process control Block (PCB) (مجموعه‌ای است از داده‌ها) . تا زمانی که ساختار داده‌ای وجود دارد فرایند هم است . از دید OS ، PCB نماینده فرایند است . خود OS این PCB را ایجاد می‌کند و وقتی حذف شد فرایند هم حذف می‌شود و منابع آزاد می‌شوند.

داخل PCB چی هست؟

- حالت فرایند Process state
- برنامه شمارنده program counter برای اینکه بفهمیم اجرای برنامه چی هست و وقتی اجرای برنامه‌ای که هم زمانی تمام شد باید بدانیم از کجا باید ادامه دهیم پس PC را در اینجا save می‌کنیم
- رجیسترها CPU registers | برای اینکه اجرای برنامه بعد از اینکه دوباره CPU را در اختیار گرفت
- اشاره‌گر پشته stack pointer | برنامه به درستی اجرا شود
- اطلاعات مدیریت حافظه Memory management information ← بیشتر مربوط به reg های دیده نشده‌اند
- اطلاعات زمان‌بندی پردازنده CPU scheduling Info ← باید این را داشته باشیم تا بفهمیم چگونه در صف قرار بگیرد
  اگر مهم باشد باید زودتر اجرا شود
  1. آیا اولویتی دارد
  2. درجه اولویت آن چی هست
- اطلاعات حسابرسی Accounting Info ← تا به حال چقدر از منابع استفاده کرده، چقدر زمان پردازنده را گرفته، چقدر از disk استفاده کرده

هر برنامه‌ای که پردازنده را در اختیار می‌گیرد اول باید اطلاعاتش در پردازنده load شود تا برنامه به درستی اجرا شود و وقتی هم که می‌خواهد پردازنده را از برنامه بگیرد اول اطلاعات جاری را در PCB ذخیره می‌کند. پس PCB در طول زمان تغییر می‌کند.

وقتی که PCB از بین می‌رود همه اطلاعات مربوط به یک برنامه از بین می‌رود. پس دیگر برنامه‌ای وجود ندارد پس لزومی ندارد که منابع را جداگانه آزاد کنیم چون دیگر کسی از آن منابع استفاده نمی‌کند پس از حذف PCB منابع آزاد شده حساب می‌شوند

point ← ایجاد فرایند = ساخت یک PCB
خروج از فرایند = حذف PCB

point) ایجاد فرآیند : گرفتن منابع از OS

- یک ساختار داده‌ای برای هر فرآیند ایجاد می‌شود.

← فرآیند اول چگونه ایجاد می‌شود؟

مجبوریم از حرف " فرآیند فقط توسط فرآیند ایجاد می‌شود " کوتاه بیاییم. فرآیند اول دستی یعنی توسط کاربر ایجاد می‌شود

(دستی ایجاد کردن فرآیند = روشن کردن سیستم ، reset کردن سیستم)

reset کردن یعنی چی: reset کردن یعنی به حالت معلوم بردن (چون اکثر مواقع حالت معلوم صفر است می‌گوییم صفر می‌شود). به حالت معلوم بردن یعنی pc و SP و reg ها و ... به یک عدد خاص (حالت معلوم) می‌رسند

PC = ...

SP = ...

reg = ...

← MMU = ... → memory management Unit

ساختن یک PCB
بصورت دستی

در واقع با reset کردن این‌ها داریم یک PCB درست می‌کنیم همه اطلاعات را برای PCB تعیین می‌کند و اطلاعات لازم را تامین می‌کند همیشه به حالت معلوم خاصی می‌روند و امکان ندارد در دو دفعه reset کردن دو تا مقدار متفاوت در داخل آنها برود

این فرآیند اول اصلاً جزئی از OS نیست چون این فرآیند پردازنده را در اختیار OS قرار می‌دهد و باعث ایجاد فرآیند OS می‌شود که در واقع جزئی از BIOS است. یعنی در واقع اول بخشی از OS را داخل حافظه قرار می‌دهد بعد از آن کنترل پردازنده را به OS می‌دهد بعد از این جمله " فرآیند توسط فرآیند ایجاد می‌شود " وفادار خواهیم بود

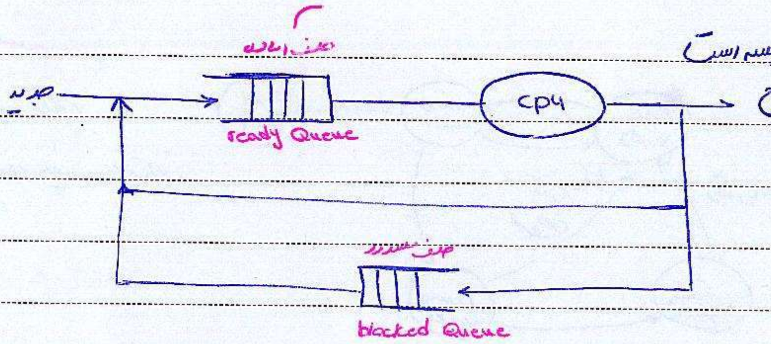
← OS هم یک فرآیند است که توسط PCB که اول ساخته شد create می‌شود.

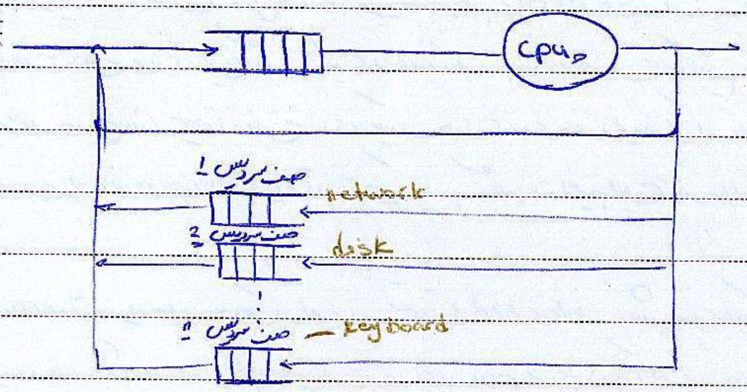
← PCB که اول به صورت دستی ساخته می‌شود همیشه یک کار ثابت انجام می‌دهد. OS را در جای خاصی از حافظه قرار می‌دهد و بنابراین همیشه آدرس شروع OS ثابت است

← این PCB همیشه ثابت است برخلاف PCB هایی که توسط OS ساخته می‌شود

← OS اطلاعات حیاتی خود را در جداول جداگانه دارد و در واقع PCB برای OS همین است و نیازی شدید به ساخت PCB ندارد. OS وقتی درخواست را می‌گیرد آن را نگاه می‌کند و حتی پردازنده را از دست می‌دهد پس نیازی آن چنانی به PCB نداریم.

دیاگرام فرایند: همان شکل اول جلسه است


صف آماده
جدید
ready Queue
CPU
خروج
صف مسدود
blocked Queue


CPU
صف سرویس 1
network
صف سرویس 2
disk
صف سرویس n
Keyboard

چون سرویس ها با هم تفاوت دارند یعنی یکی منتظر keyboard یکی منتظر port ها ... است برای هر کدام از این سرویس ها یک صف داریم. کسانی که سرویس یکسانی می خواهند باید با هم در یک صف باشند و معنی ندارد که کسانی که منتظر سرویس های مختلف هستند در یک صف باشند

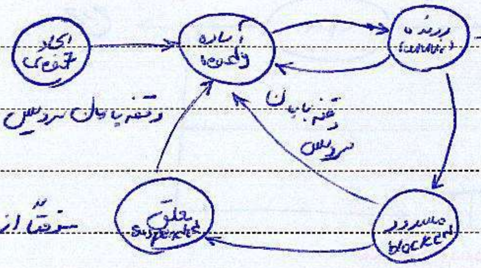
مدیریت و کنترل تمام این صف ها با OS است. کسانی که در صف هستند اصلاً نمی دانند در صف هستند فقط منتظر جواب درخواست خود هستند. چون پایان هر درخواست توسط وقفه است که پس هیچ گاه دو تا صف با هم جواب نمی دهند چون فقط یک وقفه در یک لحظه رسیدگی می شود.

فرایند شامل: کد، داده، پشته، PCB

وقتی می گوییم فرایند در صف است پس باید همه اینها در این صف باید قرار گیرند که این طور نیست در واقع PCB که نماینده فرایند است در صف است که تازه این هم که چون PCB بزرگ است تمام PCB در صف نمی رود بلکه فقط PC در صف باشد کافی است

Subject:
Year. Month. Date. ( )

جلسه هشتم:



موقتاً از حافظه خارج می‌شود.

تغییر شکل قبل برای اینکه فرآیندهای بیشتری بتوانند اجرا شوند

تعداد منابع برای فرآیند محدود است مثل حافظه. یک کاربر هم می‌خواهد هر چند تا برنامه می‌خواهد اجرا کند چون منابع محدود است در حالت معلق می‌رود یعنی اگر فرآیندی مسدود بود موقتاً بیرون می‌کنیم (از حافظه) تا زمانی که اطلاعات که خواسته به آن برسد. خارج شدن یعنی رفتن روی دیسک. چون روی هوا نمی‌توانند باشند بلکه باید در جایی ذخیره شود تمام تغییر حالت‌ها را باید بر اساس درخواست یا وقفه توضیح کنیم. اگر بتوانیم این کار را انجام دهیم هر چه می‌گوییم معتبر است

این شکل تا زمانی می‌تواند راه‌حل دهد که کلی در سمت blocked باشد برای اینکه فرآیندی در حالت انتظار نداریم و همه فرآیندها در حالت آماده‌اند. همچنین حافظه‌ای نداریم پس وقتی کلی در حالت blocked هستند نمی‌توانیم حافظه آن را بگیریم در نتیجه حافظه‌ای نداریم به فرآیند بدهیم.

← این راه‌حل برای حالتی است همواره فرآیندهای در حالت مسدود باشند

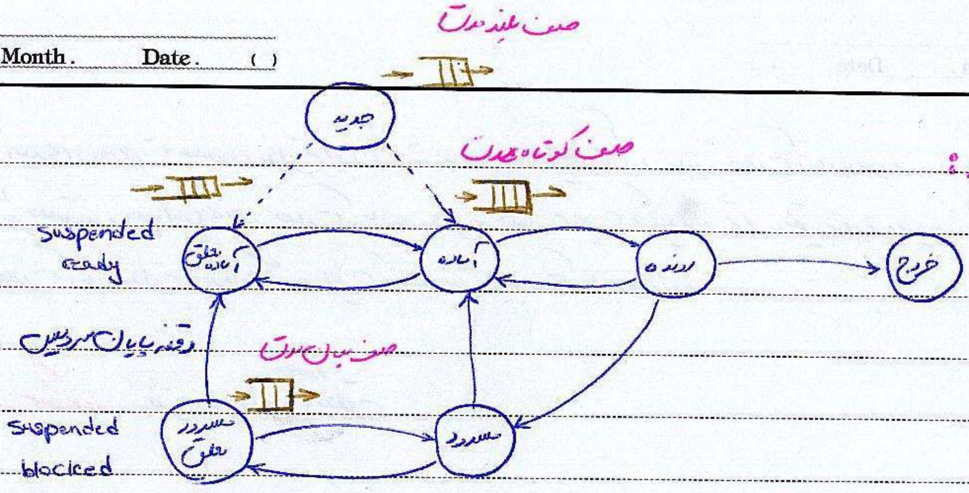
چه موقع از حالت مسدود به حالت معلق می‌رویم؟

- درخواست جدید
- وقفه پایان سرویس

← این شکل و بالا فرآیندهای بیشتری را در آن واحد اجرا می‌کند ولی همیشه جواب نمی‌دهد.

PAPCO

شکل کامل‌تر :



مسدود معلق ← وقتی فرآیند جدیدی می‌خواهد وارد سیستم شود... البته این نکته که می‌توانیم اگر حافظه نداشته باشیم از حالت مسدود به حالت معلق می‌آییم مهم

آماده معلق ← اگر در حالت مسدود چیزی نداشتیم از حالت آماده استفاده می‌کنیم و از حالت آماده یکی را معلق می‌کنیم اولویت با برای معلق کردن از حالت مسدود است اگر در حالت مسدود کسی نباشد از آخر صف حالت آماده استفاده می‌کنیم چیزهایی که در حالا حالاها نوبتشان نمی‌شود.

← هر چه فرآیند زیاد شود سرعت سیستم خیلی کم می‌شود. زمان بسیار زیادی می‌گیرد تا یک فرآیند را از حافظه به دیسک یا برعکس ببرد و حالا اگر تعداد حالات زیادی این کار را انجام دهد سیستم کند می‌شود و دائما صدای دیسک می‌آید حالا اینکه چقدر فرآیند می‌تواند اجرا شود تحمل سیستم است اینکه چقدر می‌توانند با سرعت کم کار کنند... OS نباید برای این کار محدودیت بگذارد اگر درست طراحی شده باشد مثل OS های microsoft که درست طراحی نشده‌اند اگر تعداد فرآیندها زیاد شود تحمل نمی‌کنند و crash می‌کنند

← اگر کسی را از حالت مسدود به حالت مسدود معلق می‌بریم امکان بازگشت به آن بدهیم و منتظر نمی‌شویم که حتما وقفه پایان سرویس بیاید. اگر حافظه خالی شد آن را به حافظه می‌بریم. اگر وقفه پایان سرویس آمد باید از حالت مسدود معلق به حالت آماده ببرد ولی چون در آماده معلق ممکن است چیزی باشد به آنجا می‌رود اگر نبرد بعد از آن به حالت آماده می‌برد. اگر کسی در حالت مسدود نباشد به سراغ حالت آماده می‌رویم از آن به حالت آماده معلق می‌فرستیم ولی امکان بازگشت به آن می‌دهیم

اگر فرآیندی در حالت مسدود معلق بود و وقفه پایان پردازش آمد باید به حالت آماده ببرد و در حالت آماده هم کسانی منتظر هستند و هم لیستی از حالت آماده معلق چون این فرآیند فرقی با بقیه ندارد باید منتظر بماند پس اول به حالت آماده معلق می‌رود اگر در این حالتی نبود به حالت آماده می‌رود.

توجیه این شکل : رفتن از حالت مسدود معلق

مهم چند شکل) صف‌های متعددی می‌توانند در شکل موجود باشند حتی در فرآیند جدید ممکن است چند فرآیند با هم create شوند پس درخواست‌ها در صف قرار می‌گیرند و به نوبت به آنها رسیدگی می‌شود. چه سیاستی را برای این صف‌بندی پیش بگیریم؟ (الگوریتم صف‌بندی، زمان‌بندی، نوبت‌بندی، scheduling)

point برای اینکه از حالت مسدود به مسدود معلق ببریم هم بهتر است صف باشد چون اگر نباشد یعنی اینکه random انتخاب کردیم ولی باید بر اساس برنامه مشخص باشد

صفی که در حالت آماده است صفی مهم است و تأثیر زیادی روی کارایی سیستم و OS دارد که به آن صف کوتاه مدت می‌گویند (short term queue)

از دیگر صف‌ها صف مسدود معلق ← صف میان مدت mid term queue
" " " ایجاد ← " بلند مدت long term queue

اسم‌گذاری این صف‌ها بر اساس مدت زمانی است که به صف سرکشی می‌کنیم

short term queue ← در زمان کوتاه (نزدیک) به آنها سرکشی می‌کنیم با فواصل زمانی کوتاه به آن سرکشی می‌کنیم

mid term queue ← در زمان‌های متوسطی به آن‌ها سرکشی می‌کنیم با فواصل طولانی‌تر نسبت به کوتاه مدت

long term queue ← بعد مدت زیاد به آنها سرکشی می‌کنیم

صف کوتاه مدت: فرآیند در حالت رونده، بیشتر از برش زمانی نمی‌ماند چون برش زمانی کوتاه است پس مرتب به آن سر می‌زنیم. اگر برش زمانی باشد حداکثر زمان برش زمانی طول می‌کشد اگر برش زمانی نباشد اجرا پیوسته است و خیلی تا زمانی است که یک درخواست برسد البته حالت‌هایی است که کوتاه‌تر نباشد ولی به صورت میانگین کوتاه است.

برای مدت‌های بیشتر بازای هر بار خالی شدن پردازنده برنامه‌هایی را در صورتی که در صف کوتاه مدت پس از هر بار خالی شدن پردازنده به برنامه آن می‌دهیم

چرا به صف ایجاد صف بلندمدت می‌کنیم؟

اگر دائماً داخل منابع دچار مشکل بشویم قاعدتاً چرخه را روی کسانی که می‌خواهند وارد سیستم بشوند می‌گذاریم به جای که دارند کار می‌کنند کار نداریم و فرایندهایی که می‌خواهند create بشوند را معطل می‌کنیم. در واقع اصلاً صف ایجاد وجود ندارد اگر دچار نشده باشیم یعنی در منابع کمبودی نداشته باشیم صف ورودی نداریم ولی اگر صفی باشد باید سیستم از هم تحمل کند.

سیاست‌ها: معمولاً برای صف کوتاه مدت سیاست‌های زمان‌بندی به کار برده و برای صف‌های بلندمدت و میان‌مدت از این سیاست‌ها استفاده می‌کنند چون زیاد روی سیستم تاثیر ندارد منطقی‌ترین سیاست صف‌بندی برای صف‌های بلندمدت و میان‌مدت از سیاست FIFO استفاده می‌کنند (منطق صف) چون مراجعه به آنها کم است پس تاثیری زیاد روی سیستم ندارند

میان‌مدت | FIFO
بلندمدت | FCFS First Come First Served

برای اینکه پیاده‌سازی سیاست FIFO ساده است به خودمان برای طراحی صف پیچیده زحمتی نمی‌دهیم چون تاثیری روی کارایی سیستم ندارد از همین سیاست ساده استفاده می‌کنیم چون مراجعه به صف کم است بهینه کردن آن فایده‌ای ندارد

چون که ما زیاد به صف‌های کوتاه‌مدت سر می‌زنیم پس بهینه کردن این صف روی کارایی سیستم تاثیر دارد پس روی بهینه کردن این صف تمرکز می‌کنیم

الگوریتم‌های زمان‌بندی پردازنده:

- non preamptive (بدون قبضه)
- preemtive (با قبضه)

non preemptive ـ پردازنده را نمی‌توان گرفت بلکه باید خودش پردازنده را پس بدهد سهم زمانی در این الگوریتم‌ها نمی‌دهند

preemptive ـ پردازنده را می‌گیرند یعنی قبضه می‌کنند

point ← این فایده کم زمانی برای کاربر است چون که هر کاربر که الان نتیجه کار خود را بگیرد. می‌بینیم همه سیستم‌ها یک کاربره نیستند است مثل ماهواره‌ها. اکثر سیستم‌های کامپیوتری کاربر ندارند. مهم‌ترین گروه سیستم‌های کامپیوتری embeded است ← پس در نتیجه مهم است بسیاری از OSهای گروه non-preemptive می‌باشند پس می‌توان گفت این الگوریتم‌ها کم استفاده است.

← OSهایی که در گروه non preemptive هستند نمی‌توان پردازنده را از آنها گرفت پس نمی‌توان از حالت رانینگ به حالت آماده رفت. ولی OSهایی که در گروه preemptive هستند در آنها می‌توان از حالت رانینگ به حالت آماده رفت

معیارهای مقایسه روش‌های زمان‌بندی:

۱- زمان انتظار (waiting time): از زمانی که فرایند وارد می‌شود چه مدت در حالت انتظار بوده و پردازنده را در اختیار نداشته تا زمانی که خارج شود. یکی از معیارها کاهش زمان انتظار است.

کل زمان‌هایی که پردازنده را در اختیار نداشته

2. زمان کل (Turnaround time): کل زمان running، انتظار و ... را با هم جمع کن یعنی از زمانی که وارد می‌شود تا زمانی که خارج می‌شود چقدر طول می‌کشد که کاهش باید یابد. اگر زمان انتظار کم شد این هم خود به خود کم می‌شود

3. زمان پاسخ (response time): از زمانی که وارد سیستم می‌شود تا زمانی که اولین بار پردازنده را در اختیار بگیرد به مدت زمان لازم برای اولین سرویس پردازنده. کسانی هستند که از ما جلوتر باشند پس تا زمانی که پردازنده را در اختیار بگیریم مدتی طول می‌کشد

point) همیشه ممکن است همه بهبود یابند بعضی تغییرات تاثیری روی این اشکال ندارد باید ببینیم برای کاربر خاص کدام از آنها مهم است

۴. استفاده از پردازنده (processor utilization): کاری کنیم که پردازنده اکثر اوقات مشغول باشد و کاری را انجام دهد. این استفاده در سوپرکامپیوتر استفاده از پردازنده در حالت‌ها تا حدی قابل قبول است باید اولویت اساسی هست.

B

گذردهی
5_ توان عملیاتی (Throughput): اصولاً Throughput یعنی ما می‌خواهیم مقدار کارهای انجام شده در واحد زمان بالا برود. به اینکه چقدر استفاده کردیم، یا چقدر کارکردی یکبارت چیست برایش مهم نیست، مهم این است که مقدار کارهای بیشتری انجام دهد. این زیاد برای سیستم‌های interactive مهم نیست.

← می‌توان از زمینه‌های دیگری را هم لیست کرد مثل: عدالت :D

جلسه هفتم: 7,21

الگوریتم‌های زمان‌بندی پردازنده: (Process Scheduling)

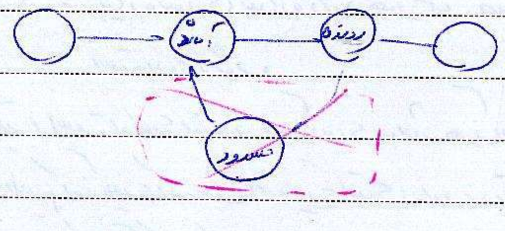
- بدون قبضه non-preemptive ← اینقدر می‌رود تا خودش پردازنده را از دست بدهد
- با قبضه preemptive ← پردازنده را ازش می‌گیریم

1_ FCFS ← روش‌های بدون قبضه

مثال)

فرایندهای	A	B	C	D	E
به سیستم وارد می‌شوند ← زمان ورود	0	2	4	6	8
زمان سرویس (Ts)	3	6	4	5	2
زمانی که در سیستم بوده ← زمان کل (Tr) زمان پایان - زمان ورود	3	7	9	12	12

روشی که استفاده می‌کنیم FCFS است یعنی هر که زودتر وارد شد زودتر سرویس می‌گیرد. (مثال خیلی ساده است یعنی با یکبار سرویس گرفتن کار اینها تمام می‌شود) یعنی بخش مسدود حذف می‌شود.



زمان پایان	3	9	13	18	20

A در زمان صفر وارد می‌شود و چون کسی پردازنده را ندارد کارش را انجام می‌دهد و بعد از 3 واحد زمان رها می‌کند چون کارش تمام شده. B در 2 وارد می‌شود ولی چون تا 3 پردازنده در دست A است باید صبر کند تا کار A تمام شود. وقتی که C وارد می‌شود باید صبر کند تا کار B تمام شد یعنی در 9 کارش شروع می‌شود با اینکه در 4 وارد شده است.

فاکتورگیری تری می‌کنیم (T_T/T_s)

	A	B	C	D	E	میانگین
(T_T/T_s)	1	1.17	2.25	2.40	6.0	2.56

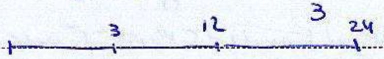
بهترین رژیم این نسبت به ۱ نزدیک بشود این نشان می‌دهد زمان انتظار در سیستم پایین است اگر زمان انتظار کم باشد این عدد به یک نزدیک می‌شود اگر از این اعداد میانگین بگیریم هرچه به یک نزدیک‌تر باشد بهتر است

← مشکل این روش (FCFS) با مثال زیر معلوم می‌شوند

T_T	زمان سرویس (Ts)	ورود	
3	3	0	A
12	9	0	B
24	12	0	C

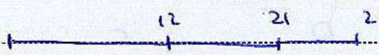
ترتیب ورود ↓

میانگین (T_T) ← $\frac{3+12+24}{3} = 13$



ترتیبشان را عوض می‌کنیم C→B→A

$\frac{24+21+12}{3} = 19$



اولی بهتر است ترتیب، چون زودتر سرویس می‌دهد و زودتر فرایندها را بیرون می‌فرستد

در نتیجه چون این ترتیب‌ها تصادفی است پس می‌گوییم جواب این روش به تصادف بستگی دارد و تضمینی برای خوب یا بد بودن زمان وجود ندارد

اشکال ← تضمین وجود ندارد

خوبی ← پیاده‌سازی ساده دارد چون یک صف ساده است، به همین دلیل خیلی وقت‌ها از آن استفاده می‌شود.

این روش به ترتیب است و کاری ندارد چه فرایندی باشد هر فرایندی که اول آمده باشد زودتر سرویس می‌گیرد اگر بتوانیم کاری کنیم که فرایندی که زمان سرویس کمتری دارد زودتر انجام شود زمان کمتری خواهیم داشت که این قابل اثبات است یعنی اگر به آنکه کوتاه‌تر است زودتر سرویس دهیم میانگین کمتری خواهیم داشت.

2- SPN (Shortest Process Next) (ابطال ناپذیر) $1/T_s$

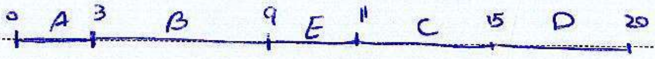
این روش بهینه است

در مثال قبلی این روش را استفاده می‌کنیم

میانگین	E	D	C	B	A	
	8	6	4	2	0	زمان ورود
	2	5	4	6	3	زمان سرویس
	11	20	15	9	3	زمان پایان
7.60	3	14	11	7	3	T_r
1.79	1.5	2.8	2.75	1.17	1	T_r/T_s

در لحظه 0 فقط A است و رقابتی وجود ندارد در زمان 3 فقط B است پس باز هم رقابتی وجود ندارد در لحظه 9 دو سه فرآیند جدید وارد سیستم می‌شوند در اینجا باید کوتاه‌ترین را انتخاب کنیم و به همین ترتیب

در اینجا انتظار را داریم میانگین‌ها از روش قبلی بهتر باشند. همان طور که انتظار داشتیم کمتر شد و به روشی رسیدیم که جواب بهتری از قبلی دارد.



زمان سرویس از کجا معلوم است؟ معلوم نیست (نمی‌پرسیم CPU از کجا می‌داند بلکه باید بپرسیم OS از کجا می‌داند)

اشکالات ← 1- از آنجا که زمان سرویس را قبل از اجرا شدن نمی‌دانیم پس زمان بهینه به راحتی به دست نمی‌آید و دشواری دارد و روش های برای تخمین وجود دارد که ممکن است از واقعیت دور باشد. زمان سرویس را نمی‌دانیم خودش [illegible] می‌توانیم حدس بزنیم و تخمین !

2- حالتی پیش می‌آید که فرآیندی زمان سرویس طولانی دارد و از بدشانسی آن همیشه فرآیندی با زمان سرویس کوتاه‌تر وجود دارد و همیشه این فرآیند عقب می‌افتد و این امکان وجود دارد که نوبت به این نرسد اصلاً. به این حالت گرسنگی یا starvation. احتمال غیرصفر برای این وجود دارد که هرگز سرویسی به فرآیند داده نشود.

Subject:
Year. Month. Date. ( )

point) اصولاً الگوریتم های اولویت داری می توانند به گرسنگی منجر بشوند

سعی می کنیم مشکلات را حل کنیم. راه حل دوم مشکل است و اینست که الگوریتم را عوض کنیم

حل مشکل اول:

روش هایی برای تخمین وجود دارد که با این حالت سربار سیستم زیاد می شود.

برای تخمین زمان سرویس

متوسط گیری نمایی: ← $S_{n+1} = \alpha T_n + (1-\alpha) S_n$

واقعیت اینست که یک فرآیند بارها سرویس می گیرد و یک بار سرویس گرفتن کارش تمام نمی شود. n می گوید چندمین بار است که سرویس می گیرد. این تخمین را با استفاده از سابقه فرآیند می زنیم

α ← $0 < \alpha < 1$

S ← تخمین

T ← زمان واقعی

به این علت می گوییم نمایی که می شود این رابطه را داد.

$$S_n = \alpha T_{n-1} + (1-\alpha) S_{n-1}$$

برای بسط از فرمول اصلی استفاده می کنیم برای اینکه بهتر بفهمیم چی می گوید آن را بسط می دهیم

$$S_{n+1} = \alpha T_n + (1-\alpha)\alpha T_{n-1} + \dots + (1-\alpha)^i \alpha T_{n-i} + \dots + (1-\alpha)^n S_1$$

توان باعث می شود اهمیت اعدادی که دورتر از یک کم می شود این بسط تمام تاریخچه فرآیند را دارد. رفتارهایی که اخیراً انجام داده اهمیت بیشتری دارد تا رفتارهای گذشته. می گوید رفتار الانش چه ربطی به گذشته اش داشته است. α چگونه گرفته شود الان هم می شود.

هر چه α را به یک نزدیک کنیم ← رفتارهای اخیر تأثیرشان خیلی زیاد می شود. رفتارهای قدیمی ارزشی نخواهند داشت یعنی فقط به رفتارهای اخیر نگاه می کند

هر چه α را به صفر نزدیک کنیم ← کل تاریخچه را در نظر می گیرد. قدیمی ها را هم نگاه می کند

کسانی که این کار را می‌کنند باید عملی نگاه کنند که هیچ بستگی نداشته باشد پس بهتر است $\alpha = 1/2$

با این می‌بینیم خیلی ارزان می‌شود یعنی محاسبات floating point لازم نیست که اگر این می‌شد (floating point) با الگوریتم خیلی زمان بر می‌شود

$\alpha = 1/2$ ← هم مقدارها را در نظر می‌گیرد

که محاسبات تنصیف shift راست زدن بیش نیست

← مشکل اولی را نتوانستیم حل کنیم با اینکه خیلی دقیق نتوانستیم

حل مشکل دومی:

این مشکل را گارانتی نمی‌توان کرد پس اصلاً از این الگوریتم استفاده نمی‌کنیم بلکه این الگوریتم را اصلاح می‌کنیم پس:

3- HRRN Highest Response Ration Next

چون می‌خواهد با حالت گرسنگی مقابله کند جوابی به خوبی SPN نخواهد داشت و چون دارد چیزی را اصلاح می‌کند پس باید هزینه‌ای بدهد

از یک نسبت صحبت می‌کند علاوه بر Ts، زمان انتظار را در نظر می‌گیرد، به انتظار امتیازی می‌دهد در SPN پردازش اصلاً هم نبود که چقدر انتظار کشیدی

امتیاز برای انتظار T_w / T_s

هر چه بیشتر انتظار بکشی امتیازت بالاتر می‌رود، هر چه زمان کوتاه‌تر باشد باز هم امتیازت بالاتر می‌رود

← برای مثال قبل)

	A	B	C	D	E	میانگین
		9	5/4	3/5		
زمان ورود	0	2	4	6	8	
Ts	3	6	4	5	2	
زمان پایان	3	9	13	20	15	
زمان کل (T_r)	3	7	9	14	7	8.0
(T_r/T_s)	1	1.17	2.25	2.8	3.5	2.14

0 A 3 B 9 C 13 E 15 D 20

C 5/4
D 3/5
E 1/2

D 7/5
E 5/2

نسبت T_w/T_s ←

سر T_w/T_s

Subject:
Year. Month. Date. ( )

نمی‌بینیم پس کاراییش به خوبی قبلی باشد چون می‌خواهیم گرسنگی را از بین ببریم پس باید هزینه‌ای بدهیم
انتظار داشتیم که از FCFS بهتر بشود چونکه FCFS رقابتی است و می‌توانند از بین بهتر بشود.

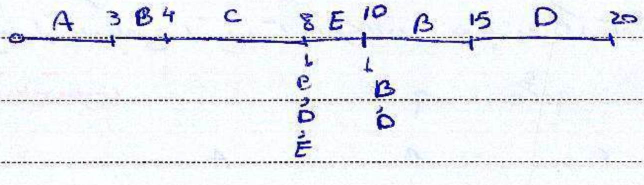
اگر بخواهیم از روش های non-preemptive استفاده کنیم انتخاب بهترینش و بدترینش FCFS است
چون SPN که مشکل گرسنگی دارد و HRRN هم دارای محاسبات زیاد و سربار بالای باشد.

روش های preemptive :

4_ SRT Shortest Remaining Time : (با قبضه)
در هر لحظه می‌توان فرآیند را قطع کرد. همین الگوریتم SPN را اجرا می‌کند اما در هر لحظه اجرا می‌کند

A	B	C	D	E
0	2	4	6	8
3	6	4	5	2

در اینجا در لحظه 2، چه اتفاقی خواهیم داشت چون می‌توان پردازنده را از فرآیند گرفت چون A کوتاه‌تر است
اتفاقی نمی‌افتد. در لحظه 3، B را داریم و در لحظه چهار هم B و C که B چون یک واحدش انجام شده 5 تا مانده ولی
چون Ts برای C کمتر از 5 است پس
B قطع می‌شود و C اجرا می‌شود.



انتظار داریم از SPN هم بهتر شود
در هر لحظه SPN را اجرا می‌کند و آن هایی که زمانش کمتر است را اجرا می‌کند

جلسه هفتم: 7,23

5- روش تقسیم زمانی (پیش‌دستانه) RR (Round Robin)

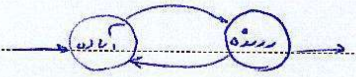
	A	B	C	D	E
زمان ورود	0	2	4	6	8
زمان سرویس	3	6	4	5	2

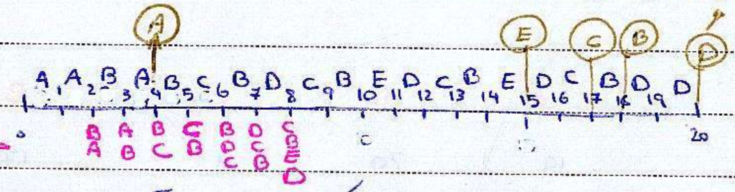
در دو حالت بررسی می‌کنیم ← q=1 ← زمان نسبتاً کوچکی است

q= ← زمانی که از بعضی از اینها بزرگتر است

q=1

زمان تکمیل شدن


آماده
رونده


A B A C B E D
A A B A B C B D C B E D C B E D C B D D
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
محتویات صف ←
B A
A B
B C
C B
B D C
D C B
C B E D

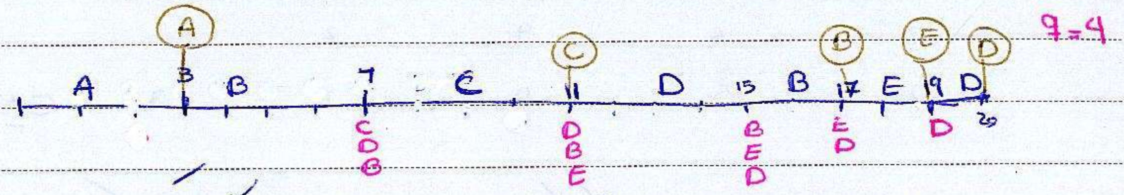
در زمان 0، فقط A است و سهم خود را می‌گیرد و به حالت آماده می‌رود، می‌بیند در صف کسی نیست پس دوباره به حالت رونده می‌رود یک واحد دیگر کارش را انجام می‌دهد. در لحظه‌ی 2، B هم آمده و این قرار را گذاریم که در این حالت B سر صف باشد و چون A همین الان سرویس گرفته پس B سرویس می‌گیرد (خاصیت نوبت زمانی عادلانه بودن آن است) اگر فرآیند وارد شود و در همان لحظه فرآیندی از حالت رونده به حالت آماده‌ی خواهد برود اول فرآیند جدید در صف قرار می‌گیرد.

	A	B	C	D	E	میانگین
زمان پایان	4	18	17	20	15	
زمان کل T_r	4	16	13	14	7	10.80
T_r/T_s	1.33	2.67	3.25	2.8	3.80	2.71

در مقایسه با اعداد پیش خیلی بدتر است ← طبیعی است چون زمان بیشتری را در سیستم صرف کرده و همین که هی باید به صف آماده برگردد زمان بیشتری می‌خواهد.

اما در کل زمان پاسخ این روش بهتر است یعنی اولین باری که سرویس می‌دهد مثلاً در روش های قبلی هیچ وقت نمی‌توانست در زمان 5، به C نوبت بدهد و هم‌چنین در این روش گرسنگی وجود ندارد.

کارهای interactive به دنبال زمان پاسخ هستیم و همین که به زمان پاسخ این روش خیلی خوب است
این روش به بدی خودش در تعداد زمان پایان و زمان کل نباید ما را گول بزند



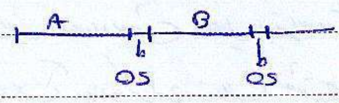
A زمان 4 را نیاز ندارد چون در 3 زمان کارش تمام می‌شود در این لحظه به صف نگاه می‌کنیم چون در صف تنها B است 4 تای B را به آن می‌دهیم. در حالی که B مشغول بوده D و C وارد شدند و 4 تای C را به آن می‌دهیم و C تمام می‌شود

	A	B	C	D	E	
زمان پایان	3	17	11	20	19	
زمان کل T_T	3	15	7	14	11	10
T_T/T_S	1	2.5	1.75	2.80	5.50	2.71

اگر باز هم به زمان پاسخ نگاه می‌کنیم می‌بینیم در حالی که q=1 است خیلی کمتر است. اگر q را بزرگ بگیریم همان FCFS می‌شود یعنی به کم زمانی اصلاً توجه نکردیم

ادعا) اگر quantum کوچک شود شبیه SPN می‌شود که هم در SPN فرآیند کوتاه‌تر زودتر تمام می‌شود و در این جا هم همین طور است ولی دلیلی ندارد که همیشه یکسان شود چون از یک الگوریتم استفاده نمی‌کنند

ما نمی‌توانیم q را خیلی کوچک کنیم چون شکل این طوری نیست بلکه دقیقاً به شکل زیر است

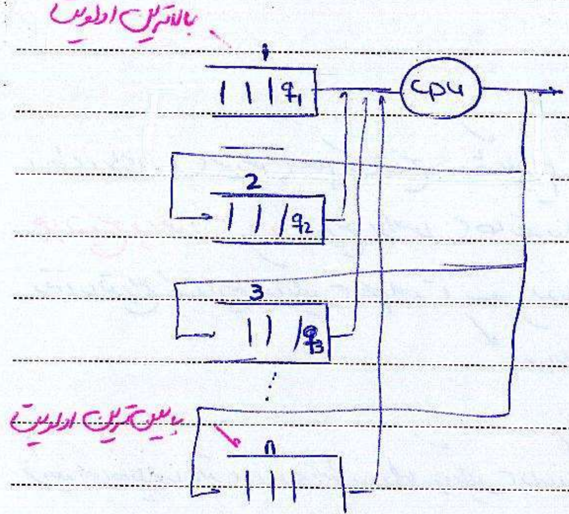


در این مثال از زمان‌هایی که OS کار می‌کند صرف نظر می‌کنیم
OS زمان بندی را انجام می‌دهد نه CPU و به صورت فرآیند این کار را می‌کند اگر چه کوچک است ولی همیشه نمی‌توان از آن صرف نظر کرد اگر q خیلی کوچک باشد مجبور هستیم زمان OS را هم به حساب بیاوریم.

در هر زمانی OS قبل از اینکه پردازنده را به فرآیند بدهد Timer را صفر می‌کند تا کل زمان‌ها به درستی محاسبه شود.

1 point q باید طوری انتخاب بشود که 10 برابر زمانی باشد که در OS نیاز دارد ولی اگر خیلی بزرگ باشد شبیه FCFS می‌شود که خوب نیست ولی هرچه کوچکتر باشد عادلانه‌تر است و شبیه SPN می‌شود ولی به سیستم محدودیت دارد.

روش Feedback Queue : (بازخورد)



در این روش به جای یک صف از چندین صف استفاده می‌کند کسی که وارد می‌شود وارد صف اول می‌شود و یک q دارد که اگر کافی نباشد به صف دوم می‌رود که در صف دوم q می‌تواند همان قبلی باشد یا نباشد اولویت با صف اول است و تا زمانی که فرآیندی در صف اول است نوبت به صف دوم نمی‌رسد. احتمال اینکه کسی در صف اول نباشد زیاد است چون برگشت به صف اول نداریم اگر در صف دوم کارش انجام شد چه بهتر اگر انجام نشد به صف سوم می‌رود و می‌تواند q دیگری داشته باشد

صف آخر دیگر نمی‌تواند q داشته باشد چون اگر q آن تمام شود کجا می‌برد صف دیگری نداریم که پس باید صبر کند تا این فرآیند به طور کامل انجام شود

این روش شبیه الگوریتم SPN را اجرا می‌کند چون اولویت با کوتاه‌ترها است و اگر فرآیندی طولانی باشد ممکن است دچار حالت گرسنگی بشود بدون اینکه زمان سرویس را بداند دارد SPN را شبیه‌سازی می‌کند

هرچه صف بیشتر بشود وظیفه OS بیشتر می‌شود

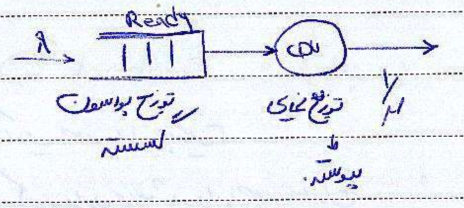
این صف جزء پارامترهای OS هستند و در کنار OS ذخیره می‌شوند.

← تا حالا بهتر بودن روش‌ها را با مثال نشان دادیم ولی باید به طور عملی اثبات کنیم. برای اثبات از چندین روش استفاده می‌کنیم ولی الان ساده‌ترین نوع آن گفته می‌شود.

الان با روش آماری کار می‌کنیم ولی این بسیار پیچیده است و فقط اساسش را اثبات می‌کنیم

Subject:
Year. Month. Date. ( )

یک صف داشتیم و پردازنده‌ای که به آن خدمت می‌کرد. بسته به اینکه چه توزیع آماری و چه روشی را برای این درنظر بگیریم تحت بررسی می‌شود. پس روشی که انتخاب می‌کنیم FCFS است. حالا ساده‌ترین توزیع عملی را بررسی داریم



از طرفی برای زمان سرویس هم یک توزیع در نظر بگیریم ← توزیع نمایی

چرا توزیع پواسون ← توزیع پواسون حافظه ندارد و هیچ اطلاعی ندارد قبلاً چه اتفاقی افتاده است برایش مهم نیست

وقتی با توزیع کار داریم میانگین هم است ← در واحد زمان چندتا فرآیند وارد صف می‌شوند (λ میانگین تعداد)

↙ میانگین زمان سرویس به فرآیند چقدر است ($1/\mu$ میانگین زمان)

فرآیندها خیلی نامرتب وارد می‌شوند ولی میانگین دارند. چون نامرتب بودن آنها را همان توزیع مشخص می‌کنند.

رویداد ما ورود فرآیند است که گسسته است (1، 2، 3، ...) ← یعنی 2.5 تا فرآیند نداریم ولی 5 تا فرآیند داریم

زمان بین ورود فرآیندها چطور است؟ عدد گسسته ندارد بلکه پیوسته است. توزیع آن پیوسته است و تفاوت زمان‌ها را توزیع آن مشخص می‌کند

میانگین ورود ↑

$$P(k) = (\lambda T)^k e^{-\lambda T} / k! \qquad k = 0, 1, \ldots$$

← احتمال ورود k فرآیند در فاصله زمانی T

بدست آوردن میانگین این توزیع:

$$E(k) = \sum_{k=0}^{\infty} k P(k) = \lambda T$$

واریانس: $\sigma^2 = \lambda T$ ← چه مقدار از میانگین دور یا نزدیک هستیم

← کم ← خیلی به میانگین نزدیک، زیاد ← از میانگین دور هستیم

توزیع نمایی:

τ : زمان سرویس

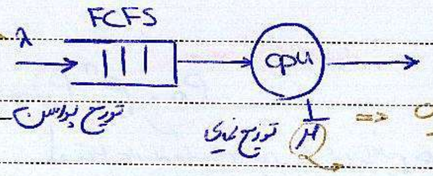
$f(\tau) = \mu e^{-\mu\tau}$

اگر برای رویدادها توزیع پواسون داشته باشیم زمان بین آنها توزیع نمایی است.

$E(\tau) = \int_0^\infty \tau f(\tau)\, d\tau = \frac{1}{\mu}$ → میانگین

$\sigma_\tau^2 = \frac{1}{\mu^2}$ → واریانس

جلسه یازدهم 28/7/



میانگین ورود ← λ

توزیع پواسن ← حافظه ندارد در اینکه قبلاً چه اتفاقی افتاده گذشته

توزیع نمایی $\frac{1}{\mu}$ ⇐ میانگین زمان سرویس

μ = تعداد میانگین سرویس

ρ بهره‌وری $= \frac{\lambda}{\mu}$ ⇐ مقدار با زمان نسبت عکس دارد.

← توانایی سرویس دادن

توانایی سرویس دهی باید از تعداد ورود باید بیشتر باشد ⇐ $\lambda < \mu$

در غیر اینصورت صف به سمت بی‌نهایت میرود حتی مسدود هم می‌تواند باشد.

به هر حال زمان های بیکاری خواهد داشت و همیشه مشغول کار نیست ⇐ $\rho < 1$

← نسبت ورود به نسبت توانایی سرویس

مشخصات آماری یک صف محدود

← ظرفیت مشخصی دارد که نمی‌تواند تا بی‌نهایت ادامه پیدا کند

میانگین طول صف

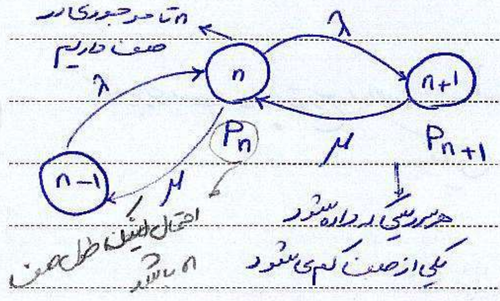
- میانگین زمان انتظار

- " " کل

- احتمال مسدود شدن صف

چند تا کار در واحد زمان

- میانگین توان عملیاتی (Throughput)


n تا مشتری در صف داریم
λ
λ
n
n+1
n−1
P_n
P_{n+1}
μ
μ
هر مشتری که پردازش شود یکی از صف کم می‌شود
احتمال اینکه طول صف n باشد

صف می‌تواند از حالت n به حالت n+1 برود

تغییر چندتایی انجام نمی‌دهد بلکه تغییر حالت یکی یکی است.

تعادل برقرار است ← $\lambda P_n = \mu P_{n+1}$

خاصیت توزیع پواسن در این ⇐ $P_n = \rho P_{n-1}$ یا $P_{n+1} = \rho P_n$ ⇐ با استفاده از تعریف ρ

روابط به راحتی نوشته می‌شود.

عدد ثابتی است.

احتمال خالی بودن صف ⇐ $P_n = \rho^n \boxed{P_0}$

← قابل اندازه‌گیری نیست، نمی‌دانیم چقدر است

گفتیم صف محدود است اما برای سادگی مسئله حرفمان را پس می‌گیریم پس صف را نامحدود در نظر می‌گیریم

$$\sum_{i=0}^{\infty} P_i = 1 \Rightarrow \sum_{i=0}^{\infty} \rho^i P_0 = P_0 \underbrace{\sum_{i=0}^{\infty} \rho^i}_{\text{تصاعد هندسی}} = \frac{P_0}{1-\rho} = 1 \Rightarrow \underline{P_0 = 1-\rho}$$

$$E(n) = \sum_{i=0}^{\infty} i P_i = \frac{\rho}{1-\rho}$$ ⇐ میانگین طول صف نامحدود

اینها برای صف نامحدود بود حالا می‌رویم سراغ صف محدود (طولش را N می‌گیریم). هرجا که بی‌نهایت (∞) استفاده کردیم حالا محدودش می‌کنیم و N قرار می‌دهیم و نتایج زیر به دست می‌آید.

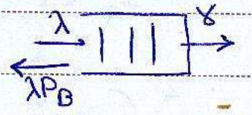
$P_0 = (1-\rho) / (1-\rho^{N+1})$

$P_n = (1-\rho)^n / (1-\rho^{N+1})$

$P_N = (1-\rho)\rho^N / (1-\rho)^{N+1}$ ⇐ احتمال اینکه صف پر شود

چون وقتی که صف پر شود بقیه فرایندها block می‌شوند ⇝ $P_B = P_N$

وقتی که صف پر شود فرایندهای جدید که وارد می‌شوند پذیرفته نخواهند شد.



توان عملیاتی ← $\gamma = \lambda (1 - P_B)$

* به خاطر توزیع های انتخاب شده تمام اینها از تحلیل دور است

یک رابطه ای مستقل از نوع توزیع است و همیشه صادق است.

n طول صف

w زمان انتظار

$\Rightarrow n = w \cdot \lambda$ ⇝ رابطه لیتل

از روی این رابطه به راحتی میانگین زمان انتظار پیدا می‌شود:

$$w = \frac{n}{\lambda}$$

$$\text{زمان کل} = w + \frac{1}{\lambda}$$

μ ← میانگین: پردازنده برای سرویس دادن است پس از این استفاده نمی‌کنیم و از آن که زمان پردازش می‌کند استفاده می‌کنیم

تا زمانی فرایند وارد صف می‌شود که صف پر نشده باشد و λP_B تا از آنها رد خواهند شد

← احتمال block شدن

تمرین) از زمان‌بندی HRRN استفاده کنیم

اگر کسی منتظر شد بیشتر الویت می‌دهیم $1 + \frac{t_w}{t_s}$

از این رابطه در حالت کلی استفاده می‌کنیم ولی اینجا وارد می‌کنیم

	زمان ورود	زمان سرویس
P_1	t_1	r_1
P_2	t_2	r_2
P_3	t_3	r_3

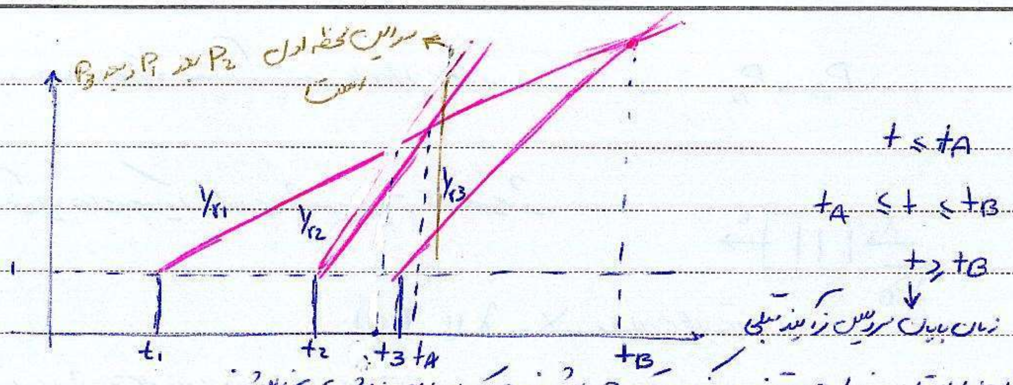
ترتیب اجرای فرایندها:

$$t_1 < t_2 < t_3$$

$$r_2 < r_3 < r_1$$

$1 + \frac{t_w}{t_s}$ ← در صورت نتایج تعیین اولویت است

Subject:
Year. Month. Date. ( )



توابع اولویت سه خط هستند چونکه با یک جمع شدند هیچ کدام از صفر شروع نمی‌شوند

ملایم‌ترین شیب ← $R_1 = 1 + \frac{t_w}{t_s} = 1 + \frac{t - t_1}{r_1}$ زمان انتظار است مقدار زمانی که از شروع کرده منهای زمانی که وارد شده (در این زمان وارد شده)

تیزترین شیب ← $R_2 = 1 + \frac{t - t_2}{r_2}$

← $R_3 = 1 + \frac{t - t_3}{r_3}$

در نمودار در یک لحظه اولویت با کسی است که بالاتر است یعنی شیب بیشتری دارد.

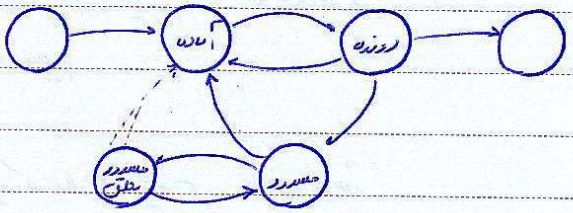
$t \le t_A \rightsquigarrow \overrightarrow{P_1, P_2, P_3}$

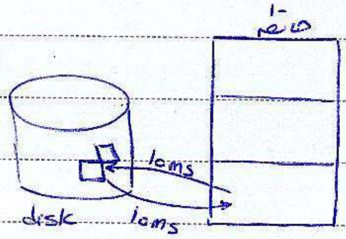
$t_A < t \le t_B \rightsquigarrow \overrightarrow{P_2, P_1, P_3}$

$t_B < t \rightsquigarrow \overrightarrow{P_2, P_3, P_1}$ درصورتی که تمام نشده باشند ادامه پیدا می‌کنند

اگر تمام شده باشند پس فرآیند تمام شده از این ترتیب حذف می‌شود.

تمرین) حجم‌زدایی swaping





حافظه این سیستم partition ثابت دارد یعنی در حافظه 3 تا فرآیند بیشتر نمی‌توانند در حافظه باشند. اگر بیشتر شد باید به حالت مسدود معلق بروند

برای وارد کردن فرآیند جدید یکی را از حافظه به دیسک ببریم و فرآیند را از دیسک به حافظه بیاریم.

زمان دسترسی به دیسک 4ms ← برای پیدا کردن اطلاعات است

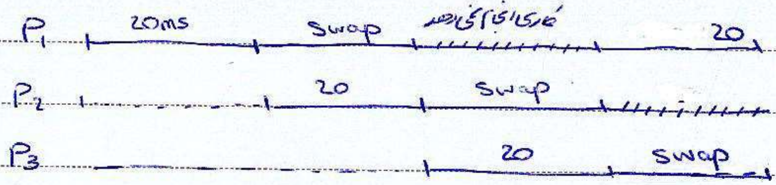
زمان انتقال 6ms

پس هر کدام از اینها برای انتقال به حافظه 10ms طول می‌کشد (یعنی جمع زمان دسترسی + زمان انتقال)

۱۵) برای بهره‌وری ٪100 یک فرآیند چقدر به پردازش ادامه دهد؟

← پردازنده بیکار نباشد همیشه کاری برای انجام داشته باشد

← برای انتقال هر فرآیند باید اول یکی را بیرون کنیم یکی را باید وارد کنیم پس هر انتقال 20ms طول می‌کشد زمان یک swap ، 20ms است و برای اینکه فرآیند بعدی آماده کار شود باید فرآیند در حال اجرا 20ms کار کند که در حین این عمل swap انجام شود که وقتی عمل swap انجام شده فرآیند بعدی 20ms انجام دهد.



پردازنده هیچ وقت بیکار نیست و همیشه چیزی برای اجرا است. swap ها نمی‌توانند هم انجام دهند چون فقط یک دسترسی به دیسک می‌توانیم داشته باشیم

Subject:
Year. Month. Date. ( )

b) اگر هر کاربر یک دستور در هر ثانیه بدهد، حداکثر تعداد کاربران؟

یعنی زمان پاسخ باید 1 ثانیه باشد چون اگر بیشتر باشد متوجه می‌شود

در هر ثانیه به 50 کاربر پاسخ می‌دهیم

$\frac{1s}{20ms} = 50$

c) شکل قبل را برای حالتی بکشید که زمان سرویس 10ms باشد

جلسه چهاردهم 7,30

تمرین) در صورت دو تا فرایند بسازیم که هم زمان اجرا شود چه اتفاقی می‌افتد.

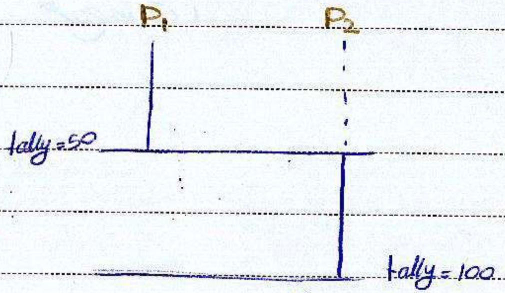
کد زیر را ببین tally را مشخص کن

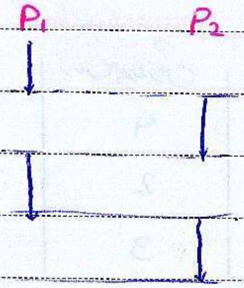
```
Const int n=50;
int tally;
void total()
{   int count;
    for ( count=1; count <=n; count ++)
    { tally ++;
    }
}

void main ()
{ tally=0
                 P1        R2
  Parbegin(total(), total());  ←
  write (tally);
}
```

دو تا فرایند را همزمان اجرا کن

P1 و P2 چون بصورت همروند هستند زمان بین آنها تقسیم می‌شود ما هیچ نمی‌دانیم در هر ترتیبی از این‌ها می‌تواند باشد. ترتیب های مختلفی که می‌تواند وجود داشته باشد کران بالا و پایین هست

کران بالا که حالا مشخص است چون P1 اجرا می‌شود و P2 هم اجرا می‌شود ← 100





در حالت دیگری را ببینیم از این کمتر می‌شود

tally ++ = tally +1 ; زبان اسمبلی

Reg1 = tally;
Rg = Reg1 +1 ;
tally = Reg1;

این را compile کردیم (خودمان compiler شدیم)

برای اجرا باید به زبان سطح پایین تبدیل شود

چیزی که اجرا می‌شود این است

اگر تکه زمان‌ها را در نظر می‌گرفتیم روی چیزهایی که اجرا می‌شود تاثیر دارد.

اگر در حال افزایش tally تصادفاً تکه زمانی در وسط کار یعنی بعد از اجرای Rg = Reg1+1 تمام شد. به خودمان که می‌گوییم چه اشکالی دارد، دفعه بعد که نوبتم شد می‌نویسیم. به خیال باطل چون نوبت را به کس دیگری دادیم مثلاً کس دیگر منتظر بوده که 11 را ببیند ولی جواب 15 را می‌بیند

Reg2 = tally 10

Reg2 = 11

tally = 11

پنج بار تکرار لوپ => tally = 15

tally = 11 ⇐ دوباره نوبت به P1 می‌رسد

به نظرتان 50 جواب باشد برای کران پایین ولی جواب این نیست چون که جواب 2 است، حالا چرا؟

تکه زمانی وقتی بی‌اجازه ما تقسیم می‌شود خیلی مشکلات می‌تواند ایجاد کند و باعث این کارها می‌شود.

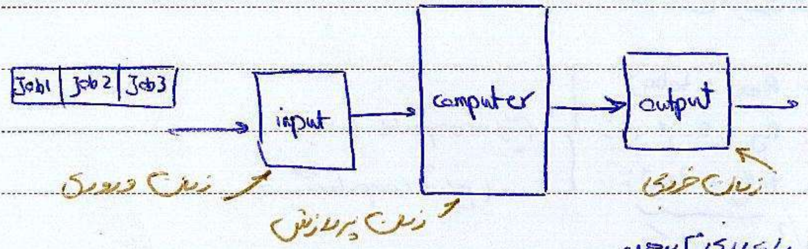
کارهای که در یک دسته به آن می دهند

دسته‌ای

تمرین) راجع به یک مسئله خیلی تیپیکی (در درس گفته نشده)

جدول زیر زمان های لازم برای ورود، پردازش و خروج سه کار را در یک سیستم batch نشان می دهد. حداقل کل زمان مصرفی برای اجرای هر سه کار چقدر است؟ (ترتیب ورود تعیین کننده ترتیب پردازش و خروج است)

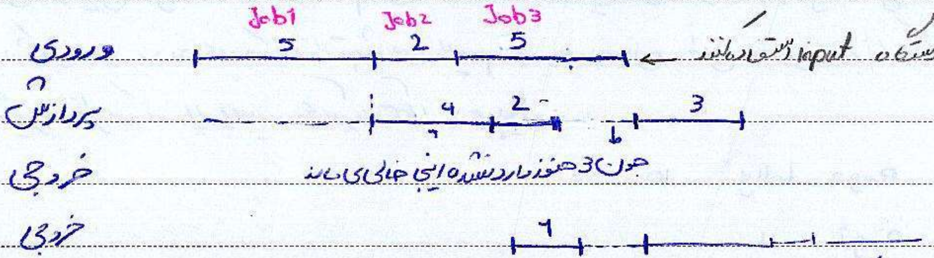
	Input time زمان ورودی	processing time زمان پردازش	output time زمان خروجی
Job1	5	4	1
Job2	2	2	3
Job3	5	3	2


Job1
Job2
Job3
input
computer
output
زمان ورودی
زمان پردازش
زمان خروجی

سیستم batch

این سیستم‌ها تعاملی نیستند
چون ورودی را می‌گیرد که از آن در خروجی را تحویل می دهند

می توانیم به این سیستم یک سری Job برای اجرا بدهیم
هر کدام از این ها زمان های لازم دارند که خوانده شوند ← زمان ورودی


Job1
Job2
Job3
ورودی
پردازش
خروجی
خروجی
5
2
5
4
2
3
1
یکی یکی باید از دستگاه input خوانده شوند
چون 3 هنوز وارد نشده اینجا خالی می ماند

پردازش باید وقتی شروع شود که ورودی به طور کامل خوانده شود یعنی پردازش Job1 بعد از 5 واحد زمانی شروع می شود ولی بعد از آن چون Job2 به طور کامل وارد شده آن را پردازش می کنیم ولی چون بعد از آن Job3 هنوز کامل وارد نشده نمی توانیم آن را شروع کنیم پس سیستم تا حدی بیکاری می ماند و وقتی Job3 کامل آمد شروع به پردازش آن می کند.