آریا خلیق ۹۵۲۴۰۱۴

تمرین سری دوم درس سیستم‌های عامل

۱)

در بسیاری از سیستم‌ها فرایندها دایما در حال تغییر هستند به همین دلیل یک روند ثابت برای اجرای این فرایندها وجود ندارد به همین دلیل فقط می‌توان تقسیم زمانی CPU را مورد بحث قرار داد. به خاطر همین ما از تخمین استفاده می‌کنیم و یک تابع توزیع احتمال به دست می‌اوریم. این تابع توزیع نمایی می‌باشد.

یک کامپیوتر را می‌توان شبکه‌ای از سرورها دانست که هر سرور یک صف از فرایندهای در صف دارد. CPU هم یک سرور می‌باشد که به فرایندها سرویس می‌دهد.

فرض کنیم یک صف به طول متوسط n داریم و W هم متوسط زمان انتظار فرایند در صف باشد و L متوسط نرخ رسیدن و وارد شدن فرایندهای جدید به صف باشد. اگر صف در حالت پایدار باشد باید تعداد قرایندهایی که وارد می‌شوند برابر تعدادی باشد که خارج می‌شوند که بنابراین باید رابطه زیر برقرار باشد:

$n = L * W$

این فرمول، فرمول Little نام دارد و همچنین برای هر الگوریتم زمان‌بندی و رسیدگی به ورودی ها صادق است. با این فرایند می‌توان هر یک را با دوتای دیگر حساب کرد و تخمین زد.

۲)

مشکل زمان‌بندی زمانی رخ می‌دهد که یک فرایند با اولویت بالاتر می‌خواهد اطلاعات کرنل را که الان فرایند با اولویت پایین‌تر به آن دسترسی دارد را بخواند یا بنویسد. اطلاعات کرنل هم توسط Lock در مقابل یک فرایند دیگر نگه داری می‌شوند به همین دلیل اگر فرایند با اولویت بالا بخواهد به آن دسترسی پیدا کند باید منتظر پایان فعالیت فرایند کم‌اولویت‌تر باشد. این مشکل در سیستم‌های با تعداد اولویت بیشتر از ۲ رخ می‌دهد.

فرض کنیم سه فرایند ۲، ۱ و ۳ را داریم. (۱ بیشترین اهمیت، ۲ متوسط و ۳ کمترین) در ابتدا فقط فرایند ۱ در حال اجرا است که وارد CS می‌شود، حال فرایند ۲ شروع شده و CPU را در دست می‌گیرد. حال فرایند ۱ شروع به کار می‌کند و CPU را از فرایند ۲ می‌گیرد. فرایند ۱ می‌خواهد وارد CS خود شود ولی چون فرایند ۳ داخل CS است بلاک می‌شود. CPU به فرایند ۲ داده می‌شود و تا زمانی که اجرای آن تمام نشده و CPU به فرایند ۳ داده نشده که از CS درآید در همین حالت می‌ماند.

برای رفع مشکل برای اولویت‌های بیشتر از ۲ تا از priority-inheritance protocol استفاده می‌شود که اولویت فرایند کمتر که داخل CS است به بالاترین سطح یا سطح اولویت بالاتر می‌رسد و اینگونه از شر اولویت‌های وسط در امان می‌ماند.

۳)

سیستم‌های بلادرنگ سیستم‌هایی هستند که زمان در آن‌ها نقش بسیار مهمی دارد. این سیستم‌ها به دود دسته hard real time و soft real time تقسیم می‌شوند.

در hard real time باید سر زمان مشخص **حتماً** پاسخ و نتیجه را دریافت کنیم و این زمان به هیچ عنوان نباید رد شود ولی در حالت soft ممکن بسیار سعی می‌شود رد نشود ولی ممکن است کمی تفاوت داشته باشد.

وقایعی که سیستم‌های بلادرنگ به آن باید پاسخ دهند به دو بخش periodic و aperiodic تقسیم می‌شوند که در مواقعی سیستم باید به تعدادی وقایع periodic پاسخ دهد. ممکن است تعداد وقایع و زمان بندی آن‌ها به گونه‌ای باشد که پاسخگویی به همه آن‌ها غیر ممکن باشد.

فرض کنیم واقعه ۱ دارای پریود P1 و زمان C1 برای پاسخ به آن نیاز باشد. باید جمع تمامی Ci/Pi ها کمتر از ۱ باشد که سیستم بتواند به آن‌ها پاسخ دهد و اگر بیشتر باشد غیر ممکن می‌شود (overhead پردازنده بسیار کوچک در نظر گرفته شده)

الگوریتم‌های سیستم‌های بلادرنگ می‌تواند static یا dynamic باشد. در حالت استاتیک تصمیمات باید قبل از شروع کار سیستم گرفته شوند و در حالت داینامیک سیستم در حال اجرا تصمیمات را می‌گیرد. حالت استاتیک زمانی ممکن است که اطلاعات خیلی خوبی در مورد فرایندها قبل از شروع داشته باشیم.

۴)

۳ دسته بندی کلی داریم: Batch – Interactive – Real time

به طور کلی یک الگوریتم زمان‌بندی باید دارای شرایط زیر باشد:

- دادن عادلانه CPU به هر فرایند => باید به فرایندها بر اساس نیازشان، اهمیتشان و ... CPU اختصاص داد.
- قسمت‌ها را به صورت بالانس فعال و مشغول نگه داشتن => مثلاً اینگونه نباشد که اول فعالیت‌های در ارتباط با CPU اجرا و بعد فرایندهای مربوط به I/O Device اجرا شوند و این دو باید هم‌زمان با هم اجرا شوند.
- باید اطمینان داشت که قوانین به درستی اجرا می‌شوند => مثلاً یک فرایند برای نباید برای مدت زیادی داخل CS باشد و دایم باید برای درست اجرا شدن قانون چک شود.

در Batch سیستم‌ها:

- زمان Turnaround Time حداقل
- CPU باید همیشه مشغول باشد => CPU گران است و بهتر است همواره رد حال اجرای کاری باشد.
- Throughput (تعداد کارهایی که در یک ثانیه کامل می‌شوند) باید حداکثر باشد

نکته: لزوماً افزایش throughput باعث کاهش turnaround time نمی‌شود.

در Interactive سیستم‌ها:

- Response time سریع => برای کاربر بسیار مهم است.
- برآورده کردن انتظارات کاربران => مثلاً اتصال به مودم به زمان ۴۰ ثانیه نیاز دارد و این برای کاربر پذیرفته شده است ولی قطع کردن آن نه.

در Realtime سیستم‌ها:

- اجرای هیچ فرایندی نباید ددلاین را رد شود => ددلاین‌ها بسیار مهم هستند.
- جلوگیری از افت کیفیت در سیستم‌های چند رسانه‌ای => مثلاً در سیستم‌هایی که ارتباط با صدا دارند لگ زیاد پذیرفته شده نیست.

۵)

در سیستم‌عامل زمانی که یک فرایند fork می‌شود یعنی یک کپی از خود ایجاد می‌کند. در سیستم‌عامل UNIX و سیستم‌عامل‌های مشابه (GNU/Linux) این کار از طریق system call به نام fork انجام می‌شود. فرایند ایجاد شده را فرایند فرزند و فرایند اصلی که فرایند از آن fork شده را والد می‌نامند.

فرایند والد اجرای خود را ادامه می‌دهد در حالی که فرایند فرزند شروع به اجرا از همان جایی که fork شده می‌کند.

دو فرایند جدا یک برنامه را اجرا می‌کنند ولی data، stack و heap آن‌ها متفاوت از هم است. در زمانی که تابع fork فراخوانی شده بود قسمت‌های data، stack و heap والد و فرزند مشابه هم هستند ولی پس از آن متفاوت خواهند بود به همین دلیل هر فرایند که داده‌های خود را تغییر دهد روی دیگر تأثیری نمی‌گذارد.

فراخوانی fork که در کتابخانه unistd.h تعریف شده است پس از فراخوانی سه مقدار باز می‌گرداند.

- If fork() returns negative value; creation of child process was unsuccessful.
- If fork() returns a zero; newly child process is created.
- If fork() returns positive; process _ID of child process to the parrent. It is type of pid_t defined as in sys/types.h

۶)

زمان‌بندی در لینوکس:

برای لینوکس ۲.۴ به قبل لینوکس از الگوریتم زمان‌بندی real-time به همراه non-real-time استفاده می‌کرد. از لینوکس ۲.۶ به بعد الگوریتم non-real-time در لینوکس بازنگری شد.

الگوریتم زمان‌بندی Real-Time:

سه کلاس زمان‌بندی در لینوکس عبارتند از:

- SCHED_FIFO => FIFO real-time thread
- SCHED_RR => Round-Robin real-time thread
- SCHED_NORMAL => Other, Non-Real_Time thread

داخل هر کلاس می‌توان از انواع اولویت‌ها استفاده کرد. هر یک از این مقادیر حالت پیشفرض دارند.

برای نخ‌ها FIFO قوانین زیر اعمال می‌شود:

۱) سیستم به این رشته اینتراپت نمی‌دهد جز در مواقع زیر:

- یک نخ دیگر با اولویت بالاتر در حالت ready باشد.
- اجرای نخ بلاک شود برای دریافت یک event مثل I/O
- خود نخ به صورت داوطلبانه CPU را پس بدهد.

۲) زمانی که در حین اجرای نخ اینتراپت داده می‌شود در جای مناسب خود با توجه به اولویتش قرار می‌گیرد.

۳) اگر نخ در حالت ready قرار می‌گیرد و اولویت بالاتری از نخی که در حال اجرا است دارد CPU به نخ با اولویت بالاتر داده می‌شود.

کلاس SCHED_RR بسیار شبیه به کلاس SCHED_FIFO می‌باشد ولی time slice بیشتری در در هنگام انتظار به آن می‌رسد. زمانی که نخ SCHED_RR برای یک time slice اجرا می‌شود آن نخ suspend می‌شود و یک نخ با اولویت برابر یا بیشتر برای اجرا انتخاب می‌شود.

زمان‌بندی در ویندوز:

هدف ویندوز ارایه تجربه کاربری عالی به کاربران در هر دو حالت دسکتاپ و سرور می‌باشد. ویندوز یک زمان‌بند قابل پس گیری را با حالت‌ها و درجه‌های مختلف قابل پس گیری بودن پیاده‌سازی می‌کند. در ویندوز نخ‌ها واحدهای زمان‌بندی هستند(به جای فرایندها)

اولویت‌ها در ویندوز به دو دسته بلادرنگ و متغیر تقسیم می‌شوند و هر کدام هم دارای ۱۶ اولویت مختلف هستند. نخ‌هایی که باید به صورت بلادرنگ رسیدگی شوند در دسته بندی real-time قرار می‌گیرند و به صورت بلادرنگ اجرا می‌شوند. ویندوز از یک سیستم دارای اولویت استفاده می‌کند که نخ‌های در گروه بلادرنگ دارای اولویت بالاتر نسبت به بقیه هستند. در کلاس نخ‌های بلادرنگ اولویت‌ها ثابت هستند و در این کلاس به صورت Round-Robin سرویس داده می‌شود.

۷)

این الگوریتم تقریباً سعی می‌کند به صورت یکسان به فرایندهایی که منتظر CPU هستند سرویس دهیم و زمانی که یک فرایند وارد شد با آن مثل زمانی رفتار می‌شود که انگار قبلاً وجود داشته. به عبارتی الگوریتم به گذشته نزدیک خود نگاه می‌کند و سعی می‌کند به فرایندها به گونه‌ای سرویس دهد که به صورت مساوی CPU دریافت کنند.

در این حالت بین فرایندها با اولین بالاتر و عطش برای CPU و فرایندهای دیگر تفاوتی قایل نمی‌شویم و هر دو را به صورت برابر می‌بینیم چون ممکن است فرایند با عطش CPU در ابتدا انجام شود ولی بعد از آن همان زمان را به فرایندی با اولویت کمتر خواهیم داد. در این حالت عدالت رعایت نمی‌شود. برای فرایندها با عطش I/O هم به همین شکل است.

۸)

دو نوع سیستم Real-Time داریم، اولی که Soft Real-Time است سیستم گارانتی نمی‌کند که عملیات سر ددلاین انجام شوند در عوض فعالیت‌های بحرانی را به فعالیتّ‌ای غیر بحرانی ترجیح می‌دهد. در Hard Real-Time نیازمندی‌هایمان بیشتر است و هر فعالیت و وظیفه باید در ددلاین خود انجام شود و رد شدن ددلاین به هیچ وجه مورد قبول نیست.

مقداری تأخیر با توجه به نوع کار سیستم و آن فعالیت قابل تحمل است، دو نوع تاخیر داریم:

- تأخیر وقفه => فاصله زمانی که وقفه به وجود می‌آید تا زمانی که CPU شروع به پردازش وقفه می‌کند.
- تأخیر Dispatch => مدت زمانی که نیاز است تا dispatcher یک فایند را متوقف و فرایند دیگر را شروع کند.

انواع زمان‌بندی‌ها:

زمان‌بندی Priority-Based:

مهم‌ترین کاری که الگوریتم‌های زمان‌بندی در سیستم‌های Real-Time باید انجام دهند، دادن هر چه سریع‌تر CPU به فرایندی است که اولویت بالاتری دارد و CPU می‌خواهد.
در این نوع زمان‌بندی ما اولویت‌هایی برای انواع مختلف فرایندها داریم (در ویندوز ۳۲ نوع اولویت مختلف که از ۱۶ تا ۳۱ برای فرایندهای real-time است).
این نوع زمان‌بندی Soft Real-Time را تضمین می‌کند.

زمان‌بندی Rate-Monotonic:

این الگوریتم از الگوریتم زمان‌بندی کلی مبتنی بر دادن اولویت به کارها پیروی می‌کند. در ای نوع الگوریتم هر فرایندی که وارد می‌شود بر اساس پریود تکرار آن یک اولویت (هر چقدر پریود آن کوتاه‌تر باشد اولویت آن بیشتر است)
علت این کار دادن اولویت بالاتر به task هایی هست که در بازه زمانی کوتاه‌تری تکرار می‌شوند و این کار زمان انتظار را کاهش می‌دهد.

زمان‌بندی Earliest-Deadline_First:

در این نوع زمان‌بندی اولویت‌بندی به صورت داینامیک با توجه به ددلاین فرایندها صورت می‌گیرد به این صورت که هر چه ددلاین کاری زودتر باشد اولویت آن بالاتر می‌شود و برعکس. بر طبق قوانین این الگوریتم زمانی هر فرایند باید ددلاین خود را به سیستم اعلام کند. در این نوع زمان‌بندی بر خلاف Rate-Monotonic دیگر اولویت‌ها ثابت نیستند.

زمان‌بندی Proportional:

در این زمان بندی کلاً T سهم بین تمامی اپلیکیشن‌ها تقسیم می‌شود. هر اپلیکیشن می‌تواند n سهم داشته باشد و در کل صاحب n/T زمان CPU خواهد بود.
این نوع زمان‌بندی باید همراه قانون admission-control باشد که مطمین شویم هر اپلیکیشن زمان مناسب و مشخص خود را دریافت می‌کند.

زمان‌بندی POSIX Real-Time:

POSIX یک استاندارد برای برنامه‌ریزی سیستم‌های بلادرنگ است. این سیستم دو کلاس برای threadها مشخص می‌کند(قبلا بررسی شده):

- SCHED_FIFO
- SCHED_RR

کلاس اول threadها را بر اساس first come firs serve و صف FIFO اولویت بندی می‌کند و فرایند با اولویت بیشتر که در ابتدای صف است CPU را گرفته و تا زمانی که تمام شود یا بلاک شود آن را نگه می‌دارد. در حالت SCHED_RR به صورت round robin به نخ‌هایی با اولویت برابر رسیدگی می‌شود.

تمرین امتیازی ۱)

۱) چرا باید اجازه اجرای هم‌زمان برنامه‌ها را داد؟
برنامه‌ها می‌توانند به صورت هم‌زمان اجرا شوند، چندین برنامه که هم‌زمان اجرا می‌شوند می‌توانند با هم همکاری داشته باشند. از فواید اجازه اجرای هم‌زمان دادن: فرض کنیم چند کاربر یا برنامه می‌خواهند با همزمان با هم به یک منبع دسترسی داشته باشند. باید این اجازه را داد . در سیستم‌هایی با چند هسته پردازنده می‌توان یک برنامه را برای افزایش سرعت به تکه‌هایی تقسیم کرد و اجرای آن را به صورت هم‌زمان جلو برد. کاربر ممکن است هم‌زمان چندین برنامه را با هم اجرا کند.

۲) فواید استفاده از نخ به جای ایجاد چند فرایند هم‌زمان را در سرور شرح دهید.
در گذشته این اتفاق می‌افتاد که زمانی که یک سرور یک request دریافت می‌کرد، یک فرایند جدا برای سرویس دادن به آن request ایجاد می‌کرد. اینکه برای هر request یک فرایند بسازیم زمان زیادی می‌برد و منابع زیادی هم مصرف می‌کند به جای این کار بهتر است از یک فرایند استفاده کنیم که خود چند نخ دارد. در این حالت سرور یک نخ درست می‌کند که به request ها گوش می‌کند. زمانی که یک request از client آمد به جای ایجاد فرایند یک نخ برای پاسخ به آن request ایجاد می‌شود.

۳) راه حلی عملی برای رفع مشکل starvation در سیستم‌های دارای اولویت ارایه کنید.
در اینگونه سیستم‌ها ممکن است فرایندهایی که اولویت کمی دارند همواره منتظر CPU بمانند ولی هیچ‌گاه CPU دریافت نکنند و تا همیشه در انتظار CPU باشند. برای رفع این مشکل از Aging استفاده می‌شود به گونه‌ای که فرایندی که مدت زیادی منتظر CPU مانده به تدریج اولویتش زیادتر می‌شود، اینگونه اگر برای مدت زمان زیادی CPU به آن نرسد اولویتش به گونه‌ای افزایش پیدا می‌کند که CPU دریافت کند.

تمرین امتیازی ۲)

۱) چرا نرم‌افزارها نباید به زمان‌بندی خود سیستم اتکا کنند؟
فرض کنید می‌خواهیم یک ویدیو اجرا کنیم که همراه با خود یک صوت دارد. اگر به زمان‌بندی خود سیستم اتکا کنیم و برای مثال یک loop خالی با ۱۰۰ بار اجرا بگذاریم ممکن است CPU در وسط از صدا گرفته شود و زمان‌بندی صدا و ویدیو به هم بخورد.

۲) پیاده‌سازی نخ‌ها در فضای کاربر چه مزیتی نسبت به کرنل دارد؟
اولین مزیت این است که کرنل ممکن است اصلاً threading را ساپورت نکند و این مشکل را می‌توان به کمک کتاب‌خانه ها و پیاده‌سازی در فضای کاربر رفع کرد. مورد بعدی این است که می‌توان الگوریتم زمان‌بندی را به صورت دلخواه پیاده‌سازی کرد (مثلاً garbage collector در یک حالت نامناسب متوقف نشود). به scalability هم کمک می‌کند چون table space های کاربران و کرنل جداست وجود تعداد بسیار زیادی thread در کرنل باعث ایجاد مشکل می‌شود.

۳) اصلی‌ترین و مهم‌ترین اهداف الگوریتم‌های زمان‌بندی چیست؟
باید با در نظر گرفتن اولویت فرایندها به آن‌ها CPU داده شود. مثلاً اینکه همیشه یک فرایند CPU داشته باشد یا یک برنامه بسیار پر اهمیت به میزان کمی CPU دریافت کند پذیرفته نیست. برنامه زمان‌بند باید قوانینی را که برای فرایندها وجود دارد چک کند و از اجرا شدن آن‌ها مطمین شود. در مورد سوم زمان‌بند باید از اجرای همیشه و لازم CPU مطمین باشد زیرا CPU یک قطعه گران است و زمانی که کار داشته باشیم باید مشغول انجام دادن کارها باشد.

پاسخ تمرینات تشریحی:

۱) الگوریتم زمان‌بندی دوری با کوانتوم زمانی ۲۵

A → ۶۰ → ۴۰
B → ۲۵
C → ۵۰ → ۴۰ → ۱۰
D → ۴۰ → ۱۰

Context switch → ۵
RR و TS → ۲۰

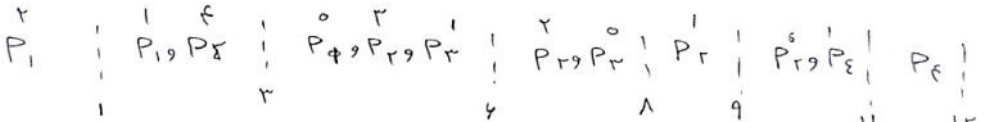
Gantt chart:

	ورود	C.B.T	خروج	T.T	W.T
A	۰	۴۰	۱۲۰	۱۲۰	۸۰
B	۰	۲۰	۴۵	۴۵	۲۵
C	۰	۵۰	۱۷۵	۱۷۵	۱۲۵
D	۰	۳۰	۱۶۰	۱۶۰	۱۳۰

$$T.T = \frac{۱۲۰ + ۴۵ + ۱۷۵ + ۱۶۰}{۴} = ۱۲۵ \qquad W.T = \frac{۸۰ + ۲۵ + ۱۲۵ + ۱۳۰}{۴} = ۹۰$$

۲) الگوریتم آزاد ۸۱

زمان [illegible] باقی‌مانده تا اجرای فرایند بالای آن نوشته شده

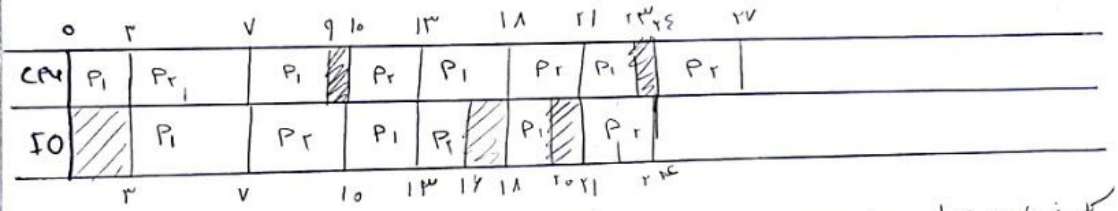


	ورود	C.B.T	خروج	T.T	W.T
P۱	۰	۳	۴	۴	۳
P۲	۱	۵	۱۱	۱۰	۵
P۳	۳	۲	۸	۵	۳
P۴	۹	۲	۱۲	۳	۱

$TT = 4\ min$ $\qquad W.T = \frac{3 + 0 + 3 + 1}{4} = 2\ min$

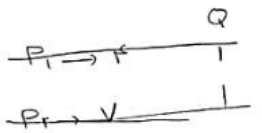
③ ۱، شد، دولتی، ۸۱

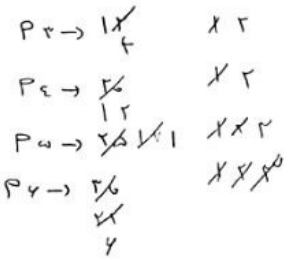
P_1	P_2
~~CPU 3~~	~~CPU 4~~
~~Net 4~~	~~Disk 3~~
~~CPU 2~~	~~CPU 3~~
~~Disk 3~~	~~Net 3~~
~~CPU 5~~	~~CPU 3~~
~~Disk 2~~	~~Net 3~~
~~CPU 2~~	CPU 3

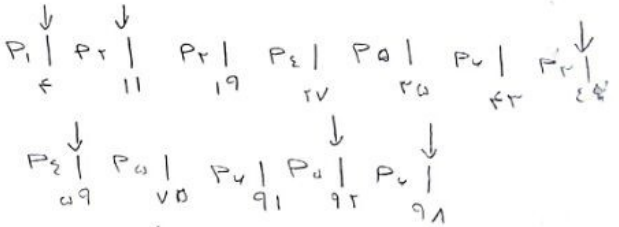


کل زمان سرایر $= \max(27, 24+27) = 27$ و مقدار هدر رفتنی ۲ است.

④ ۱، شد، دولتی، ۸۵







Avg W.T = ۲۷/۵

Avg T.T = ۵۱/۲۸

	CBT	T.T	W.T
P_1	۴	۴	۰
P_2	۷	۱۱	۴
P_3	۱۲	۴۷	۳۵
P_4	۲۰	۵۹	۳۹
P_5	۲۵	۹۲	۶۷
P_6	۳۵	۹۸	۶۸

⑤ الگوریتم SJF آزاد

	CBT	Arrival time	T.T	W.T
P_1	۷	۰	۱۴	۸
P_2	۳	۱	۳	۰
P_3	۲	۳	۳	۱
P_4	۳	۷	۴	۰

$P_1 \to \frac{8}{10}$

$P_2 \to \frac{0}{0}$

$P_3 \to \frac{1}{0}$

$P_4 \to \frac{0}{0}$

P_1 | P_2 | P_3 | P_1 | P_4 | P_1

۱ ۴ ۷ ۱۰ ۱۴

AVG W.T: $\frac{۱+۰+۱+۸}{۴} = ۲/۲۵$

---

⑥ الگوریتم SJF آزاد با ۵

	CBT	AT	
P_1	۷	۰	
P_2	۸	۱	
P_3	۲	۳	
P_4	۴	۵	
P_5	۲	۷	

① P_1: $\frac{۵+۷}{۷} = \frac{۱۲}{۷}$ P_3: $\frac{۳+۳}{۲} = ۳$ ✓

P_4: $\frac{۱+۳}{۲} = \frac{۴}{۲}$

② P_2: $\frac{۸+۷}{۷} = \frac{۱۵}{۷}$ P_4: $\frac{۴+۲}{۲} = \frac{۶}{۲}$ ✓ P_{5-1}: $\frac{۲+۲}{۲} = ۲$

③ P_5: $\frac{۵+۲}{۲} = \frac{۷}{۲}$ ✓ P_2: $\frac{۱۱+۷}{۷} = \frac{۱۸}{۷}$

P_1 | P_3 | P_4 | P_5 | P_2

۰ ۶ ۹ ۱۲ ۱۴ ۲۱

⑦ ارشد، آزاد، ۸۵

	BST	AT	TT	W.T
P_0	۴	۵	۱۳	۷
P_1	۴	۲	۱۶	۱۲
P_2	۲	۳	۳	۱
P_3	۱	۸	۲	۱

~~$P_0 \to$ ۴ ۲~~
~~$P_1 \to$ ۴~~
~~$P_2 \to$ ۲~~
~~$P_3 \to$ ۱~~

P_0 P_2 P_0 P_3 P_0 P_1
۳ ۴ ۶ ۷ ۸ ۹ ۱۰ ۱۱ ۱۲ ۱۴ ۱۸

AVG W.T: $\frac{9+12}{4} = \frac{21}{4} = 5/25$

---

①

$P_0 \to$ A $P_1 \to$ ۴ $P_2 \to$ ۳ $P_3 \to$ ۴ $P_4 \to$ ۲ $P_5 \to$ ۶ $P_6 \to$ ۴

در این سوال میزان بهره وری CPU برابر است با: $\frac{ECBT}{ECBT + EC.S}$ که برابر است با:

$\frac{۲۸}{۲۸ + n \times t(C.S)}$

اگر $t(RR)$ برابر ۴ باشد، ۸ بار C.S خواهیم داشت. ($n+2$ تا برای n تا اول و دوم می‌باشد)

اگر $t(RR)$ برابر ۵ باشد، ۷ بار C.S خواهیم داشت. ($n+1$)

اگر $t(RR)$ ۶ و یا بیشتر باشد ۶ بار C.S خواهیم داشت.

پس بهترین حالت در حالتی است که هر ۶ تا شریک یکبار C.S داشته باشیم.

از طرفی بهترین حالت در بین بیشتر از ۶ تا، ۶ تا بودن بهترین حالت ۶ است.

$\frac{۲۸}{۲۸ + ۶ \times t(C.S)}$

⑨ ارشد آزاد ۸۷

هر ۵۰ ثانیه ۱۰ فرایندی آید که هرکدام ۲ ثانیه زمان نیاز دارند ← همین دلیل کند زمان ۲×۱۰=۲۰ ثانیه CPU فعال است

زمان بیکاری سرویس = ۲ ثانیه

RR ، q = 100ms

$\frac{20}{50} = 0.4$

---

⑩ برای پیاده‌کردن حداقل زمان W.T از الگوریتم SJF استفاده می‌کنیم.

چون x گسترده از ۵ است پس CBT آن از سه فرایند دیگر گسترده است پس اول وارد می‌شود.

	CBT	T.T	
P_1	x	x	
P_2	۷	x+۱۲	
P_3	۵	x+۵	
P_4	۸	x+۲۵	

P_1 | P_3 | P_2 | P_4 |
x | x+۵ | x+۱۲ | x+۲۰

$T.T = \frac{4x+5+12+20}{4} \Rightarrow \frac{4x+37}{4} = 13/5 \Rightarrow x = 9/25$