اگه اون thread را از اون core جدا کنیم بیرم توی یه core دیگه، cache ← core جدید، باید از اول بیاد داده‌های مربوط به این thread را از حافظه‌ی اصلی توی حافظه‌ی خودش ذخیره کند ← هزینه‌ی زیادی برای ما

از cache به خوبی استفاده نشده

ج) virtual runtime و

اگه اولویت پروسه‌ها را در نظر بگیریم، runtime ← یک پروسه یعنی چقدر اون پروسه تاحالا توی CPU اجرا شده ← این مقادیر را ذخیره می‌کنیم تا وقتی scheduler خواست یک پروسه جدید را انتخاب کند تا بهش CPU بدهد، اون پروسه‌ای را انتخاب می‌کند که runtime ← کمتری داشته

اگه بخاهیم اولویت پروسه‌ها را هم در نظر بگیریم، یک مقدار جدیدی از روی runtime تعریف می‌کنیم به اسم virtual runtime که این مقدار به اولویت ← task ها وابسته است.

یعنی اگه یک task اولویت متوسطی داشته باشد ← vruntime = actual runtime

اگه اولویت بالایی داشته باشد ← virtual runtime < actual runtime

کمی ← >

سوال ۲

الف) cpu utilization و responsetime

به طور مثال در الگوریتم round robin و برای کاهش زمان response و باید quantum time را کم کنیم ← اگر q را خیلی کم کنیم و تعداد context switch ها هم زیاد می‌شود ← می‌دانیم cs هم یک زمانی می‌گیرد تا اجرا شود و او پردازه را عوض کند ← یک overhead دارد هر بار درست می‌کند ← باعث کاهش بهره‌وری از cpu می‌شود.

در حالت کلی و بهره‌وری از cpu زمانی زیاد می‌شود که کمترین overhead را در context switch ها داشته باشیم. overhead های cs و زمانی می‌توانند کمتر شوند که تعداد cs ها را کم کنیم ← زمان response زیاد می‌شود برای پردازه‌ها ← تضاد دارد

ب) میانگین زمان برگشت و زمانی min می‌شود که کوتاه‌ترین task و اول اجرا شود ←

اگر الگوریتم scheduling این طوری باشد ← مثل shortest job first

↓ باعث گرسنگی در پردازه‌هایی که زمان زیادی می‌خواهند برای اجرا شدن و می‌شود

↓ باعث افزایش زمان انتظار می‌شود

سوال ۵

الف) FCFS ← خیلی X چون به پروسه ها داریم به ترتیب CPU ی دهیم ← گرسنگی هیچ وقت پیش نمیاد

ب) SJF ← بله ✓ ممکن است برخی از پروسه ها دچار گرسنگی شوند
توی این الگوریتم چون همیشه داریم اون پروسه‌ای را انتخاب می کنیم که کمترین burst را دارد (یعنی CPU کمتری داره) ← اگر همیشه یک پروسه وارد شود که burst کمتری نسبت به P2 دارد ← P2 هیچ وقت اجرا نمی‌شود ← دچار گرسنگی می‌شود

ج) RR ← round robin ← خیلی X چون به نوبت به پروسه ها CPU می‌دهیم و عادلانه رفتار می کنیم

د) priority ← بله ✓ اون پروسه‌هایی که اولویت کم دارند و ممکن است هرگز اجرا نشوند چون در این الگوریتم اول اون پروسه‌های اجرا می‌شوند که اولویت بالاتری دارند

سوال ۴

در الگوریتم FCFS اگر اول IO bound ها اجرا شوند و بعدش CPU bound ها average waiting time کمتری می‌شود اما اگر اول CPU bound ها اجرا شود avg wt زیاد می‌شود و اصلا بهینه نیست ← این الگوریتم برای کارهای short مناسب‌تر است تا کارهای long

حالت بهینه اول کوچکترها اجرا شوند ← به نفع پروسه‌های کوچک می‌شود
حالت بد اول بزرگ‌ترها ← به ضرر
در حالت کلی این الگوریتم تبعیض برای پروسه‌های کوچک قائل می‌شود
چون هر پروسه‌ای که زودتر آمده زودتر اجرا می‌شود ← تبعیض ندارد | و عادلانه رفتار می‌کند

ادامه‌ی سوال ۶

RR ← توی این الگوریتم وقتی به پردازه‌ای یک قسمت کوچکی از cpu time
یا همان time quantum می‌گیرد و بعد از گذشت زمان q، cpu از پردازه گرفته
می‌شود و به بعدی داده می‌شود ← به صورت کاروانه با پردازه‌ها رفتار می‌کند و
تعیین قائل می‌شود

اگر q خیلی بزرگ باشد ← پردازه‌های کوچک در همان زمان q اجرا می‌شوند
و صبح مثل همان FCFS

اگر هم q خیلی کوچک باشد ← تعداد context switch ها زیاد می‌شود

اما در حالت کلی چون به هر پردازه در اندازه‌ی q زمان اجرا در cpu می‌دهیم و
زمان response کمتر می‌شود و کاروانه رفتار می‌کند
اگر q را مناسب انتخاب کنیم.

multi_level feedback queue ← توی این الگوریتم، پردازه‌ها را طبق زمان اجرایشون در صف‌های
مختلف قرار می‌دهیم ← در صف اول، اون پردازه‌های قرار می‌گیرند که
time quantum کمی دارند (real time ها مثلاً) در صف دوم، اونهایی که زمان اجرایشون
کمی طولانی‌تر است، در صف سوم، cpu bound ها قرار می‌گیرد ←
در این الگوریتم، اولویت اجرا با پردازه‌های کوچک است ← به نفع پردازه‌های کوچک
عمل می‌کند ← تعیین می‌گذارد
بیشترین اولویت با real time process ها است بعد پردازه‌های سیستم
بعد interactive process ها و در آخر، پردازه‌های batch که cpu bound
هستند - البته باید دقت کرد که توی این الگوریتم، اگر یک پردازه‌ای داشتیم که
در ابتدا cpu bound بود، وسط اجرا، یک خواست، اولویتش زیاد می‌شود و
feedback میره و میره در صف اول