سیستم های عامل

"دکتر حقیقت"

تهیه کنندگان:

عبدالکریمی-کرمانشاهی

www.yadmane.com
yadmane@yahoo.com

ویرایش ۱۰۰۰

جلسه اول ۱۳۸۸/۴/۲۲ ساعت ۱۲:۰۰

مرجع اصلی: کتاب تنباوم (ترجمه دکتر حقیقت)

فصل اول

تعریف سیستم عامل

25:00

- سیستم عامل یک نرم افزار است که در دسته نرم افزار های سیستمی قرار می گیرد.
- هرچند نمی توان نرم افزار ها را به دسته های کاملاً مجزای سیستمی و کاربردی تقسیم کرد اما سیستم عامل نرم افزاری است که از سایر نرم افزار ها سیستمی تر است.

تعریف سیستم عامل درکتاب Silberschatz:
سیستم عامل یک نرم افزار سیستمی است که کامپیوتر را راه اندازی کرده و همیشه درحال اجراست.

- **ایرادی که به این تعریف می توان گرفت** : این است که چون بحث ما بر روی تک پردازنده ای است اجرای واقعی همزمان سیستم عامل در کنار برنامه های دیگر ممکن نیست بلکه سیستم عامل در بازه های زمانی مختلف می آید و نقش کنترلی خود را انجام می دهد و در زمانی که CPU در اختیار برنامه دیگر است در آن زمان سیستم عامل یا قبلاً تاثیر خود را گذاشته و یا بعداً می گذارد.

نکات:

ـ اجرای برنامه ها در سیستم به صورت همزمان انجام نمی شود بلکه به صورت همروند Concurrency انجام می شود . این شیوه پارالل نیست بلکه شبه پارالل می باشد. یعنی زمان CPU به بازه های بسیار کوچکی (کوانتوم زمانی) در حد چند میکروثانیه تقسیم و CPU به طور متناوب به پردازه های مختلف سرویس می دهد این کار آنقدر سریع صورت می گیرد که به نظر ما سیستم عامل در آن واحد به همه آنها سرویس می دهد.

- در زمانی که برنامه دیگری در حال اجراست هرچند سیستم عامل حافظه دارد اما بدون CPU هیچ کاری نمی تواند انجام دهد.

ـ حضور سیستم عامل در هنگام اجرای دیگر برنامه ها به معنی اجرا نیست بلکه به معنی آماده واکنش است. یعنی در پشت پرده (background) آماده اجرا می باشند به این نوع برنامه ها demon می گویند.

- معنی لغوی demon : خدایی که دارای قوه خارق العاده است.

- یکevent هایی ممکن است رخ دهد که باعث واکنش سیستم عامل می شود . یعنی CPU می داند که در مواقعی که event هایی مثل وقفه رخ می دهد باید برنامه جاری را رها کرده و به وقفه ها پاسخ دهد.

- پس اگر *Silberschatz* می گوید سیستم عامل برنامه ای است که همیشه در حال اجراست از دید انسان و از دید سطح بالاست نه از دید *CPU*.

ـ Interupt نرم افزاری برای صدا زدن سیستم عامل system call می باشد.

Interupt نرم افزاری به دوصورت رخ می دهد:

1. **خطای نرم افزاری**
2. System Call

51:00

تعریف تننبام از Process :

process is running program

این تعریف نیز از دید سطح بالا درست است چرا که در آن واحد فقط یک پروسس درحال اجراست. و مابقی درحالت ready و suspend و .... هستند.

البته بعضی ها معنی Running و Execute را از یکدیگر تفکیک کرده اند و توجیه این جمله را در تفاوت این دو می دانند.

- در زمان اجرای دیگر برنامه ها سیستم عامل یا وظایف مدیریتی خود را قبلاً انجام داده و یا بعداً انجام خواهد داد مثلا قبلاً برخی رجیستر ها را ست کرده ، زمان بند را برنامه ریزی کرده و... و یا هر زمان وقفه ای رخ دهد حاضر می شود.

55:00

وظایف سیستم عامل :

1. Resource manager (مدیریت منابع)

- استفاده بهینه از CPU
- جلوگیری از dead lock
- رعایت اولویت ها
- جلوگیری از قحطی و گرسنگی
- و....

2. واسط کاربر ومنابع

- هدف سادگی کار با کامپیوتر

توضیح درباره جنبه واسط بودن سیستم عامل:

هر چند کار با ماشین بدون سیستم عامل محال نیست اما بسیار سخت است. کاری که سیستم عامل می کند این است که به عنوان یک واسط قرار گیرد و باعث می شود کسی که از بالا نگاه می کند یک دید انتزاعی (abstract) و highlevel و user friendly نسبت به سیستم پیدا کند.

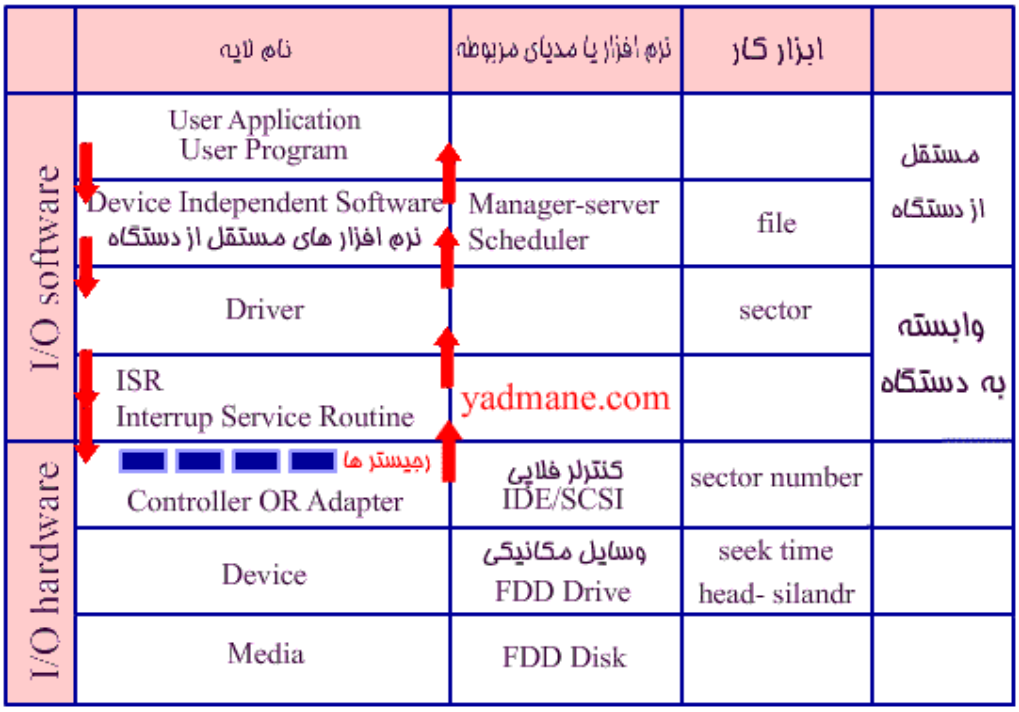
جمله ای از tanenbaum :

Operating system as virtual machine

Operating system as resource manager

بعضی نیز سیستم عامل را **Extended machine** می گویند.

نگاهی اجمالی به عملیات I\O :



با توجه به شکل بالا می توانیم نتایج زیر را استنباط کنیم:

- در لایه های بالا با فایل و رکورد سر و کار داریم اما در لایه های پایین با سکتور و سیلندر و .... .
- هر چه لایه بیشتر وابسته به دستگاه باشد باید با جزئیات سخت افزار بیشتر آشنا باشد .
- لایه ای که در صورت تغییر سخت افزار آن لایه نیز باید تغییر کند Driver می باشد. به همین خاطر همیشه همراه سخت افزار ها Driver آن عرضه می شود.
- برای کار با سخت افزار می توان به روش busy waiting و یا از تکنیک وقفه استفاده کرد.

برای پاسخ به interupt برنامه ای به نام روتین سرویس دهی به وقفه (Interupt Service Routine) یا ISR باید اجرا گردد که برخی کتب آن Intrupt handler یا اداره کننده وقفه نیز می نامند.

نکته: هرگاه نام Driver یا Interupt بیاید وابستگی به دستگاه داریم .

به تعداد Device ها درایور و Interupt Service Routine داریم.

مراحل بازکردن یک فایل ونوشتن در آن:

1ـ user program به file manager با استفاده از دستورات سطح بالا مثل fopen این دستور را می دهد.

2ـ file manager به جداول نگاه می کند و می فهمد باید چه سکتورهایی را بخواند چون خودش نمی تواند با Controller کارکند به همین خاطر این فرمان را به Driver منتقل می کند .

3ـ Driver رجیسترهای کنترلر را program می کند.

4ـ کنترلر روی Driver عمل خود را انجام می دهد وDriver از روی مدیا اطلاعات را می خواند. عمل خواندن که انجام شد Driver به کنترلرخبر می دهند.

5ـ کنترلر یک وقفه صادر میکند . CPU درحال اجرای برنامه دیگری بوده است، برنامه جاری را رها کرده و شروع به اجرای flopy intrupt routine می کند.

6ـ intrupt routine چک می کند. خطا نیست پس درایور را صدا می زند و سکتور خوانده شده را تحویل file manager می دهد. عمل خواندن سکتور بعدی به همین ترتیب انجام می شود تا فایل کامل شود. وقتی فایل کامل شد فایل به برنامه کاربردی داده می شود.

ابتدای صدای 2 جلسه اول

بررسی سیستم عامل به عنوان Resource management :

تعریف منبع : هرآنچه که فرآیندها برای شروع وادامه کار به آن نیاز دارند را منبع می گویند.

فرآیند برای شروع به اجرا و ادامه کار نیاز به منابع دارد این منابع را یک جا درخواست نمی کند بلکه به تدریج و هر زمان که نیاز داشته باشد درخواست می کند مثلا هنگامی که نیاز به پرینتر پیدا می کند درخواست می دهد (request) حال ممکن است پرینتر آزاد باشد که به فرآیند می دهیم (allocation یا تخصیص) و اگر آزاد نباشد درخواست را در صف قرار می دهند. عمل صف بندی را scheduler یا زمانبند انجام می دهد . وقتی نوبت فرآیند برسد آن resource را به آن فرایند allocate می کند و کار فرایند که با منبع تمام شد آن منبع را deallocate یا release می کند.

انواع منبع :

1. **Logical :** رکورد ، فایل .....
2. **Physical :** پرینتر.....

نکته : مدیریت منابع فیزیکی و منطقی برعهده سیستم عامل است.

وظایف سیستم عامل درحوزه مدیریت منابع :

1. استفاده بهینه ازمنابع
2. تخصیص وآزاد سازی منابع :

- ایجاد ،حذف ،وتخصیص واداره فایل و دایرکتوری ها .
- ایجاد ، حذف و اداره فرایندها .
- ایجاد ارتباط بین فرایندها .
- مدیریت حافظه .

3. زمانبندی scheduling
4. حسابداری Acounting

سایر وظایف سیستم عامل :

1. ایجاد امنیت : جلوگیری از ورود همزمان فرایندها به ناحیه بحرانی، جلوگیری از دخالت درحریم دیگران و..... ازدیگر وظایف سیستم عامل است.
2. ایجاد حذف و اداره فایل ها و دایرکتوری ها
3. ایجاد حذف و اداره فرایند ها
4. مدیریت حافظه
5. اتخاذ سیاست مناسب جهت جلوگیری و برخورد با بن بست
6. جلوگیری از گرسنگی یا قحطی
7. جلوگیری از ورود همزمان فرایند ها به ناحیه بحرانی

14:00

تست :

ـ CPU دراختیار فرایند کاربراست اگر فرایند بخواهد تخلفی انجام دهد چه کسی آن را کشف وجلوی آن راخواهد گرفت؟

✓ 1)CPU 2) هسته سیستم عامل 3) process manager 4) Scheduler

پاسخ:

گزینه **1** صحیح است . زیرا درهنگامی که CPU دراختیار برنامه کاربر است درآن زمان سیستم عامل حضور ندارد بلکه سیستم عامل یا قبل از آن تاثیرات مدیریتی خود را گذاشته و یا بعد از آن خواهد گذاشت بلکه این CPU است که تخلف را تشخیص خواهد داد .

ـ برخی ازInstruction های CPU حساس وخطرناک هستند که نباید دراختیار کاربران قرار داشته باشد. بعضی از CPU ها قادر به ساپورت protection هستند یعنی مد حفاظت شده دارند CPU های اینتل از 386 به بعد این مد را (protected mode) پشتیبانی می کند.

انواع مدها:

1. real mode

1.protected mode

درسیستم های حفاظت شده دو مد وجود دارد:

1. مدکاربر User mode

2. مدهسته (مدناظر، مانیتور یاکرنل) kernel mode

privillage Instruction :

دستورالعمل هایی راکه خطرناک هستند و فقط درمدهسته قابل اجرا هستند را دستورالعمل های ممتاز یا privillage Instruction می گویند.

به نظرشما کدامیک ازدستورالعمل های زیر خطرناک هستند؟

ـ ازکار انداختن وقفه ها (Interupt disable) خطرناک

ـ Direct Interupt خطرناک

ـ jmp خطرناک نیست البته باید بازه آدرس چک شود.

ـ نوشتن در program status word چون psw هر برنامه مربوط به خودش می باشد. خطرناک نیست

ـ تنظیم ساعت سیستم (تایمر سیستم) خطرناک

ـ نوشتن در رجیستر های کنترلر: چون مسئول این کار سیستم عامل است و این نوعی دخالت است. خطرناک

ـ دست زدن به رجیستری که level حفاظتی رانشان می دهد. خطرناک

34:00

آدم سالم مردم آزار است.

36:00

مثال معروف وقفه

53:00

تاریخچه سیستم های عامل

نسل اول کامپیوترها :

ورودی یک سری تخته مدار بود که درواقع برنامه نویس ها برنامه خود را سیم بندی می کردند. سیستم عاملی وجود نداشت. برنامه نویس درواقع خودش اسمبلر است یعنی برنامه را با زبان ماشین و با صفر و یک ها باید نوشت. پردازنده نیز با استفاده از لامپ های خلأ ساخته می شد و خروجی نیز یک سری چراغ بود.

ایراد سیستم عامل های نسل اول :

1. منابع را هدر می دادند چون وقتی ورودی کار می کند خروجی بیکار است و...
2. turnaround time آنها بالا بود یعنی زمان پاسخ خیلی طولانی بود.
3. ارتباط باکاربر offline بود.
4. Debug کردن برنامه ها سخت بود

نسل دوم کامپیوترها :

دراین نسل سخت افزارهای جدیدی آمد مثلا ترانزیستور به جای لامپ خلا ، card reader به جای تخته مدار سوراخ دار، پرینتر به جای لامپ هایی که خاموش وروشن می شدند ، tape drive به عنوان ورودی که می توانیم OS آن را لود کنیم. کامپایلر بوجود آمد و زبانهایی که مثل PL1، fortran و cobol .

OS و کامپایلر بر روی tape ها ذخیره می شدند که به آن system tape می گفتند ، برنامه کاربر بر روی کارت پانچ ها و به وسیله Card reader خوانده و در حافظه Load می شد. بعد از پایان هر کار CPU به سراغ کار بعدی می رفت.

نکته: نسل دوم اولین نسلی است که درآن سیستم عامل بوجود آمد سیستم عامل های این نسل را batch می گویند.

batch : به معنی دسته ای می باشد ، یعنی دسته ای از کار ها را پشت سر هم انجام می دادند. سیستم عامل های این دوره به صورت ردیفی کارمی کردند یعنی تک برنامه بودند و تا یک برنامه تمام نمی شد برنامه بعدی شروع نمی شد.

نکته: دقت کنید واژه های Job، Program، Instruction وCommand باهم معادل نیستند.

1. Instruction دستورات زبان ماشین مثل jmp
2. command دستورات زبان سطح بالا که خود شامل چند instruction است.
3. Job عبارت است از :

1. برنامه ها

2. command های سیستم عامل به زبان JCL (فرامین به زبان JCL)

3. داده های ورودی

مجموعه ای ازکارهای بالا در سیستم عامل های دسته ای به صورت پشت سر هم قرار می گرفت و اجرا می شد .

درس تا صفحه 23 کتاب تننباوم ترجمه حقیقت

جلسه دوم ۱۳۸۸/۴/۲۹ ساعت ۱۲:۰۰

ایراد سیستم عامل های نسل اول :

1. منابع را هدر می دادند چون وقتی ورودی کار می کند خروجی بیکار است و...

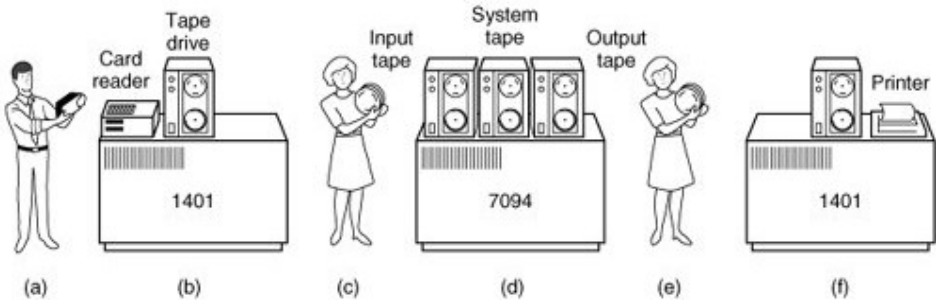
2. turnaround time آنها بالا بود یعنی زمان پاسخ خیلی طولانی بود.

3. ارتباط باکاربر offline بود.

4. Debug کردن برنامه ها سخت بود

ـ حالا می خواهیم ببینیم برای رفع مشکلات (هدررفتن CPU و نارضایتی کاربران) چه راه هایی می توان یافت ؟ ابتدا سعی درحل مشکل هدر رفتن منابع می کنیم.

حل مشکل هدر رفتن منابع در نسل اول :



دوعدد1401 برای افزایش کارایی CPU خریداری شده است، 1401 ها برای کارهای ساده وکند مثل ورودی ـ خروجی استفاده می شود . 7094 ها کار پردازش را انجام می دهند.

1401 ها ، یکی یکی کارت ها را می خواند و روی نوار می ریزد. روی نوار چندین job قرار می گیرد که به مجموع آنها batch گفته می شود.

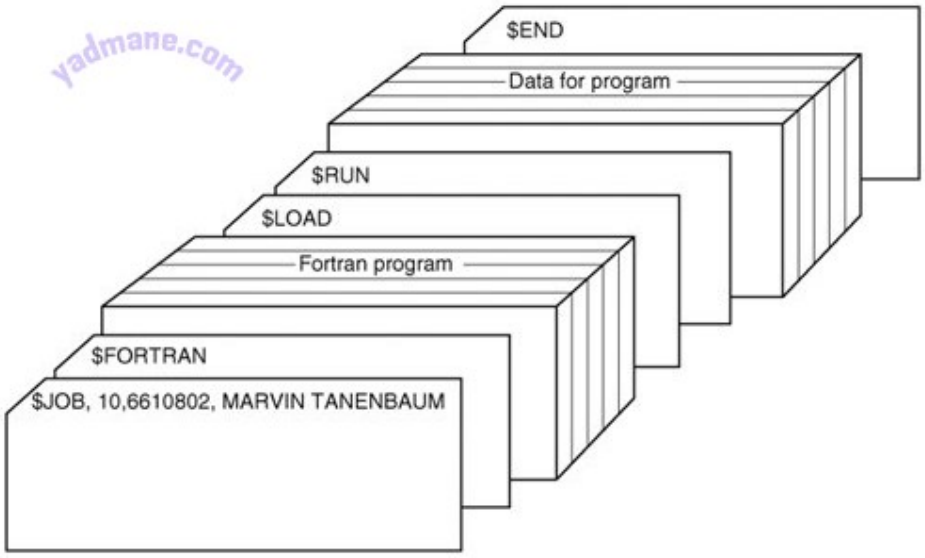
Job : مجموعه ای از برنامه ها ، داده ها وفرامین کنترل روند اجرای کار، به زبان jcl می باشد.

وقتی نوار پر شد آن را با نوار خالی جاگزین کرده و نوار پر را به اتاق پردازش (7094) می برند در این لحظه که اتاق پردازش کار خود را می کند اتاق ورودی نیز بی کار نیست و درحال خواندن کارت های جدید است.

7094 کار(job) ها را از ورودی می خواند.

بر روی نوار سیستم نیز سیستم عامل وکامپایلر هایی مانند فرترن قرار دارد.

JCL ها همه با $ شروع می شوند مثل $LOAD و $END و..... مثلا وقتی در برنامه عبارت $FORTRAN باشد می فهمد که باید کامپایلر فرترن را روی حافظه load کند .



خروجی پس از پردازش دوباره روی نوار فرستاده می شود (نه روی دستگاه خروجی مثل پرینتر) وقتی نوار پر شد نوار پر را به اتاق خروجی 1401 می برند و خروجی ها از نوار خوانده شده و چاپ می گردد.

اگر در این لحظه که اتاق خروجی نیز شروع به کارکرده دقت کنیم تمام دستگاه ها مشغول به کار هستند.

نکته:

آنچه که در اینجا باعث شد بهره روی سیستم بالا برود ، اجرای همزمان همه دستگاه ها بود که به آن spool می گویند.

تعریف Spool :

عملیات همزمان وofline دستگاه های جانبی را Spool می گویند.

Spool درحالت offline راغیر مستقیم گویند چرا که دستگاه های جانبی به صورت مستقیم به سیستم پردازشگر متصل نیستند.

Spool: simultaneous peripheral operations on-line عملیات آنلاین و همزمان دستگاه های جانبی

همزمان : simultaneous

دستگاه های جانبی: peripheral

عملیات : operations

اگر مشخص نکردند چه نوع spoolingی منظور Online Spooling است. البته offline spooling در همه کتاب ها نیامده است.

مزایای Spooling:

1. افزایش بهره وری چرا که همه دستگاه ها با هم کار می کنند.
2. استفاده از راه دور ساده شد یعنی شرکت های کوچک می توانند یک کامپیوتر ارزان بخرند برنامه را بوسیله کارتخوان خوانده و برنامه را به صورت نوار به سرور اصلی انتقال دهند.

معایب Spooling :

1. ارتباط با کاربر همچنان offline است .
2. زمان برگشت کار طولانی است .
3. چک کردن و برطرف کردن ایرادات وقت گیراست.

تعریف Parallel :

اجرای همزمان در چند پردازنده ای را گویند.

نکته:

ایرادی که درspooling نسبت به نسل قبل اضافه می شود این است که نمی توان در کارها وjob های یک batch اولویت ایجاد کرد. .

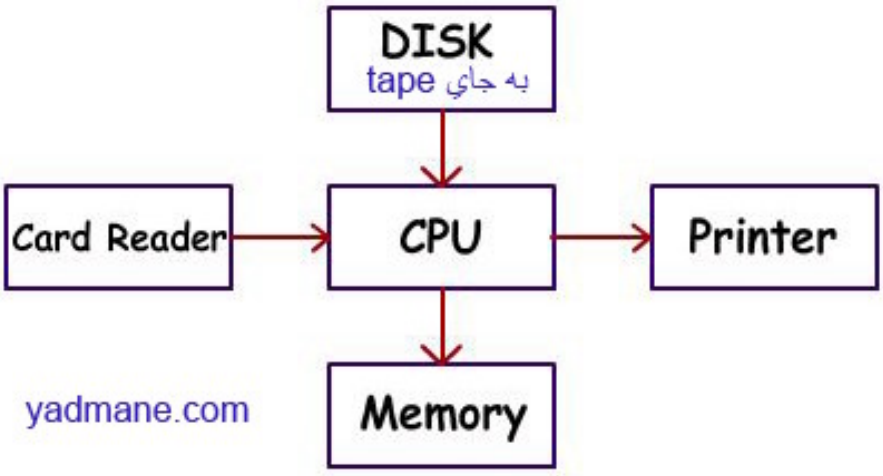
۳۵:۰۰

نسل سوم کامپیوترها :

در این نسل در سخت افزار اتفاقات جالبی افتاده بود:

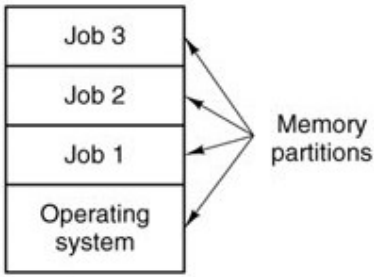
1. ظهور IC ها
2. ظهور هارد دیسک که یکی ازتحولات این نسل است. آنقدر مهم که مایکروسافت نام اولین سیستم عامل خود را DOS به معنای Disk Operating System گذاشت.
3. مکانیزم هایی مانند buffering، interrupt و online spooling دراین نسل آمده است.
4. در اواخر این نسل ظهور ترمینال ها باعث شد ارتباط با کاربر online و interactive شود.

سازمان کامپیوتر های این نسل :



: Batch Multiprogramming

حال ازسیستم های batch ساده به batch چند برنامگی می رویم . این نسل را Multiprogramming یا batch Multiprogramming می نامند. در نسل قبلی یکی یکی برنامه را Load، پردازش و به خروجی می دادیم سپس کار بعدی آغاز می شد . اما در این نسل چند برنامه همزمان درداخل حافظه قرار خواهد گرفت .



سوال: چرا چند برنامه را همزمان درحافظه قرار می دهیم؟ چه تاثیری درسرعت دارد؟

بااین کار در صورتی که برنامه اول به هردلیلی 'خطا یا نیاز به I/O ' ، CPU را رها کرد بلافاصله به برنامه بعدی switch کنیم.

نکته: سرعت سوئیچ دراین جا عامل مهم است وتفاوت در بلافاصله بودن این سوئیچ است .

اگر برنامه اول به هر دلیلی CPU را از دست بدهد یک interrupt رخ می دهد و متوجه می شویم که نوبت برنامه بعدی است . ممکن است interrupt به دلیل error و یا خاتمه برنامه باشد که البته هرکدام ازآنها interrupt مخصوص خود را دارند.

تعریف : Non Preemptive

در روش هایی که تاکنون گفته ایم وقتی که CPU را به کسی می دهیم آن را به زور پس نمی گیریم بلکه هر وقت خودش بخواهد CPU را می دهد به این مدل Non Preemptive (غیر قابل تخلیه پیش هنگام) (غیر قابل پس گرفتن) یا تخصیص انحصاری می گویند.

معنای لغت به لغت : Non غیر قابل ، Pre پیش هنگام ، empti تخلیه کردن ، ve صفت ساز

نکته:

این انحصار با Exclusion که در بحث انحصار متقابل که انحصار مکان است اشتباه نشود.

Multiprogramming سیستمی است که برنامه ها را داخل حافظه می چیند تا هر وقت فرآیندی داوطلبانه CPU را رها کرد بلافاصله به فرآیند بعدی سوئیچ کنیم ، نوع زمانبندی نیز انحصاری است.

57:00

- فرض کنید برنامه درحال اجرا به دلیل اینکه باید منتظر رخداد event ی مثل I/O باشد نمی تواند ادامه یابد چه راه هایی را پیشنهاد می کنید:

1ـ CPU رامشغول نگه داریم ومنتظر می شویم(Busy waiting)

2ـ CPU را داوطلبانه رها می کنیم و منتظر می شویم (asleep wakeup) (Blocked) وبه CPU می گوییم هر وقت رخداد مورد نظر روی داد خبر دهد تا فرایند ادامه یابد.

نکته:

فرایندی که به asleep state رفته (خوابیده است) CPU ندارد وحتی درصف CPU خواهان هم نیست ، درواقع block شده است .

هر زمان رویدادی که برنامه منتظر آن بوده است رخ دهد سیستم عامل این فرایند رابیدار (wake up) می کند و این فرآیند درصف Ready قرار خواهد گرفت اینکه کجای صف قرار خواهد گرفت بستگی به الگوریتم زمان بندی دارد.

* فرایندی که به حالت Block می رود منتظر چه رویدادی است ؟

1ـ I/O

2ـ منتظر نتیجه محاسبه در یک فرآیند دیگر

3- منتظر اجرای فرایند فرزند

4ـ منتظر یک سینگال ازطرف یک فرایند دیگر

پس عملیات I/O صرفا به معنی دستگاه های I/O نیست.

1:08:00

تست:

هدف ازMultiprogramming چیست؟

1ـ داشتن یک response Time مناسب برای کاربران

2ـ رعایت deadline درسیستم های بلادرنگ

✓3ـ استفاده بهینه ازمنابع مخصوصا CPU

تست:

درکدامیک ازسیستم ها CPU بهتر و بهینه تر از بقیه است ؟

1ـ Batch ساده

✓2ـ Multiprogramming : بهترین حالت استفاده از CPU است. با استفاده از online-spooling

3ـ Time sharing : چون switch ها زیاد است سربار زمانی دارد دراینجا هدف response time خوب است نه سرعت .

4ـ Batch multi programming

1:15:00

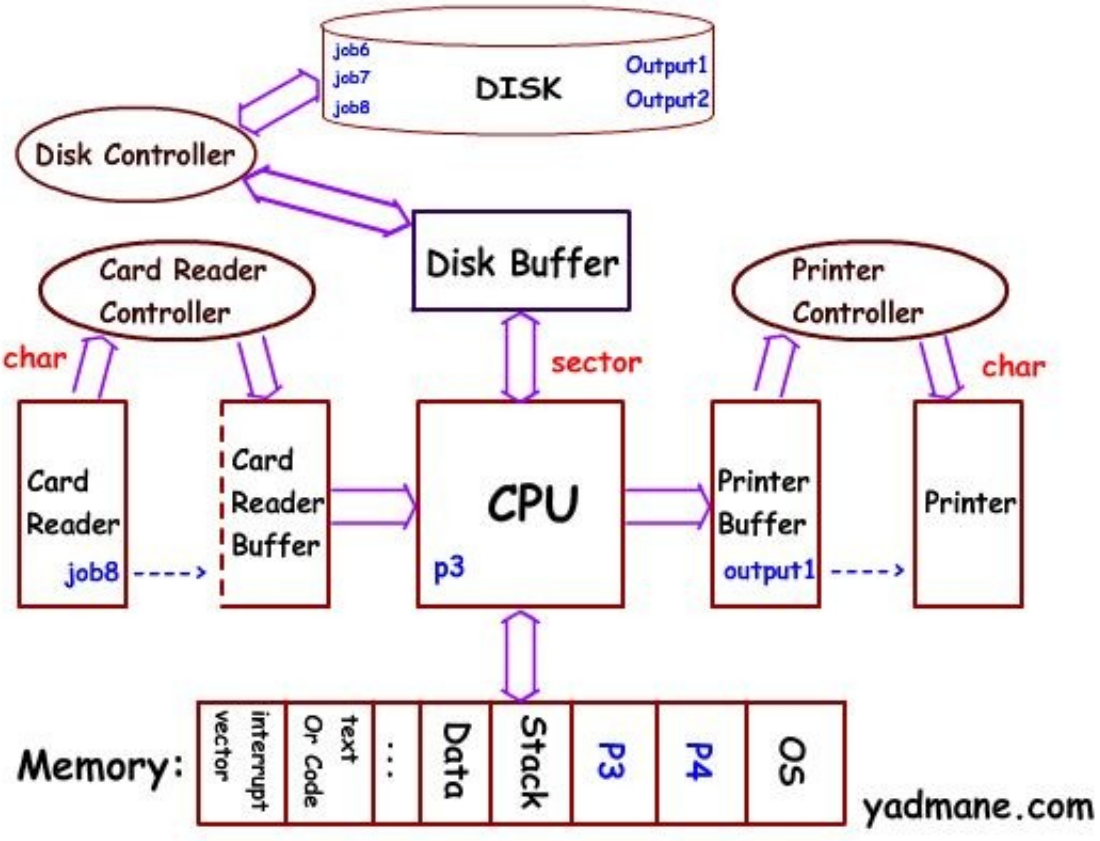
برنامه ها به دودسته تقسیم می شوند : (صفحه 26 کتاب)

1. I/O Bound (I/O Limmited) بیشتر وقت زمان صرف I/O می شود مثل برنامه های تجاری ، بانک ها و..
2. CPU Bound (CPU Limmited) بیشتر زمان صرف محاسبات می شود. برنامه های محاسباتی وشبیه سازی

نکته:

درچند برنامگی ایده آل این است که ترکیبی از برنامه های I/O Limmited و CPU Limmited داشته باشیم اما این کار در اختیار ما نیست و سیستم عامل اطلاعی از داخل برنامه ها ندارد.

ادامه مبحث نسل سوم کامپیوترها :



در شکل بالا نسبت به شکل نسل قبل تفاوت عمده این است که دیسک به جای tape آمده و بین وسایل ورودی خروجی و CPU بافر ها اضافه شده اند. تا وقت CPU کمتر گرفته شود. همچنین نوار های وروی خروجی برداشته شده و ورودی خروجی ، مستقیم و آنلاین شده است.

ازآنجایی که سرعت CR (card reader) نسبت به CPU بسیار کند است برای رعایت اینکه وقت CPU مرتباً صرف کارهای کوچک نشود اطلاعات خوانده شده از CR را در بافر ذخیره و یکجا تحویل CPU می دهیم. در واقع بافر نقش منشی اداره را بازی می کند که نامه ها را جمع کرده وکارتابل را یک جا تحویل رئیس می دهد.

1:35: 00

چگونگی اطلاع یافتن از وضعیت بافر:

1ـ خود بافر اعلام کند که پر یا خالی شده است (Interrupt)

2ـ خودمان مرتبا بافر را چک کنیم (Pooling)

1:38: 00

تفاوت Pooling با busy waiting :

Pooling با busy waiting متفاوت است در pooling بعد از انجام کار یک سری کار، ادوات را چک می کند که کاری دارند یا نه . اما در busy waiting کاری انجام نمی دهد بلکه فقط چک می کند .

مثال در رد pooling : تماس تلفنی به تک تک فامیل برای اینکه ببینیم کسی با ما کاری دارد یا نه .

مثال برای اینکه Interrupt گاهی سخت است: در وسط یک کار ضروری Interrupt می آید حال باید کار جاری را رها کنی.Interrupt آسنکرون و تصادفی است و همیشه باید آمادگی پاسخگویی به آن را داشته باشیم.

1:45:00

حال فرض کنید وقفه آمده چه عکس العملی باید نسبت به آن داشته باشیم ؟

با فرض آنکه کار ما ظرف شستن است و هرInstruction شامل Fetch کردن ظرف جدید وشستن آن باشد فرض کنید ظرف جدید را برداشته ایم که در این لحظه وقفه ای آمده است . مثلاً کسی زنگ منزل را زده چه باید کرد :

1ـ ظرف را رها کرده و به وقفه پاسخ می دهیم .

2ـ ظرف جاری (Instructionجاری) را به پایان رسانده و به وقفه رسیدگی می کنیم .

3ـ کار خود را تا پایان انجام داده (ظرف ها راتا آخر می شوییم در نتیجه کسی که پشت در بود نا امیدانه می رود) بعد به وقفه رسیدگی می کنیم .

✱CPU وقتی که Interrupt آمد Instruction جاری راکه درحال اجراست تمام می کند .

حال با توجه به نوع Interrupt ای که آمده Interrupt Service Routin یا ISR مربوط به آن را اجرا می کند .

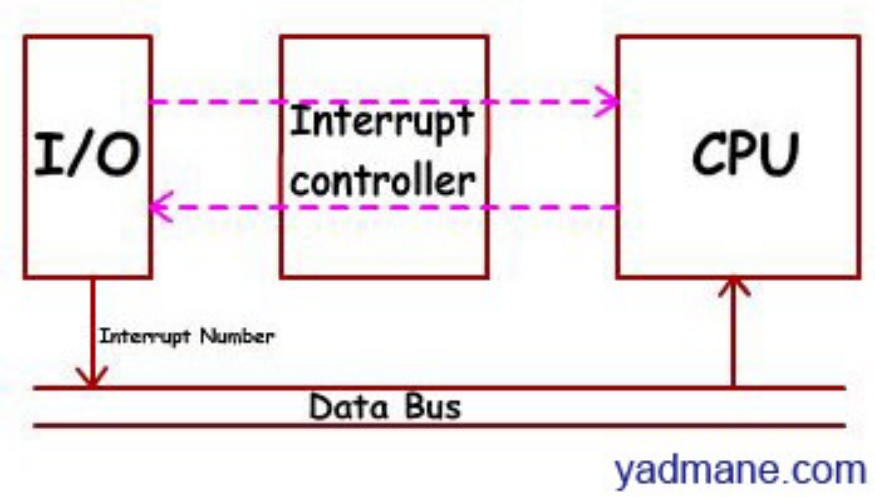
هر Interrupt برای خود یک ISR مخصوص به خود دارد یعنی به تعداد وقفه ها ISR داریم .

مثلاً وقتی رفته گر یا خواستگار زنگ می زند روال برخورد با آنها متفاوت است.

کسی که وقفه را می دهد باید مشخصات خود را بدهد بعد وقتی که وقفه دهنده را شناختیم روال برخورد با آن وقفه را پی می گیریم مثلا از جدول وقفه می بینیم چه تابعی را باید فراخوانی کنیم .

حال اینکه بفهمیم چه کسی Interrupt را داده خود روش های مختلفی دارد مثلا فرض کنید هر زنگ نشان دهنده یک نفر باشد این کار برای تعداد محدود ممکن بود مثلا در قدیم کوبه در مخصوص زنان و مردان متفاوت بود تا وقتی کسی در می زد مشخص باشد که مرد است یا زن. اما اینکه دقیقا چه کسی است نیاز به زنگ های مختلفی داشت که امکان ندارد همین مدل در CPU های قدیمی وجود داشت یعنی هر Interrupt به یک پایه CPU متصل بود اما با افزایش تعداد Interrupt ها این روش عملا ناکارآمد است.

2:03:00



در CPU های جدیدتر هنگامی که وقفه به CPU می رسد، CPU یک ACK می فرستد یعنی اعلام وصول می کند این اعلام وصول دو معنی دارد یکی اعلام آمادگی CPU و یکی هم در واقع پرسیدن نام یا شماره وقفه دهنده، حال دستگاه وقفه دهنده شماره وقفه را که یک عدد 8 بیتی بین 0 تا 255 می باشد را روی Data Bus می گذارد و CPU با برداشتن شماره وقفه متوجه می شود وقفه دهنده کیست.

مشکلی که در اینجا ممکن است پیش آید این است که اگر چند نفر باهم وقفه بدهند CPU چگونه متوجه شود؟

برای رفع این مشکل یک نگهبان (Interrupt controller) می گذاریم که قابلیت برنامه ریزی بوسیله سیستم عامل را دارد زیرا باید بتواند اولویت ها را مشخص کند. و از آنجا که قابل برنامه ریزی است به آن Programmable Interrupt Controller یا PIC می گویند بعضی نیز به آن Priority Base Interrupt Controller می گویند چون اولویت وقفه ها را مشخص می کند.

2:14:30

مراحل وقفه :

1ـ دستگاه I/O سیگنال درخواست وقفه را به PIC می فرستد.

2- PIC چک می کند که CPU ، Interrupt ها را disable نکرده باشد.

3ـ PIC وقفه با اولویت بالاتر را انتخاب می کند (درصورت وجود وقفه های همزمان)

4ـ Cotroller به CPU سیگنال وقفه می فرستد

5ـ CPU ، Instruction جاری را خاتمه می دهد

6ـ CPU به PIC یک Interrupt ACK می فرستد

7ـ PIC شماره وقفه راروی Data Bus می گذارد

8ـ CPU شماره وقفه را روی Data Bus برمی دارد

9ـ CPU رجیسترهای مهم مثل PC، PSW و ... رادرstack فرایند جاری ذخیره می کند.

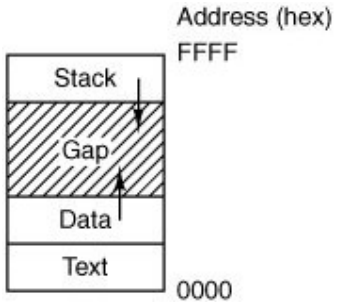
10ـ CPU به خانه شماره N بردار وقفه درابتدای حافظه مراجعه می کند

11ـ CPU آدرس ISR شماره N را بر می دارد و درPC قرارمی دهد

12ـ CPU از مد کاربر به مد هسته سوئیچ می کند واجرای ISR را آغاز می کند

نکته:

هرفرایند stack خاص خود را دارد (حداقل یک stack) درچند نخی بیش از یک stack دارد. سیستم عامل نیز برای هر قسمت مثلا هر سروری که روی آن نصب است یک stack دارد. هر برنامه شامل فضای stack، data، text و gap می باشد که gap فضای خالی برای رشد استک می باشد.



*بعضی ازمراحل بالا ممکن است به صورت موازی انجام شود.

تا اینجا خبری ازOS نبود البته OS قبلا اثرات خود را گذاشته بود مثلا PIC رابرنامه ریزی کرده بود.

2:29:00

فرض کنید که وقفه شماره 7 آمده و CPU با توجه به جدول وقفه می داند که وقفه مربوط به چه دستگاهی است برای اینکه بداند دستگاه مورد نظر چه درخواستی دارد باید رجیستر status در PIC را بخواند و نوع عمل را می فهمد.

CPU در این مدل بسیار خوشبخت است چرا که از هر طرف با بافر ها کار می کند که بسیار سریع هستند و اصلاً منتظر I/O ها نمی شود و به جای اینکه کاراکتر ها را جابجا کند سکتور سکتور اطلاعات جمع شده روی بافر ها را می خواند و می نویسد.

مراحل بالا Online spooling بود.

چهار مکانیزم مهم درonline spooling که اگر نبود سرعت به این حد بالا نبود:

1ـ وجود کنترلرها که موازی با CPU عملیات I/O راکنترل می کنند

2ـ Buffering

3ـ مکانیزم Interrupt (شناسایی ورسیدگی به Interrupt ها)

4ـ وجود Disk یاهمان Random Access Disk عامل مهمی درspooling است و بدون آن درنسل قبلی با tape ها Spooling امکان پذیر نبود. چرا که tape ترتیبی است .

Spooling به ما اجازه Multiprogramming را داد که بدون آن این امکان وجود نداشت.

ابتدای صدای 2 جلسه 2

پیدایش وساخت Terminal ها:

اواخر نسل سوم ترمینال ها ظهور کردند، ترمینال ها مشکل offline بودن ارتباط کاربران را برطرف می کردند اما بدون ایجاد تغییراتی در روش کار این مدل مشکلاتی داشت که بررسی می کنیم:

انحصاری بودن با سیستم های جدید که دارای ترمینال است تناقض دارد. دراین حالت کاربر online است و انحصاری بودن باعث می شود زمان پاسخ (Response Time) کاربران بیش از حد مجاز بوده و برای کاربران نامناسب باشد. به دلیل اینکه در سیستم قبلی کاربران آفلاین بودند انحصاری بودن و زیاد بودن Response Time مشکل ساز نبود.

Response Time چیست ؟

زمان پاسخ از لحظه ورود یک دستور تا زمانی که اولین پاسخ دریافت شود.

(درسیستم های online معنی دارد) نوع سیستم ما Inter Active MultiProgramming است.

هدف ما این است که گارانتی کنیم که زمان پاسخ کاربران مطلوب باشد حتی اگر بازده CPU به دلیل سوئیچ ها تا حدی پایین بیاید. باید از زمانبندی غیرانحصاری (Preemptive) استفاده کنیم وچون سر زمان های خاصی CPU را ازکاربران می گیریم به این روش Time sharing می گویند.

Time sharing :

دراین روش زمان به بازه های کوچکی به نام کوانتوم یا time slice تقسیم می شود. در پایان کوانتوم زمان سنج یک Interrupt سخت افزاری می فرستد. وقتی برنامه ای وقتش به پایان رسید scheduler است که می آید و برنامه بعدی را مشخص می کند.

وقتی کوانتوم در اختیار یک کاربراست سه اتفاق ممکن است بیافتد:

1ـ وسط کوانتوم کارش تمام شود ← بلافاصله سوئیچ می کنیم.

2ـ وسط کوانتوم به یک I/O نیازپیدا می کند ← بازCPU را داوطلبانه رها می کند.

3ـ کوانتوم تمام می شود اما هنوز به CPU نیاز دارد ← CPU را به زور از Process می گیریم که در اینجا یک overhead برای context switch داریم.

13:30

سوال: برنامه ای درحال اجراست درطی کوانتوم نه تمام می شود ونه بلوکه، اولین نفری که تشخیص می دهد که کوانتوم تمام شده است وباید CPU گرفته شود کیست؟

پاسخ: Timer است .

Timer چیست ؟

CPU ساعت داخلی ندارد بلکه یک ساعت برای کل سیستم داریم (IC تایمر) که سر موقع در پایان کوانتوم یک Interrupt به PIC می فرستد.PIC به CPU اطلاع می دهد، حالا scheduler وارد عمل می شود و نفر بعدی را انتخاب می کند بعد dispatcher را صدا می زنیم تا بساط نفر بعدی را پهن کند.

سوال: تایمر از کجا می فهمد که چه زمانی باید interrupt دهد؟

تایمر و تمام سخت افزارها توسط سیستم عامل برنامه ریزی شده اند. تایمر را نیز OS برنامه ریزی کرده است. زمان هر کوانتوم را هم OS معین می کند. در واقع سیستم عامل توسط مدیریت سخت افزار ها حضور خود را حفظ می کند.

تست: N فرایند در سیستم اشتراک زمانی داریم. q هم اندازه کوانتوم است. یک فرایند محاسباتی داریم. این فرایند حداکثر چقدر طول می کشد تا اولین دستورش را اجرا کند؟

1) nq 2) (n-1)q 3) بلافاصله (0) 4✓) هیچ وقت

گزینه 4 . چرا که دراینجا درست که Time sharing داریم اما مشخص نکرده ایم از کدام زمانبندی استفاده می کنیم

ـ Time sharing یعنی ما سر کوانتوم CPU را از شما می گیریم اما از چه زمانبندی استفاده می کنیم می تواند متفاوت باشد . ممکن است از زمانبندی استفاده کنیم که اولویت ها در این نوع زمانبندی به گونه ای باشد که برای فرایند ما قحطی پیش آید.

36:30

نسل چهارم کامپیوترها :

دراین نسل مدارهای مجتمع بسیار فشرده VLSI ظهور کردند که به آنها میکروپرسسور یا ریزپردازنده می گویند. این پدیده باعث کاهش شدید حجم سیستم ها وکاهش قیمت آنها شد و با ظهور آنها کامپیوتر های شخصی به بازار آمد و ادارات نیز علاقمند به خرید PC به جای ترمینال ها شدند.

از طرفی استفاده از PC به جای ترمینال باعث شد شبکه های کامپیوتری بوجود آیند.

دو نوع سیستم عامل وجود داشت :

1ـ سیستم عامل های work station های شبکه win95, win98

2ـ سیستم عامل های مخصوص سرور network operating system که سرویس های ویژه شبکه ای می دهند مثل فایل سرور

3 ـ سیستم عامل های توزیع شده Distributed Operating System یا DOS .

جلسه سوم ۱۳۸۸/۵/۱۵ ساعت ۱۲:۰۰

سیستم عامل های توزیع شده Distributed Operating System یا DOS :

مشکل اصلی: کار با سیستم های شبکه ای پیچیدگی است. باید نگران امنیت ، تداخل (حضور همروند دیگران) ، به خاطر سپردن آدرس های منابع ، جابجایی منابع و..... باشیم . تمامی این موارد باعث سخت بودن کار با سیستم های شبکه ای می شود.

چیزی که می خواهیم این است که با وجود اینکه سیستم ها متمرکز (مانند سیستم های قدیمی) نیستند اما برای کاربر متمرکز به نظر برسد یعنی می خواهیم سیستم ها call by name باشد نه call by address .

- DOS / call by name (Distributed Operating Systems)
- NOS / call by Address (Network Operating Systems)

تفاوت های سیستم های Dos و Nos :

1. در DOS همه چیز متمرکز به نظر می رسد. درصورتی که در NOS همه چیز پراکنده بود.
2. در DOS راحتی کاربر مهم است و باید به نظر کاربر یک سیستم واحد به نظر برسد که گاهی به آن single systems image می گویند چرا که به صورت مجازی تک پردازنده ای به نظر می رسد.

15:00

شفافیت Transparency :

درواقع می خواهیم کاربر پیچیدگی های پشت صحنه را نبیند به همین خاطر بعضی Transparency را پنهان سازی می گویند.

به دلیل پیچیدگی های DOS ، دانشمندان به این نتیجه رسیدند که رسیدن به این هدف که Kernel ی را طراحی کنند که بر روی همه ماشین ها اعم از کلاینت و سرور با تکنولوژی های مختلف بتواند کار کند بسیار مشکل است و به همین خاطر تصمیم گرفتند به جای DOS بر روی سیستم های توزیع شده (DS) یا Distributed System تمرکز کنند.

دراین روش به جای اینکه کرنل (kernel) ها را واحد کنند یک لایه از نرم افزار بر روی کرنل و زیر لایه application به نام میان افزار (Middle ware) قرار گرفت. وظیفه این لایه این است که کاری کند که همه چیز متمرکز به نظرآید (Transparency) پس به جای آنکه کرنل ها را واحد کنند یک لایه روی کرنل برای این منظور درنظر گرفته شد. برای این نوع بهترین مثال web می باشد که ما بدون آنکه بدانیم از چه سروری استفاده می کنیم و بدون آنکه نیاز باشد ازOS خاصی استفاده کنیم می توانیم ازصفحات وب استفاده کنیم.

آشنایی با چند نوع سیستم عامل خاص:

MultiProcessing System : سیستم هایی که درآن بیش از یک پردازنده وجود دارد. More than one CPU بهتر این است که به آن **MultiProcessor** گوییم.

Embedded Systems **یا** Embedded operating systems :

سیستم های تعبیه شده (special purpose)

توکار- تعبیه شده = Embedded

یک دستگاه خاصی را ساخته اند که یک OS مخصوص دارند. معمولا این O.S بسیارکوچک است زیرا وظایفشان محدود است.

29:50

سیستم عامل های بلادرنگ - Real Time O.S :

نکته: آنچه در این سیستم ها مهم است مهلت زمانی است (Dead line) یعنی پاسخ به موقع باشد نه فوری.

انجام چند کار همزمان به صورت بی درنگ امکان ندارد بلکه باید طوری برنامه ریزی کرد که همه کار ها در مهلت معین انجام شوند یعنی کاری که مهلت کمتری دارد ابتدا انجام شود. قطعا نمی توان چند کار را که همزمان می آیند همه را بی درنگ انجام دهیم چون برای کار دوم به اندازه کار اول باید درنگ کرد همین طور تا آخر پس انجام بی درنگ و فوری کار ها آنچنان معنا ندارد بلکه سیستم باید به گونه ای کار ها را انجام دهد که کار ها در موعد مقرر انجام شوند.

فرض کنید 1 ثانیه زمان داریم ویک کاری داریم که 0/3 ثانیه service Time نیاز دارد.

Dead line = 1sec

service Time = 0/3 sec

پس 0/7 ثانیه وقت داریم تا سستی کنیم (LAXITY)

Laxity = Deadline – Service Time

پس باید طوری کارها را بچینیم که همه کارها قبل از Dead line شان انجام شود.

برای انتخاب کار ها لزوما یک انتخاب نداریم و الگوریتم های مختلفی وجود دارد.

بحث سخت و نرم هم دارم (hard real time , soft real time) یعنی کار هایی که ضرورت انجامشان خیلی زیاد است مثلا کاری که در صورتی که انجام نشود منجر به انفجار می شود.

41:00

وقفه های سیستمی :

System call : نوعی وقفه است (نرم افزاری)

1ـ درخواستی که فرایند سطح کاربر (مدکاربر) ازهسته O.S می خواهد.

2ـ از مد کاربر باید به مد هسته سوئیچ کنیم . این کار به دلیل سادگی و امنیت است چرا که برای سادگی کار خودمان راحت تر است آن را از سیستم عامل بخواهیم برایمان انجام دهد و خودمان درگیر آن نشویم. همچنین از لحاظ امنیتی اجازه آن کار را نداریم چون نیاز به دستورات privilege داریم.

*برای سوئیچ کردن از Trap Instruction استفاده می شود.

به دلیل تغییر مد و کار های دیگری که لازم است ، فراخوان های سیستمی و همه تله ها کند هستند.

System call را فراخوان سیستمی، فراخوان هسته ای، SVC(supervisor call)، kernell call یا Monitor call نیز می گویند. در ویندوز API یا Application Program Interface می گویند. البته بعضی API را یک Library می دانند که در درون خودش system call دارد می دانند.

50:00

– صفحه 133 کتاب

انواع وقفه ها

- وقفه های سخت افزاری (خارجی)
- وقفه های نرم افزاری (داخلی)

تفاوت اصلی وقفه های سخت افزاری ونرم افزاری:

وقفه های سخت افزاری از دید برنامه ناهنگام و تصادفی هستند اما وقفه های نرم افزاری همگام است.

یعنی به عنوان مثال در خط 100برنامه ، دستور ورودی داریم هر سری که نرم افزار اجرا می شود همان وقفه در همان جا رخ می دهد مثلا درخط 2000ام یک برنامه یک وقفه است (درخواست ورودی ـ خروجی یا تقسیم برصفر(استثنا یا exeption)) که هرسری برنامه را اجرا کنیم رخ می دهد.

دقت کنید وقتی نرم افزار درخواست I/O می دهد یا از file manager می خواهد یک فایل را برای او بخواند به این حالت که نوعی system call است یک وقفه نرم افزاری می گویند .

اما وقتی دستگاه I/O عمل مورد نظر را انجام می دهد یک وقفه I/O می فرستد که سخت افزاری است.

انواع وقفه های سخت افزاری :

1ـ وقفه I/O

2ـ خطا یا نقص سخت افزاری مثال : Ram parity error

3ـ وقفه ساعت (کوانتوم)

4ـ restart

1:00: 30

دستور hult یک حالت است که CPU رابی کار می کند ولی همه CPU ها آن را ندارد.

انواع وقفه های نرم افزاری :

1ـ **API ها یا System call ها .**

- آنهای که با فرایندها سروکار دارند.
- آنهایی که با سیستم عامل سروکاردارند.

2ـ **exception ها**

3ـ سیگنال هایی که نرم افزارها به هم می فرستند یا ممکن است user به نرم افزار بفرستد.

✱ system call :

واسط بین برنامه های کاربردی و سیستم عامل است . بوسیله یک مجموعه از دستورالعمل های توسعه یافته که Dos آنها را دراختیار قرار می دهد، تعریف شده است این دستورالعمل ها را system call می گویند.

تفاوت Systemcall با exeption :

فرض کنید کاری را خودمان مجاز نیستیم انجام دهیم که باید انجام شود دو راه برای انجام آن وجود دارد.

1 - OS را دور بزنیم و دور ازچشم OS این کار را انجام دهیم اما این کار از نظر CPU مجاز نیست وجلوی آن را می گیرد که به آن **Trap یا exeption** می گویند. دستوراتی که مجاز به انجام آنها نیستیم خود به دسته هایی تقسیم می شوند مثلا می خواهیم از مد کاربر یک دستورالعمل ممتاز را صدا بزنیم یا می خواهیم به فضای آدرس دیگران دسترسی داشته باشیم و یا به رجیستر های I/O دسترسی داشته باشیم و خودمان مستقیما از I/O بخوانیم که در این موارد به دیگر فرایند ها نیز صدمه می زنیم اما فرض کنید می خواهیم تقسیم بر صفر کنیم که در این صورت غیر از خودمان به دیگر فرایند ها صدمه نمی زنیم حتی ممکن است سهواً باشد مثل overflow یا تقسیم بر صفر ، اما از نظر CPU این عمل برنامه کاربر exception است یعنی استثنایی غیر قابل انجام است.

دکتر فهیمی به آن program check می گوید. استالینگ trap و تننباوم exception می گوید.

از نظر استالینگ خطا های نرم افزاری Trap می باشد اما از نظر تننباوم کلیه وقفه های نرم افزاری Trap است. چرا که در سخت افزار یک instruction داریم به نام trap و وقتی آن راصدا می زنیم وارد سیستم عامل می شود و این system call است.

2- از خود OS بخواهیم این کار را برایمان انجام دهد. **Systemcall**

3- سیگنال ها را نیز تننباوم نوع سومی از وقفه ها می داند. و می گوید سیگنال ها در نرم افزار شبیه وقفه ها در سخت افزار هستند. 1:14:00

فرآیندها ص 38 کتاب

نکته:

job همان برنامه نیست و فرایند هم همان job نیست.

job شامل برنامه، داده های ورودی و jcl است.

روند تبدیل job به فرایند یا process creation :

Job scheduler از صف job ها یک job را انتخاب می کند job از خودش تنها text (کد های برنامه) و داده های ورودی داشت. اما برای تبدیل شدن به فرایند سیستم عامل ملزومات دیگری را به آن می دهد از جمله پشته ، فضای آدرس فرایند در حافظه ، رجیستر های IP و psw و sp و ... در CPU و pcb , ... می دهد و در واقع به آن روح می دهد.

فرایند :

1ـ فضای آدرس = Core Image ، برنامه اجرایی - داده های برنامه - پشته

2ـ رجیسترها = PC,SP,Reghard ware

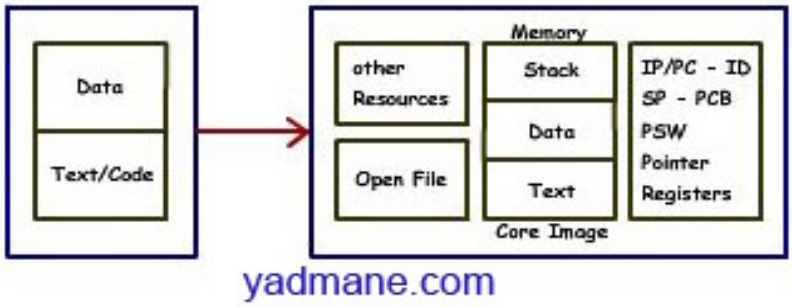
3ـ PCB = شناسه فرایند، شناسه کاربر،شناسه گروه

برنامه = کار = فرایند نیست!

وقتی یک کارتبدیل به فرایند می شود یک موجود جدید ایجاد می شود به این کار: process creation می گویند.

وقتی یک فرایند ایجاد می کنیم یک فریم یا یک قالب دارد که عناصر فرایند درآن جایش مشخص است مثل ظرف های سلف سرویس ها که جای برنج ، خورش و ... مشخص است که به آن Process context می گویند.

Context را ترجمه های مختلفی کرده اند مثل متن ، زمینه و مفاد که مفاد به نظر زیباتر است.



1:30:00

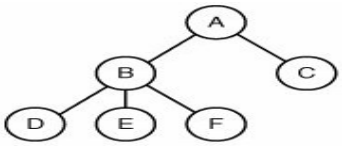
فرآیند از سه بخش تشکیل می شود :

1ـ کد (متن)

2ـ execution part (قسمت اجرایی) (رجیستر، صندوق پستی، stack و PCB)

3ـ منابع Resource

فرآیند ها به صورت درخت هستند یعنی یک ریشه دارند و مابقی فرآیند ها فرزندان و نوادگان ریشه هستند در unix فرآیند ریشه init نام دارد. تمام فرآیند ها به جز ریشه به درخواست یک پدری بوجود آمده اند مانند تمام انسان ها که به جز حضرت آدم پدر داشته اند.



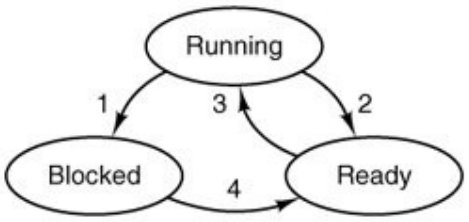
درصفحه 90 اطلاعات یک PCB آمده است.

به کل اطلاعات فرایند جدول فرایند(Process table) هرسطر آن یک PCB می گویند.

قدیم به جای رم چنبره های مغناطیسی به نام core memory بوده به همین خاطر اصطلاحات زیر معادل هستند:

تصویر حافظه اصلی= Main Memory Image = Ram Image = Core Image

فرایند وضعیت های مختلفی دارد (block، Ready و....) مثل یک موجود زنده از خودش فرزند ایجاد می کند، اولویتش کم و زیاد می شود، با هم پیام ردو بدل می کنند و... اما Job اینگونه نیست.



1. Process blocks for input
2. Scheduler picks another process
3. Scheduler picks this process
4. Input becomes available

1:46:00

توضیحاتی در مورد یونیکس:

فایل ها :

در دیدگاه unix همه چیز را فایل می بیند. دایرکتوری، پرینتر و.... همه منابع را فایل می بیند.

لوله (pipe) : صفحه 44

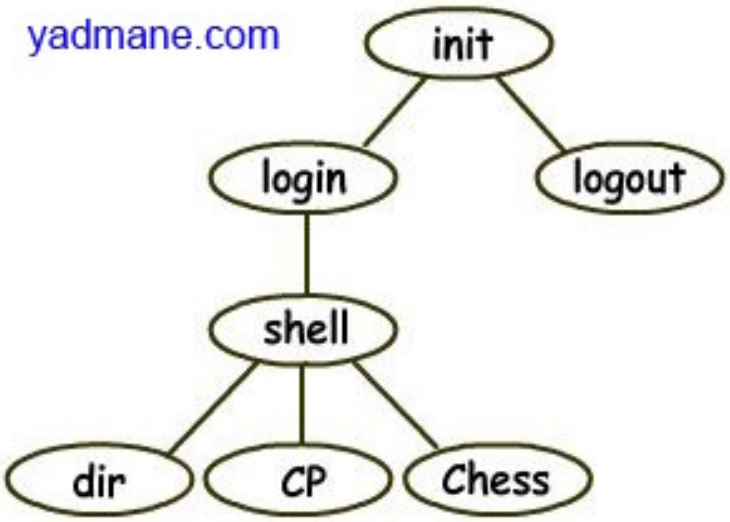
pipe یک file مخصوص است که خروجی یک دستور به صورت یک شبه فایل به عنوان ورودی دستور دیگر استفاده می شود.

Cat A > grep FF

علامت > برای redirect کردن خروجی استفاده می شود.

ابتدای صدای 2 جلسه 3

فراخوان سیستمی برای مدیریت فرایند:



Shell چگونه فرامین را از کاربر می گیرد و چگونه به ازای هر فرمان یک فرزند ایجاد می کند تا آن فرزند برود و آن فرمان را انجام دهد.

shell خودش هیچ کاری نمی کند بلکه یک command interpreter است و خودش انجام دهنده فرایند نیست بلکه یک مفسر است و مثل یک واسط انجام وظیفه می کند.

```
#Define TRUE 1
while (TRUE){     //repeat forever
        read_command(command,parameres);// یک فرمان را با پارامتر های آن می گیرد
        if(fork()!=0){ // خودش فرمان را انجام نمی دهد . بلکه fork می کند یعنی یک بچه ایجاد می کند
                از این لحظه کد برای هم پدر و هم فرزند است چرا که همه چیزش کپی پدر است //

                و فرزند هم از همین خط برنامه را ادامه می دهد منتها با آی دی های متفاوت//
                waitpid(-1,&status,0); // پدر وارد این قسمت می شود
        } else {
                execve(command,parameres,0); // فرزند وارد این قسمت می شود
                        اما قرار نیست فرزند مانند پدر شل بماند بلکه باید به برنامه مورد نظر کاربر تبدیل شود //
        }
}
```

fork یک فرزند همانند پدرش ایجاد می کند برای این کار به هسته می گوید که یک فرزند مثل خودش درست کند. اما فرض کنید قرار بوده برنامه شطرنج اجرا شود اما یک فرزند همانند برنامه پدر ابتدا ایجاد می شود مثل این است که از pcb برنامه پدر کپی گرفته باشید و منابع و همه چیز آن به غیر از id آن مانند پدر است. Fork، id فرزند را بر می گرداند پس id پدر یک id معتبر می شود اما id فرزند صفر است چون فرزندی ندارد. id فرزندان صفر است پس در کد بالا فرزندان به قسمت else می روند وپدر به قسمت if . حال اگر قرار بوده شطرنج اجرا شود درابتدا وقتی فرزند متولد می شود مانند پدر یک shell است اما می گوید من رابا chess عوض کن.

دراین موقعیت پدر 2 کارمختلف را می تواند انجام دهد :

1ـ wait for a child process یعنی منتظر می ماند تا فرزندش terminate شود بعد بیدار شود. دراین صورت فرزند که دیگرعوض شده وقتی کارش تمام شد به OS می گوید بیا ومن را exit کن.

2ـ shell می تواند نخوابد ویک سری کارهای دیگرانجام دهد. (multi tasking)

نکته: به ازای هرکاربریک shell جدید ایجاد می شود.

36 :00

(صفحه ۶۳ کتاب)

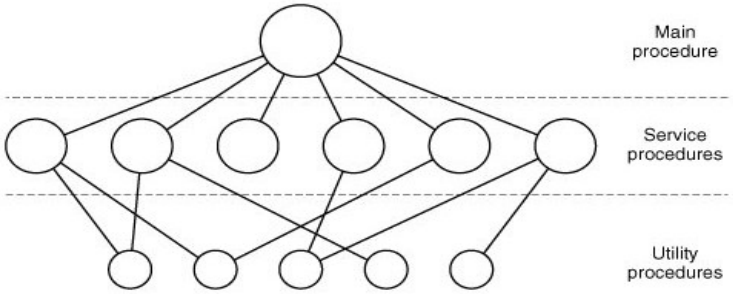
ساختارسیستم عامل :

اولین ساختار، سیستم عامل ها Monolithic بود. Monolithic درواقع ساختاری است که ساختار ندارد.

دومین ساختار، ساختار لایه ای است که توسط Dijkstra ارائه شده است.

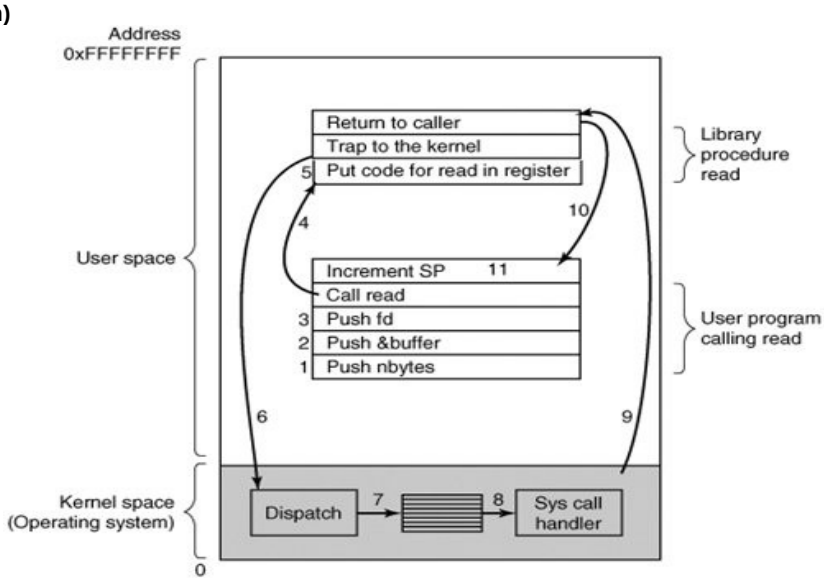
1. ساختار Monolithic :

simple structuring model for a monolithic system.



ساختار Monolithic ازنظر Modularity ـ Flexibility، Reliability وmainbility و , recovery maintenance صفرمی گیرد. ممکن است ساختارMonolithic درحالت خاص درست باشد اما درحالت کلی درست نیست. اما ساختار Monolithic ازنظرPerformance (سرعت ـ کارایی) در درجه خوبی است. (برخلاف معیارهای قبلی) چرا که تله ها کند هستند و در ساختار منولیتیک ما مینیمم یک تله برای فراخوانی سیستمی نیاز داریم . اما در ساختار های دیگر بیش از یک تله نیاز است و به همین دلیل ساختار منولیتیک سرعت بیشتری دارد.

The 11 steps in making the system call `read(fd, buffer, nbytes)`. (This item is displayed on page 43 in the print version)



2. ساختار های لایه ای

در ساختار لایه ای وظایف لایه ها ازهم جداست، لایه های مجاور باهم درارتباط هستند. مثلا لایه 5 با لایه 6 و7 درارتباط است . درسیستم های حلقوی لایه هایی که به هسته نزدیکتر است ازنظر اختیارات وامنیت در درجه بالاتری است.

تشابه سیستم عامل ها با میوه ها:

- سیستم عامل های لایه ای مثل پیاز هستند ضمن این که پیاز encapsulation هم دارد و لایه ها با یک پرده از هم جدا شده اند.
- بعضی مثل انگور هستند (کلاسترینگ) وقتی ما سرور های مختلف را از هم جدا می کنیم مثل فایل سرور وب سرور و ...
- بعضی مثل سیب زمینی هستند مثل منولیتیک

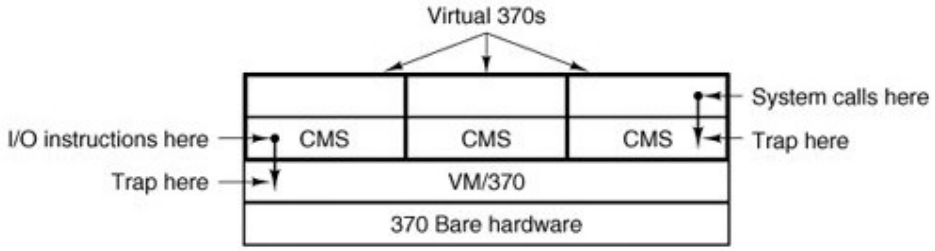
جلسه چهارم ۱۳۸۸/۵/۱۲ ساعت ۱۲:۰۰

3. ماشین مجازی:

آیا می توان روی یک ماشین بیش از یک سیستم عامل به صورت همروند کارکند؟

03:00

این کار به وسیله تکنیک ماشین مجازی (VM) که در 1970 توسط IBM ابداع شد قابل انجام است.



البته سیستم عامل های موجود در شکل می تواند متفاوت باشد.

در این روش یک لایه نرم افزاربه نام VM یا VMM (یا Virtual Machine Monitor) روی سخت افزارVM/370 اجرا می شده و وظیفه آن تقسیم منابع وتشکیل n تا ماشین مجازی کوچکتر.

- ماشین مجازی ازتقسیم منابع ماشین حقیقی بوجود می آید. مثلا تقسیم زمان، تقسیم فضا یا دیسک . همه منابع را می توان مجازی کرد و ترکیب آنها ماشین مجازی می شود .
- ازنظر معماری ماشین مجازی عیناً با ماشین حقیقی منطبق است. به همین دلیل است که هر نرم افزاری که روی ماشین حقیقی اجرا می شود می تواند روی ماشین مجازی نیز اجرا شود چون عینا همان سخت افزار را می بیند. سیستم عاملی هم که قراراست روی VM بنشیند لازم نیست بداند زیرش VM است اگر چنین بود برای VM باید سیستم عامل مخصوص VM تهیه می کردیم.
- VM نوعی OS نیست چون وظایف OS ساده کردن و resource Manager بود. VM هیچ کاری را ساده تر نمی کند وخاصیت انتزاعی برای سیستم ایجاد نمی کند. در زمینه مدیریت منابع هم فقط تقسیم منابع می کند.
- VM میان افزار هم نیست . میان افزار بالای سیستم عامل قرار می گیرد اما VM زیر سیستم عامل قرار می گیرد.

14:30

یک نوع VM دیگر هم به ترتیب زیر می باشد که یکی از روش های Dos در ویندوز به این روش بوده است که قسمتی ازCPU به صورت واقعی به عنوان یک پردازنده کوچکتر همانند سازی کرده اند. مثلا فرض کنید قسمتی از یک پردازنده پنتیوم 8086 را شبیه سازی کرده اند. در این حالت فرض کنید ویندوز روی پردازنده پنتیوم اجرا می شود داخل آن Dos می تواند روی پردازنده کوچکتر اجرا شود. این مدل VM با VM قبلی فرق می کند در آنجا یک CPU را تقسیم زمانی می کردیم در حالی که در اینجا دو پردازنده واقعاً در کنار همدیگر کار می کنند. البته نه به صورت صد در صد موازی چون I/O منابع مشترکی دارند که باید با یکدیگر هماهنگ باشند.

نوع دیگری از VM ها java virtual machine در واقع virtual machine monitor نیست بلکه process virtual machine است. مثلا شما می توانید یک process را از روی یک ماشین بردارید و روی یک ماشین متفاوت اجرا کنید. در واقع VM جاوا را روی ماشین های فیزیکی متفاوت تقلید می کند . با این VM شما می توانید یک برنامه را روی چند ماشین مختلف اجرا کنید بدین ترتیب که برنامه شما به زبان یا کد های جاوا ترجمه می شود بعدا این کد های جاوا روی هر ماشینی که بروند توسط مفسر JVM یکی یکی و run time به instruction های آن ماشین حقیقی تفسیر می شوند این کار یک کم سرعت را پایین می آورد ولی در مقابل قابلیت حمل را بالا می برد.

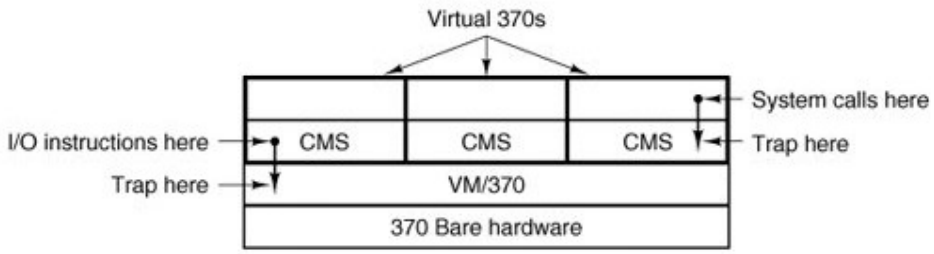
4.Exokrenel ها

نوع دیگر VM ها Exokrenel می باشد . ExoKernel که شبیه virtual machine monitor یا VM370 می باشد با این تفاوت که نیازی به لایه نگاشت آدرس ها ندارد. نگاشت یعنی فرض کنید رم را تقسیم بندی کرده ایم از 0 تا 1000 برای ماشین A،از 1000 تا 2000 برای ماشین B و ... حال وقتی ماشین A می گوید آدرس 200 همان 200 اما وقتی ماشین B می گوید 200 یعنی 1200. یعنی آدرس ها باید ترجمه شود این کار در VM توسط خود VM انجام می شد که باعث کاهش سرعت می شد اما درExoKernel این کار به صورت خودکار توسط سخت افزار مدیریت حافظه انجام می شود.

وقتی برنامه ای روی CMS اجرا می شود وقتی یک درخواستی از CMS داشته باشد به عنوان system call فراخوان سیستمی می کند. تا اینجا مثل تله در ماشین حقیقی است ولی وقتی که CMS می خواهد سراغ حافظه یا I/O و ... برود نمی داند که زیرش VM است و مستقیما می خواهد به سراغ حافظه برود در صورتی که مثلا I/O را VM در اختیار یک

سیستم دیگر قرار داده است. در اینجا جلوی این عمل باید توسط CPU گرفته شود این کار با مکانیزم تله هایی از نوع exception انجام می شود. در واقع اینجا دو تله داریم که باعث می شود مقداری سرعت کمتر شود.

به طور کلی VM مزایایی هم دارد از جمله ماژولاریتی ، امنیت بیشتر به این دلیل که ماشینهای مجازی واقعاً از هم مستقل هستند و همچنین قابلیت حمل آن بالاست اما نسبت به منولیتیک کند تر هستند.



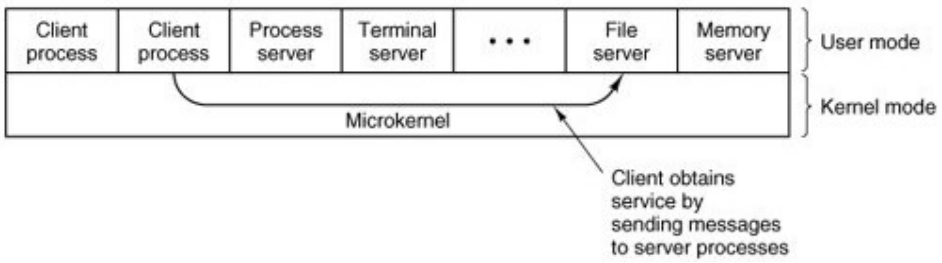
21:30

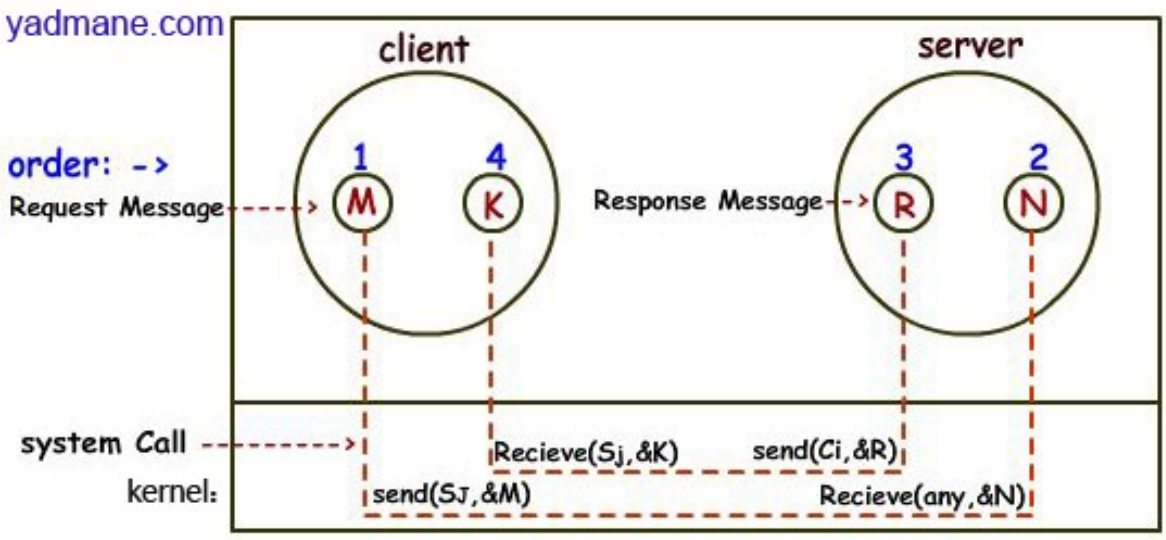
5. مدل مشتری خدمتگزار - ریزهسته یا Micro Kernel - Client –Server

دراین مدل هسته را تا حد امکان کوچک می کنیم وبسیاری از وظایف را به ماژول هایی مجزا، تفکیک شده وخارج ازهسته به نام سرور می سپاریم. سرور ها در مد کاربر کار می کنند.

سرور یک فرایند است درمد کاربر نه یک ماشین . ما می توانیم n سرور روی یک ماشین در کنار هم داشته باشیم.

رفتار server و client به صورت Request / Reply یا Message Passing است.





با توجه به شکل نحوه درخواست و پاسخ به ترتیب زیر می باشد:

1- ابتدا کلاینت یک system call ارسال می کند .آدرس پیام M است.

2- فرآیند مقابل بایستی یک بافر جهت پیام ایجاد کند و از هسته درخواست پیام می کند . (Recive). در اینجا Recive از any صورت می گیرد چون سرویس دهنده به همه سرویس می دهد. حال که این دو درخواست صورت گرفت (کلاینت send کرد و سرور Recive) هسته پیام را جابجا می کند.

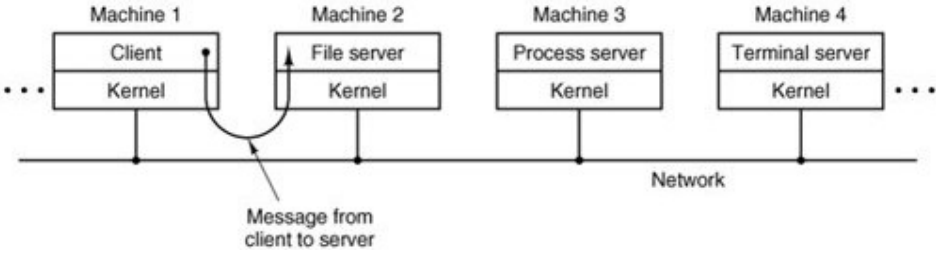
3- زمانی که سرویس دهنده پاسخ را می خواهد ارسال کند باید پیام را برای کسی بفرستد که از آن گرفته یعنی نباید برای any ارسال کند و اکنون آدرس طرف مقابل را دارد در اینجا سرویس گیرنده نیز recieve می کند و هسته پیام را جابجا می کند.

در طی مراحل بالا 4 تله داریم ضمن اینکه فرض کنید خواسته باشیم سرور یک فایل را بخواند که در این صورت فایل سرور برای خواندن باید سراغ دیسک برود که نمی تواند چون در مد کاربر است پس نیاز به system call دارد . پس تعداد تله ها بیش از 4 است که باعث کندی کار می شود. در نهایت اینکه حدود 20٪ کارایی این روش از سیستم های monolithic کمتر است.

37:00

در سیستم های **مشتری خدمتگزار** چون هسته بسیار کوچک است Debug کردن آن ساده است. و احتمال Bug آن کم است. این سیستم ها بسیار مناسب برای سیستم های توزیع شده می باشند. زیرا سرور ها بسیار مستقل هستند. به دلیل

استقلال بسیار سرورها شما می توانید هرکدام از سرور هایی که روی سیستم عامل قرار دارد و مناسب نبود غیر فعال و سرور دیگری جایگزین آن کنید .



سیستم های های مشتری خدمتگزار به دلیل استقلالی که سرور ها از یکدیگر دارند بسیار مناسب برای سیستم های توزیع شده می باشند.

درست است که در مدل لایه ای نیز استقلال بین لایه ها وجود داشت اما درمدل لایه ای آن لایه ها در هسته بودند اما در مدل مشتری خدمتگزار سرورها در مد کاربر هستند چون در مد کاربر هستند برنامه نویس حرفه ای می تواند خودش سرور جدید بنویسد یا بهبود ببخشد و آن را روی ماشین نصب و استفاده کنم. این استقلال باعث می شود که بتوانیم سرورها را روی ماشین های مختلف قرار دهیم که در این صورت ماشین های مختلف با Message Passing با هم ارتباط برقرار می کنند.

حالا که هسته را اینقدر کوچک کرده ایم و وظایف را بین سرور ها تقسیم کرده ایم پس هسته چه کاره است؟

چهار وظیفه هسته :

1ـ تبادل پیام بین مشتری وخدمتگزار

2ـ تخصیص پردازنده وایجاد چند برنامگی

3ـ بخش سطح پایین مدیریت حافظه مانند برنامه ریزی رجستر های سخت افزار مدیریت حافظه

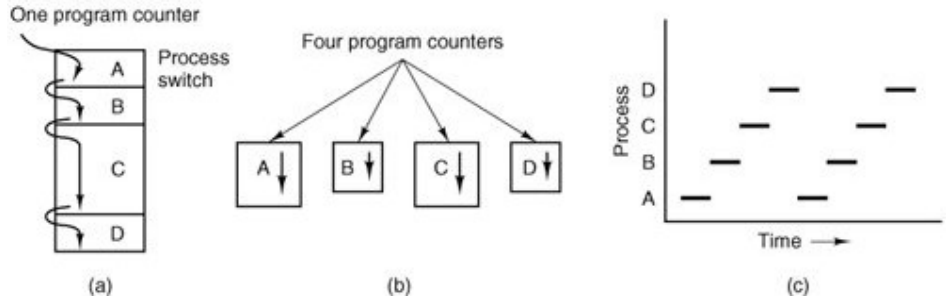
4ـ بخش سطح پایین I/O که برنامه ریزی رجیسترهای ویژه کنترل کننده رابرعهده دارد.

47:00

فصل دوم

فرآیندها

یاد آوری: درتک پردازنده ای شبه توازی داریم مثل اشتراک زمانی در واقع Concurreney است.



ایجاد فرایند:

1. توسط سیستم عامل و در هنگام راه اندازی سیستم ، چندین فرایند ایجاد می شود.
2. ایجاد فراخوان سیستمی ایجاد فرآیند جدید توسط فرآیند در حال اجرا مثل fork در unix
3. درخواست کاربر برای ایجاد فرآیند جدید
4. آغاز یک کار دسته ای (batch proccess)

اتمام فرایند :

1. خروج عادی (اختیاری) Normal exit
2. خروج با خطا (اختیاری) error exit
3. خطای مهلک (اجباری) fatal error مثلاً می خواسته به فضای حافظه دیگران دسترسی داشته باشد.
4. کشته شدن توسط فرایند دیگرkill

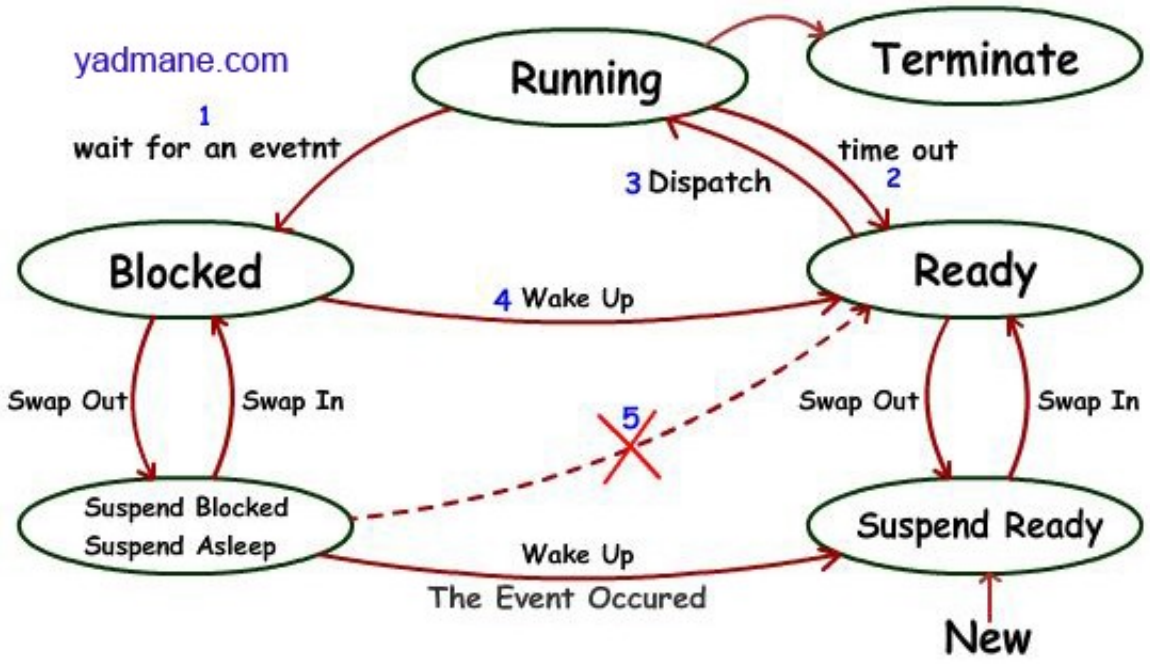
Orphan Process

برخورد با فرآیند های یتیم orphan process (فرآیندی که پدرش kill می شود.) در سیستم عامل های مختلف متفاوت است. تننباوم با اینگونه فرآیند ها مخالف است و حتی در بعضی سیستم عامل ها در دوره های زمانی مختلف دنبال فرآیند های یتیم در سیستم می گردند و همه را می کشند و حتی ممکن است هر چه فرآیند فرزند هست را بکشند.

1:37:00

ابتدای صدای 2 جلسه 4

وضعیت های مختلف فرایندها:



Running : فرایندی است که در حافظه اصلی است و درحال اجراست .

Ready : فرآیندی است که CPU ندارد، و منتظراست تا CPU را به آن بدهند تا اجرا شود.

Block : فرآیندی است که منتظر یک رویداد است مثلا پایان عمل I/O یعنی اگر CPU راهم به آن بدهند کاری نمی تواند انجام دهد.

وضعیت فرایندها: (با توجه به شکل)

1. فرایند برای یک عمل مثل I/O بلوکه می شود wait for an event
2. Time out high priority arrival
3. Dispatch زمانبند فرایند جدید را انتخاب می کند
4. wake up(the event Dccured)
5. نمی تواند رخ دهد چون دو واقعه همزمان باید رخ دهد.

سوال: سیستم عامل از کجا فرآیند های ready را تشخیص می دهد؟

تشخیص فرآیند های آماده با استفاده از صف ready queue می باشد که خود سیستم عامل آن را ایجاد و مدیریت می کند.

نکته: زمان بندی که کار ها را برای تبدیل شدن به فرآیند انتخاب می کند job schedullar گویند.

سوال: چه کسی فرآیند آماده را برای اجرا انتخاب می کند؟

پاسخ: CPU schedullar

- پس از انتخاب فرآیند توسط CPU schedullar باید عمل Dispatch برای قرار دادن دیتای لازم جهت اجرای فرآیند صورت گیرد. این دیتا را از PCB می خوانند و موقعی نیز که بخواهند CPU را از آن بگیرند مجدداً اطلاعات رجیستر ها را در PCB فرآیند ذخیره می کنند.
- سخت افزاری به نام ساعت وجود دارد که سر کوانتوم تعیین شده اعلام time out می کند.
- پس از time out اطلاعات فرآیند را در PCB ذخیره کرده و فرآیند مجدد به صف ready می رود.
- اگر وسط کوانتوم فرآیند منتظر رویدادی (تله ورودی – خروجی یا page fault) شود فرآیند block می شود.
- فرآیند بلوکه شده در صورت رخداد رویداد مورد نظرش بیدار می شود که اصطلاحا به آن wake up می گویند.
- در صورتی که جای کافی برای فرآیند ها نباشد از بین فرآیند های بلوکه شده و در صورتی که فرآیند بلوکه شده در حافظه وجود نداشت از بین فرآیند های ready یکی را به روی دیسک منتقل می کنند که به آن swap out می گویند. فرآیند اگر ready باشد و به دیسک منتقل شود suspend ready و اگر block باشد و به

دیسک منتقل کنیم suspend block گویند.

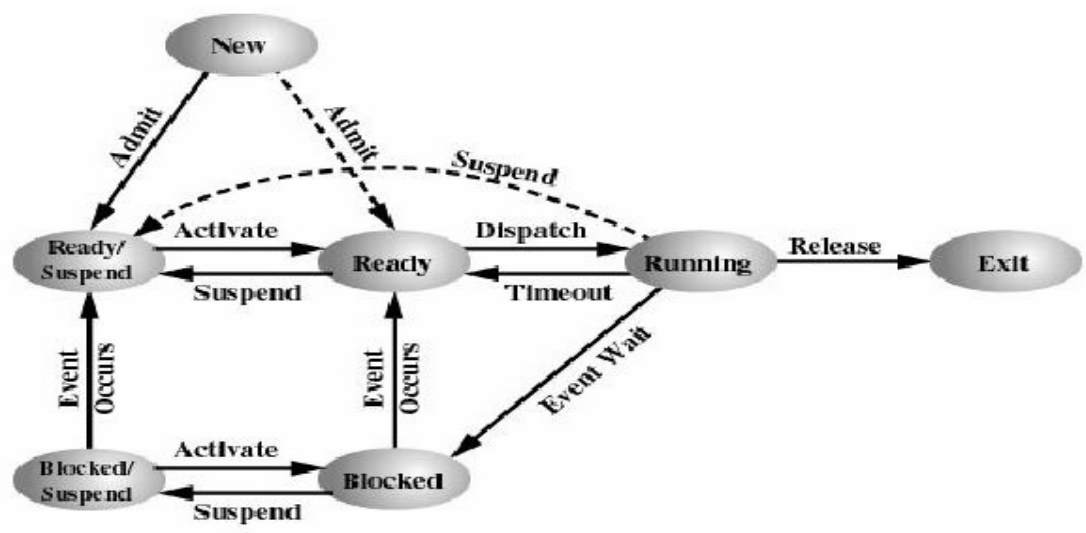
- در صورتی که فرآیند suspend block باشد و فرآیند دلخواهش رخ دهد به suspend ready می رود که این عمل wake up می باشد.

توجه:

امکان انتقال از suspend block به ready وجود ندارد چون به دو واقعه نیاز دارد:

1) رویداد مورد نظر رخ دهد
2) memory scheduler آن را جهت انتقال به حافظه انتخاب کند. که هیچ گاه دو رویداد رخ نمی دهد.

- اگر در حافظه جا برای فرآیند خالی شود . از بین فرآیند های suspend اولویت با suspend ready است.
- اگر فرآیندی exit کند پس در حال اجرا بوده پس از running به terminate می رود.
- پس از مرگ فرآیند باید بساط آن جمع شود.(pcb , mail box , ...)
- job های تازه وارد ، وارد صفی می شوند و job schedullar یا admission schedullar آنها را برای پذیرش فرآیند مدیریت می کند و وقتی یک کار انتخاب می شود به suspend Ready می رود.



1:00:00

مراحل Context Switch (تعویض متن) (تعویض فرایند)
بعضی ها به آن process switch نیز می گویند.

سیستم Time sharing است وفرایند درحال اجراست. Timer درپایان کوانتوم اطلاع می دهد. حال مراحل تعویض متن را بررسی می کنیم.

مراحل تعویض متن:

1) Timer سیگنال درخواست وقفه را به PIC می فرستد.
2) PIC وقفه ی با اولویت بالاتر را انتخاب می کند.
3) PIC به CPU درخواست وقفه می فرستد.
4) CPU دستورالعمل جاری را تمام می کند.
5) CPU رجیسترهای مهم (PSW,PC) رادرStack فرایند جاری ذخیره می کند.
6) CPU به PIC یک Interupt Ack می فرستد.
7) PIC شماره وقفه را روی Databus می گذارد.
8) CPU شماره وقفه را از روی Databus برمی دارد.
9) CPU به سراغ خانه N ام بردار وقفه می رود و از آنجا آدرس ISR مربوط به Timer را بر می دارد.
10) CPUاین را درPC قرارمی دهد.
11) ازKernel Mode به User Mode سوئیچ می کنیم.

تمام این مراحل توسط سخت افزار رخ می دهد ازاین لحظه به بعد سیستم عامل دست به کار می شود.

وظایف OS درقبال عملیات Switch :

12) سیستم عامل کلیه رجیسترهای فرایند قبلی رادرPCB فرایند ذخیره می کند.

OS درصورت تمایل می تواند PSW و PC را نیز ازStack فرایند pop کرده و در PCB قرار می دهد.

نکته: از آنجایی که اکنون PC به ابتدای Interrupt Routine اشاره می کند برای نگهداشتن PC برنامه قبلی دیگر نمی توانیم محتوای فعلی PC را در PCB ذخیره کنیم بلکه محتوای PC را که قبلاً (در ردیف 5) درStack ذخیره کرده

بودیم را **در صورت دلخواه** pop کرده و در PCB ذخیره می کنیم. به این دلیل دلخواه است که خود استک فرآیند نیز در PCB ذخیره می شود.

13) درصورت لزوم تمامی فیله های فرایند قبلی update شود. مثلا وضعیت فرآیند ازRunning به Ready تغییرکند. کلا PCB را update می کنیم . بساط قبلی را به جز SP جمع می کنیم. که در مرحله بعد بساط آن نیز جمع می شود.

14) Stack عوض می شود. و SP به بالای Stack موقتی سیستم عامل اشاره می کند. خود OS نیز درجهت فرایندهای تودرتو به Stack نیازدارد.

15) تابع CPU Scheduler راصدا می زنیم.

16) CPU Scheduler براساس الگوریتم زمانبندی فرایند بعدی را انتخاب می کند.

17) شماره فرایند انتخاب شده دریک متغیر سراسری قرار می گیرد وبازگشت (return) می کند. برای این دریک متغیرglobal می گذاریم که همه (Dispatcher , ...) متوجه شوند نوبت چه فرایندی است.

18) به Interrupt routine برمی گردیم وStack فرایند جدید برپا می شود. SP را ازPCB فرایند جدید به CPU منتقل می شود.

19) رجیسترهای مهمی مثل PSW, PC اگرازStack به PCB منتقل شده اند را دوباره به Stack برگردانده می شود.

نکته: چرا PC و PSW را به Stack می بریم و به CPU منتقل نمی کنیم؟

چون اگر PC را به CPU منتقل کنیم CPU دیگر به interrupt routine اشاره نمی کند و نمی تواند مابقی کار های interrupt routine را انجام دهد.

20) سایر رجیسترها ازPCB به CPU بارگزاری می شود.

21) سایر فیلد های PCB را در صورت لزوم Update می کنیم مثلاً وضعیت از Ready به Running

22) بازگشت از وقفه شامل تغییر مد ازهسته به مد کاربر و رجیسترهای مهم را ازstack برمی دارد ودرCPU قرارمی دهد. به این ترتیب به ادامه اجرای یک فرایند جدید بازگشتیم.

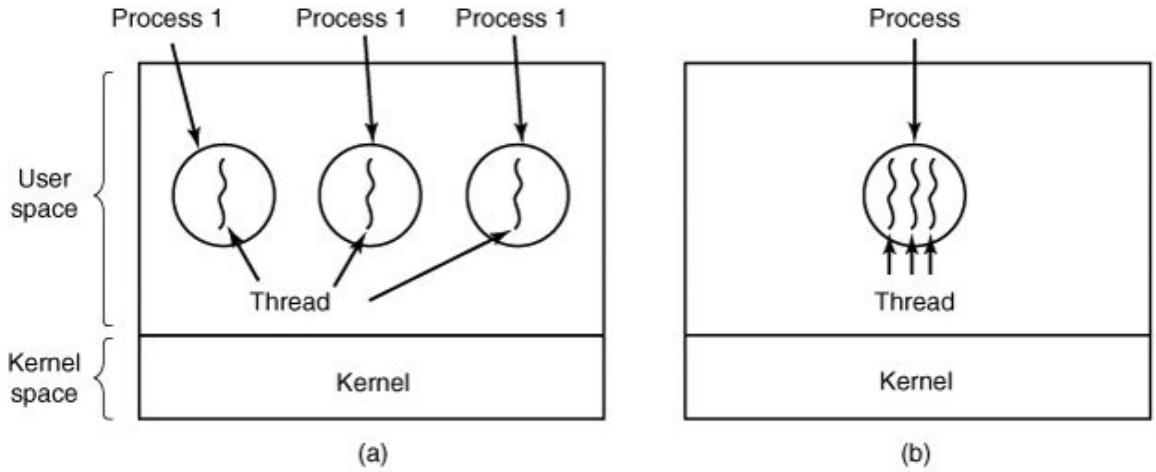
نکته:

CPU درک نمی کند که Stack عوض شده است و نمی فهمد برنامه عوض شده است. ما فقط از مکانیزم وقفه استفاده کردیم و تمام این تعویض Stack و... توسط سیستم عامل انجام می شود.

1:45:00

نخ ها (Threads)

درگذشته سیستم های Multi Threading نبود ودر واقع برنامه ها تک نخی بودند (Single thread)



اگر درک درستی از چند نخی نداشته باشیم نمی توانیم به درستی از آن ها استفاده کنیم. اگر یک برنامه به درستی به نخ های مستقل تقسیم نشده باشد نخ جز سر بار زمانی چیز دیگری نخواهد داشت.

چرا نخ ها مورد توجه قرار گرفتند:

گاهی اوقات برنامه ها باصطلاح زیادی دانه درشت هستند و می خواهیم به دلایلی آنها را بشکنیم. این دانه های ریزتر را که sequence های مجزا و همروند هستند را نخ می گوییم.

هرنخ رامی توانیم به تسبیح تشبیه کنیم و هر دانه آن یک instruction می باشد. همانطور که درتسبیح دانه ها را یکی یکی می شماریم . همروندی مثل این است که چند تسبیح داریم ومثلا با هرکدام یک ذکرمی گوییم. زمانی می توانیم بگوییم برنامه چند نخی است که هر بخش مستقل وبه صورت همروند قابل اجرا باشد ونیازی به یکدیگرنداشته باشند.

Function ها رابا نخ ها اشتباه نگیرید. مثال هایی مثل ضرب ماتریس ها مثال خوبی برای چند نخی نیست چرا که ضرب ماتریس ها را چه به صورت چند نخی وچه به صورت تک نخی انجام دهیم باعث افزایش سرعت نمی شود چرا که I/O limited نیست و عملیات ها overlap نمی شوند. ضمن اینکه به خاطر سوئیچ ها در چند نخی در این مثال سربار زمانی هم داریم.

File server می تواند مثال خوبی برای چند نخی باشد چرا که مرتبا با I/O سروکاردارد ومی تواند هروقت که به خاطر درخواست I/O یکی ازنخ ها خوابید به سراغ نخ بعدی برود و کل فرآیند نخوابد. دراین گونه موارد چند نخی باعث افزایش کارایی می شود زیرا باعث overlap شدن عملیات I/O و پردازش می شود. یعنی چند نخی حتی درتک پردازنده ای هم باعث افزایش کارایی می شود. تننباوم عقیده دارد چند نخی به ندرت باعث کاهش کارایی می شود که آن هم به دلیل ضعف برنامه نویس است .

هرنخ Stack خاص خود را دارد اما نخ های یک فرایند فضای داده مشترک دارند.

جلسه پنجم ۱۳۸۸/۵/۱۹ ساعت ۱۲:۰۰ با حدود 30 دقیقه تاخیر

فواید چند نخی :

- در CPU های موازی انجام موازی کارها باعث افزایش سرعت وکارایی می شود.
- نخ ها از یکدیگر مستقلند واگر یک نخ بخوابد نخ های دیگر به فعالیت خود ادامه می دهند وکل فرایند نمی خوابد که باعث افزایش کارایی می شود.
- نخ ها به فضای آدرس همدیگر دسترسی دارند.
- افزایش Modularity (ازدید مهندسی نرم افزار)
- بعضی برنامه ها ذاتاً چند عاملی (چند نخی) هستند . مثل بازی فوتبال

اگرچه نخ باید درفرایند اجرا شود اما نخ وفرایند آن، دومفهوم متفاوت اند ومی توانند به صورت مجزا دیده شوند.

3:30

1. درچند پردازنده ای ← توازی ← سرعت
2. درتک پردازنده ای ← هم پوشانی پردازش یک نخ با نخ بلوکه شده به خاطر I/O ← سرعت
3. ازدید مهندسی نرم افزار ← افزایش ماژولاریتی
4. بعضی ازفرایند ها ذاتا ازعوامل مستقل، مجزا وهم روند تشکیل می شوند (مثل شبیه سازی فوتبال) ←افزایش سادگی برنامه نویسی
5. دسترسی نخ ها به فضای داده مشترک ومتغیرهای سراسری فرایند ← ارتباط نخ ها سبکتر وسریعتر می شود چرا که نیاز به تله نداریم ← سرعت

استفاده ازتله به دلیل اینکه باید ازهسته عبور کنیم بسیار کار را کند می کند.

علت اینکه فرایندها نمی توانند ازفضای داده وآدرس مشترک استفاده کنند این است که به دلیل امنیتی این اجازه راندارد.

نخ ها تنها روش همروند کردن کارها وفرایند ها نیست. قبلا درمورد Fork توضیح داده شده که چگونه فرایندها با ایجاد فرزند مستقل این عمل را انجام می دادند. اما این روش خاصیت پنجم که برای نخ ها گفته شد را ندارد درنتیجه نخ ها سبکتر وهمروندی درنخ ها سبکترو کارآمد تراست.

ایجاد ــ حذف واداره نخ ها برعهده کیست؟

1. **نخ های سطح هسته** (Kernel Level Mode) : که هسته سیستم عامل این کار را انجام می دهد.
2. **نخ های سطح کاربر**(User Level Mode) : درواقع Package ایجاد ، حذف واداره نخ ها درمد کاربر اجرا می شود وبه آن Run Time System می گویند.
3. **ترکیب هسته وکاربر**

نخ های سطح کاربر همه چیزشان درسطح کاربراست وهسته هیچ اطلاعی ازوجود نخ ها ندارد. در واقع سیستم عامل Single Thread است ولی بر روی آن یک سیستم multi Thread ایجاد کرده ایم.

20:20

مزایای نخ های سطح کاربر:

1. ایجاد وحذفشان سبکتر، سریع تر و ارزان تراست. زیرا تله ندارد پس برای ایجاد وحذف نیازی به ارتباط با هسته ندارد.
2. ارتباط نخ ها سریعتر است (ازطریق حافظه مشترک بدون تله)
3. همگام سازی نخ های سطح کاربر سریعتر است (بدون تله) برای خوابیدن ، بیدارکردن بدون دخالت هسته سیگنال به هم می فرستند. دریک کلام نخ های سطح کاربر بسیار سبک وساده است .

معایب نخ های سطح کاربر :

سرچشمه معایب دراین است که هسته نخ ها را نمی بیند پس :

1. اگر یک نخ یک فراخوان سیستمی مسدود کننده انجام دهد، هسته اشتباهاً کل فرایند را مسدود می کند.
2. اگر برای یک نخ نقص صفحه رخ دهد هسته کل فرایند را مسدود می کند.
3. در سیستم های چند پردازنده هسته OS نخ ها را نمی بیند وبه این دلیل نمی تواند نخ ها را بین پردازنده ها پخش و زمانبندی کند وبه این دلیل توازن بار پردازش نداریم .

مزایا نخ های سطح هسته (هسته نخ ها را می بیند) :

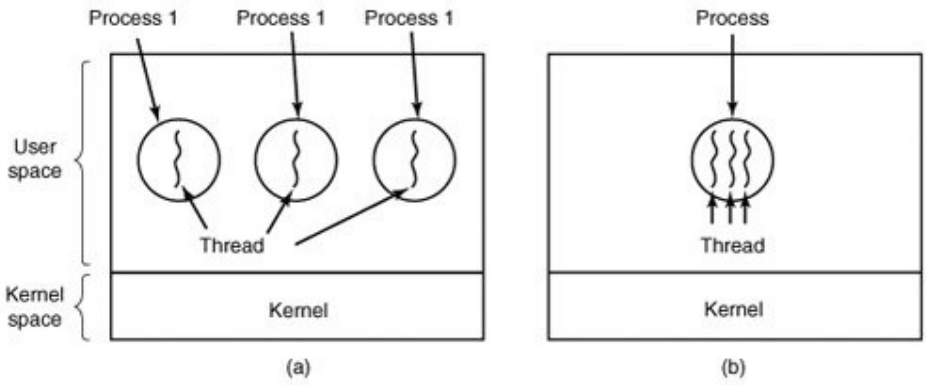
1. اگر یک نخ یک فراخوان سیستمی مسدود کننده انجام دهد فقط همان نخ مسدود می شود.
2. اگر برای یک نخ نقص صفحه رخ دهد فقط همان نخ مسدود می شود.
3. درسیستم های چند پردازنده ای هسته می تواند نخ ها را بین پردازنده ها تقسیم کند وتوازن بار پردازشی خواهیم داشت.

معایب نخ های سطح هسته :

1. ایجاد و حذف آنها کندتراست زیرا تله دارد.
2. همگام سازی نخ ها کند تراست .

نخ های سطح کاربر وهسته (ترکیب روش ها)

فقط ایجاد وحذف نخ ها را به سیستم عامل خبرمی دهیم ولی بقیه کارها درسطح کاربر انجام می شود مثل ارتباط و هماهنگ سازی و .....



36:30

موارد مخصوص نخ	موارد مخصوص فرآیند
شماره برنامه رجیستر ها پشته وضعیت	فضای آدرس متغیر های سراسری فایل های باز فرآیند های فرزند هشدار های معلق سیگنال ها و اداره کننده های سیگنال اطلاعات حسابداری

45:50

ارتباط بین فرایند ها یا ارتباط بین نخ ها

مباحث :

1. Data Communication ← تبادل داده ها (از بالا به پایین از سریع به کند)
   - Share Memory
   - Message Passing
   - Pipe
   - File sharing
2. Synchronization یا همگام سازی (ارتباط از نوع داده ای نیست)
3. Race Condition (شرایط رقابتی)
4. Mutual Exclusion انحصار متقابل

58:00

همگام سازی :

مثال:

یک گروه که با هم خانه می سازند باید با هم همگام باشند نه این که همزمان کارکنند مثلا قبل ازگچ کاری نمی توان رنگ کاری کرد.

همگام سازی یعنی با هم هماهنگ باشند، مچ باشند، همگام باشند، ازهم سبقت نگیرند. زمان هرعمل باید مشخص باشد وآن عمل به موقع وسرجای خودش انجام شود.

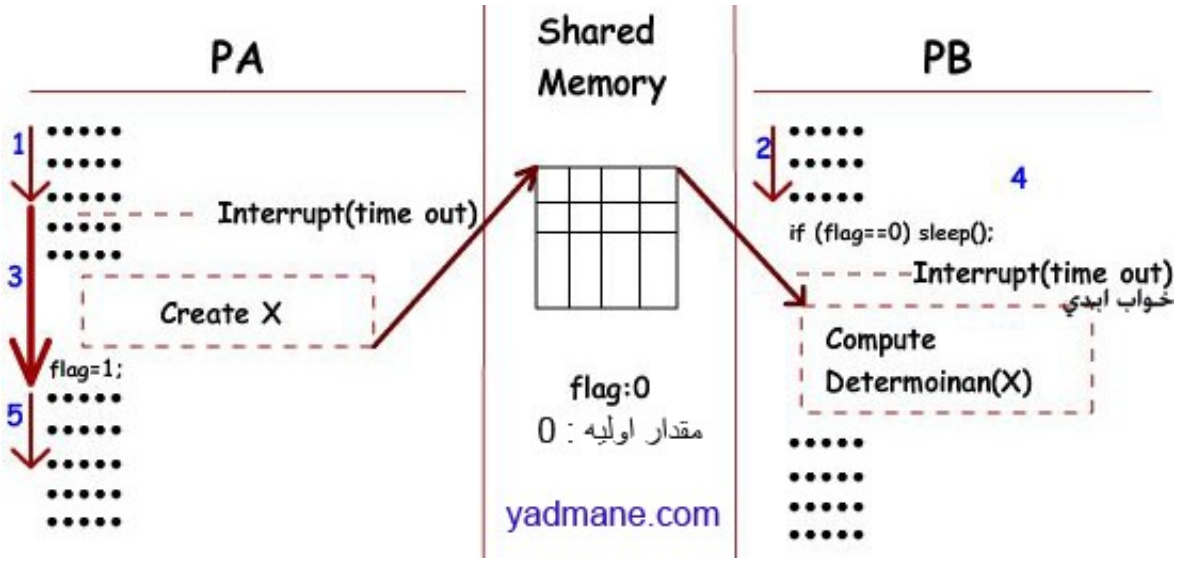
1:07:00

مثال - فرض کنید دوفرایند B وA دریک سیستم اشتراک زمانی به صورت همروند اجرا می شوند فرایند A دربخشی از کد خود ماتریس X راتشکیل می دهد. فرایند B نیزدربخشی از کد خود درترمینان ماتریس X را محاسبه می کند. این دو فرایند رابه گونه ای بنویسید که سنکرون باشند (یعنی B ازA سبقت نگیرد) .

شرایطی که برای مثال باید رعایت شود :

1. ازBusy waiting استفاده نکنید.

2. استفاده ازحافظه های مشترک مجازاست.
3. برای انتظار ازفراخوان های سیستمی Sleep و Wake up استفاده کنید. یعنی اگر کسی خواست منتظر شود لوپ نزند، بخوابد.
4. اگرفرایندی بیدار باشد سیگنال Wake up ازدست می رود.



نکته:

- ما نمی توانیم فرآیند های دیگر را بخوابانیم بلکه هر فرآیند می تواند خودش بخوابد.
- فرآیند ها می توانند فرآیند های دیگر را بیدار کنند.

درشکل بالا چه چیزهایی را می بینیم :

1. Data Communication ← حافظه مشترک مثل flag
2. همگام سازی
3. شرایط رقابتی (هدف همگام سازی بود مشکل شرایط رقابتی پیش آمد)

چرا این روش غلط است ؟

به شماره بندی ها دقت کنید اگر ترتیب اجرا مثل شماره بندی ها باشد PB به خواب ابدی می رود. یعنی اگر درمرحله 2 فلگ را چک کند و Time out شود در مرحله 3، PA سیگنال Wake up می فرستد. اما درمرحله 4 فلگ را قبلا چک کرده و حال به خواب می رود، خوابی ابدی.

نکته بسیار مهم :

دراینجا بن بست رخ نداده چراکه B منتظرA هست اما A منتظرB نیست اگرهردو منتظر باشند بن بست است.

نکته مهم :

در اینجاRace Condition باعث خواب ابدی شد.

نکته :

قرار نیست که در هر سری اجرای این برنامه این شکل (شرایط رقابتی) پیش بیاید. کاملا اتفاقی است. ازآن سری Bug هایی است که تشخیص آن سخت است.

1:44:30

نکته وسوال : طبق قرارداد زمانی که وقفه می آید، باید دستورالعمل جاری تمام شود و سپس به وقفه رسیدگی کنیم پس چرا در مثال قبل فرایند B در بین شرط به Sleep می رود؟

جواب: این است که نباید به زبان C فکرکنیم . بلکه باید به زبان ماشین فکرکنیم.

کد قسمت 4 مثال قبل به زبان اسمبلی:

1. mov reg1,flag
2. cmp reg1, 0 خطر بین این خط و خط قبل است
3. jnz cmp-det سه خط بالا معادل یک دستور در زبان سطح بالاست
4. call sleep فراخوان های سیستمی معمولا اتمیک هستند
5. cmp-det:

1:54:00

باید دنبال پنجره خطر بگردیم :

یعنی باید تک تک دستورات را بررسی کنیم که اگر وسط اجرای آن وقفه بیاید خطرناک است یا نه.

نقطه 1 : اگردراین نقطه وقفه بیاید خطرناک است چرا که flag درفرایند دیگرتغییر کرده اما رجیستر قبلا مقدارگرفته است وباعث اشتباه می شود.

نقطه 2: اگروسط CMP هم بیاید خطرناک است چرا که این بارCompare هم کرده ایم.

نقطه 3: JNZ هم خطرناک است چون مقدار program counter تغییر می کند.

فرخوان های سیستمی اتمیک هستند (غیرقابل شکست) (یاهیچی یا همه اش)

دستورات 1و 2و3 در ناحیه بحرانی (critical section) قرار دارند.

Critical Section **(مکان):** قسمتی از کد برنامه که با عوامل مشترک رقابت زا سر و کار دارد.

همیشه همه دعواها بر سر یک عامل مشترک پیش می آید.

شرایط رقابتی(زمان): هرگاه دو یا چند فرایند همزمان وارد ناحیه بحرانی کدشان بشوند شرایط رقابتی می شود.

رفتن به ناحیه بحرانی مشکلی ندارد (خطرناک نیست) بلکه نباید همزمان وارد ناحیه بحرانی شد.

2:15:00

اگرشرایط رقابتی شده است آنگاه :

2 حتما عامل مشترک داشته ایم.

2 حتما همزمان وارد شده ایم.

2 حداقل یک نفرنویسنده بوده.

نکته: عکس این قضیه همیشه درست نیست.

مثل راه حل های : Strict All ـ Swap ـ Tsl ـ Peterson ـ Decker ـ

هرعامل مشترکی رقابت زا نیست. اما اگر رقابت زا شود ناحیه بحرانی می شود.

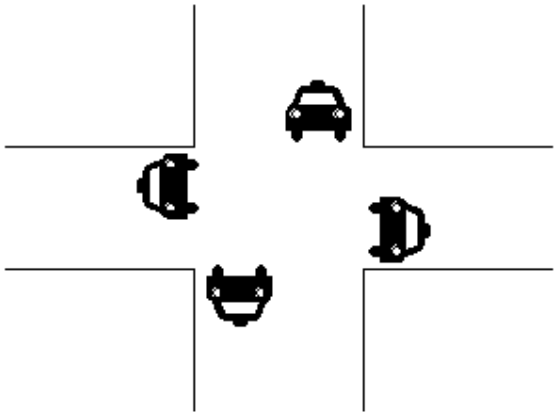
2:23:30

2:25:00

انحصارمتقابل ، دوبه دو ناسازگاری، مانعه الجمعی Mutual exclusion

مکانیسمی که باعث شده فرآیند ها همزمان وارد ناحیه بحرانی نشوند.

اگر با هم وارد ناحیه بحرانی شوند می گویند Interlock (تداخل) (conflict) شده است.



توجه:

Mutual exclusion در اینجا این است که هر چهار راه یک چراغ قرمز دارد که مانع از این می شود که دو ماشین وارد CS شوند

راه حل خوب 4 شرط دارد : (درمورد شرایط رقابتی) ص 99 کتاب

3. هیچ دوفرایندی باهم وارد ناحیه بحرانی نشوند (انحصار متقابل)
3. هیچ فرضی در مورد سرعت وتعداد CPU ها نداشته باشیم. (generality)
3. هیچ فرایندی در بیرون از ناحیه بحرانی خود امکان بلوکه کردن فرایندهای دیگر را نداشته باشد (کسی که قصد ورود را ندارد نباید جلوی بقیه را بگیرد) (شرط پیشرفت Condition Progress)
3. هیچ فرایندی نباید تا ابد منتظر ورود به ناحیه بحرانی خود شود. (bounded waiting)

مثل دو نفر که سر ورود با هم تعارف می کنند و هیچ یک حاضر نیست قبل از دیگری وارد شود. یا مثل وقتی که برای برخی افراد اولویت بگذاریم و آن قدر آنها بیایند که هیچ موقع نوبت به ما نرسد.

راه حل ها :

1. از کارانداختن وقفه ها : وقتی وارد CS می شویم Disable Interrupt را اجرا می کنیم و زمانی که ازCS خارج شدیم Enable Interrupt را اجرا می کنیم.

معایب :

- مثل این است که اسلحه را به دست یک دیوانه بدهیم این کار امنیت رابه خطر می اندازد چرا که تضمینی وجود ندارد وقتی که برنامه کاربر وقفه ها راغیر فعال کرد مجددا آن را فعال کند.
- درچند پردازندگی این کاربی معناست چرا که این مشکل ارتباطی به وقفه ندارد زیرا درچند پردازندگی مشکل به دلیل ذات توازی است . یعنی ممکن است هر دو فرایند همزمان وارد ناحیه بحرانی شوند و غیر فعال کردن وقفه ها پس از ورود هر دو آنها به ناحیه بحرانی مشکلی را حل نمی کند.

2:52:30

2. متغیر قفل :

یک متغیر سراسری مشترک به نام Lock می گیریم. اگر صفر بود هیچ فرایندی درCS نیست واگر یک بود فرایندی درCS است. اگرزمانی که یکی از پردازه ها قفل راچک کرده وبازبوده وقفه بیاید قبل ازاین که قفل راتغییر دهد پردازه بعدی که می آید نیز قفل راباز می بیند ووارد CS می شود ودرنتیجه هردو واردCS می شوند. درحالت موازی هم هردو فقط به متغیر Lock نگاه می کنند و وارد CS می شوند ورقابت ایجاد می شود.

```
while(turn){
    while(lock==1)   (1)
        lock=1;
    critical_region();
    lock  = 0;
    noncritical_region();
}
```

(1) اگر زمانی که یکی از پردازنده ها قفل را چک کرده و باز بود قبل از اینکه قفل را تغییر دهدCPU از آن گرفته شود. پردازه بعدی که می آید نیز قفل را باز می بیند و وارد CS می شود و در نتیجه هر دو وارد CS می شوند.
هم در time sharing و هم در پردازنده های موازی این مشکل پیش می آید.

در این مثال راه حل (قفل) خودش Critical section شده.

3:02:30

3. تناوب قطعی : ورود به ناحیه بحرانی به صورت نوبتی امکان پذیر است یعنی وقتی یک فرایند وارد ناحیه بحرانی می شود قفل را عوض می کند و فقط فرایند دیگر می تواند بعد از آن وارد ناحیه بحرانی شود حتی اگر فرایند دوم تا مدت ها نیاز به ورود به ناحیه بحرانی نداشته باشد و فرایند اول محتاج آن باشد.

```
while(TRUE){
      while(turn != 0)
      critical_region();
      turn  = 1;
      noncritical_region();
}
```

(a)

```
while(TRUE){
      while(turn != 1)
      critical_region();
      turn  = 0;
     noncritical_region();
}
```

(b)

معایب :

1. شرط پیشرفت را نقض می کند یعنی یک فرایند که نه درناحیه بحرانی است ونه برای ورود به ناحیه بحرانی رقابت می کند می تواند یک فرایند دیگر را بلوکه کند.
2. Busy Waiting دارد. البته این مشکل را اکثر الگوریتم های حل این مشکل دارد.

نکته: این روش را می توان به چندین فرایند نیزتعمیم داد.

جلسه ششم ۱۳۸۸/۵/۲۶ ساعت ۱۲:۰۰

4. راه حل پترسون:

PA :

Enter- region (0);

Critical-Section();

Leave-region(0);

non-critical-section (0);

PB :

Enter- region (1);

Critical-Section();

Leave-region(1);

non-critical-section (1);

```
#define FALSE 0
#define TRUE  1
#define N     2                       /* number of processes */

int turn;    /* whose turn is it?     متغیر گلوبال برای نوبت دهی استفاده می شود */
int interested[N];                    /* all values initially 0 (FALSE)*/
void enter_region(int process)        /* process is 0 or 1 */
{
     int other;                       /* number of the other process */
     other = 1 - process;             /* the opposite of process */
     interested[process] = TRUE;      /* show that you are interested */
     turn = process;                  /* set flag */
     while (turn == process && interested[other] == TRUE) /* null statement */;
}
void leave_region(int process)        /* process: who is leaving */
{
     interested[process] = FALSE;     /* indicate departure from critical region */
}
```

برای چک کردن مشکلات الگوریتم باید جا هایی که با متغیر های سراسری کار می کند را چک کنیم . یعنی قبل و بعد از آنها وقفه بدهیم.

1. هیچگاه هر دو پروسس برای بیش از یک کوانتوم زمانی در Busy Waiting نمی مانند حداکثر هر دو پروسس در یک پریود در Busy Waiting می مانند.
2. برای Trace باید درتک تک خط ها وقفه بدهیم وبررسی کنیم به مشکل می خورد یا نه .
3. الگوریتم پترسون قابل تعمیم به چند فرایند است.
4. الگوریتم پترسون Starvation (قحطی ـ گرسنگی) ندارد. چون نوبت را رعایت می کند.
5. الگوریتم پترسون کوتاهترین وبهترین ودرست ترین الگوریتم نرم افزاری است پس اگرالگوریتم نرم افزاری کوتاهترازآن داده شد حتما ایراد دارد.

6. در این الگوریتم یک فرآیند می تواند چند بار پشت سر هم وارد ناحیه بحرانی شود به شرط اینکه فرآیند دیگر نخواهد وارد ناحیه بحرانی شود.
7. مشکل بزرگ الگوریتم پترسون Busy Waiting است. البته بقیه الگوریتم ها نیز همین مشکل را دارند.
8. این الگوریتم کوتاهترین الگوریتم کاملا درست نرم افزاری است و هر الگوریتم کوتاهتر از آن حتما مشکل دارد.

تست: دوفرایند p0 وp1 به صورت همروند درحال اجراهستند راه حل زیر برای انحصار متقابل پیشنهاد شده است کدام گزینه صحیح است ؟

```
#int x=0;
#int f1=1;
#int f0=1;

///p0//////////////

while (true)
{
     f1 = 0;
     x = 1;
     while (x && !f0)
          C-S();
          f1 = 1;
          N-C-S();
}
```

```
///p1///////////////

while (true)
{
      f0 = 0;
      x = 0;
      while (!x && !f1)
            C-S();
            f0 = 1;
            N-C-S();
}
```

این الگوریتم همان الگوریتم پترسون است.

Interested (0) --> F0

Interested (1) -->F1

52:00

5. راه حل TSL (سخت افزاری)

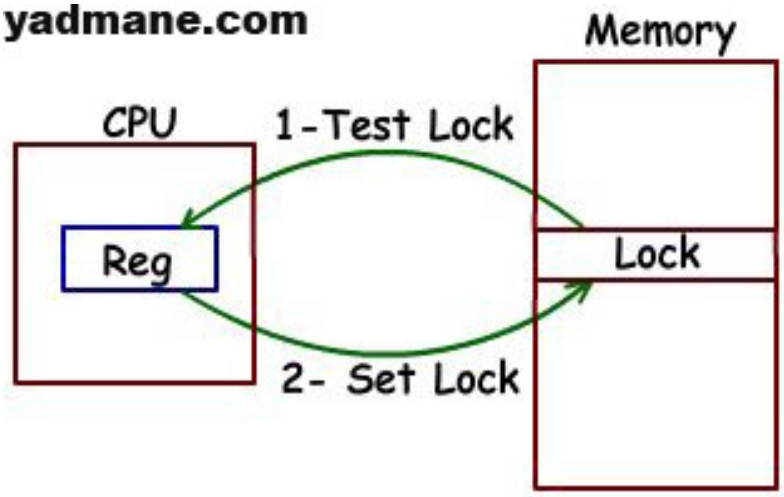
روش متغیر قفل را درنظر بگیرید. (Lock) مشکل اینجاست که اگر بین دوعمل Test lock وSet lock وقفه بیاید انحصار متقابل رعایت نمی شود.

```
While (trun) {
        while( lock = 1) ;          Test lock
        lock=1;                     Set lock
        C-S(1);
        Lock = 0;
        N-C-S(1);
}
```

راه حل :

1. وقفه ها را ازکار بیندازیم که دو مشکل دارد: 1- امنیت 2- درچند پردازنده ای بی فایده است.
2. به صورت سخت افزاری کاری کنیم که این دستورالعمل ها اتمیک شوند وتبدیل به یک دستورالعمل شوند. دستورالعمل جدید را test&set و test & set lock یا tsl می گویند.

Test lock + set lock = > test & set lock




```
enter_region:
        TSL REGISTER,LOCK       |copy LOCK to register and set LOCK to 1
        CMP REGISTER,#0         |was LOCK zero?
        JNE ENTER_REGION        |if it was non zero, LOCK was set, so loop
        RET                     |return to caller; critical region entered

leave_region:
       MOVE LOCK,#0             |store a 0 in LOCK
       RET                      |return to caller
```

امکان ندارد دو نفرهمزمان بتوانند وارد CS شوند چرا که همان لحظه که Lock راعوض می کنیم همان لحظه چک می کنیم.

1:17:00

وقتی که وقفه می آید مقدار رجیسترها درPCB فرایند ذخیره می شود به همین دلیل اگریک فرایند دیگر اجرا شود پس ازوقفه مجددا مقدار رجیسترهای فرایند ازPCB آن لود شده به همین دلیل برنامه ازهمان محل وبا همان داده قبلی به کارخود ادامه می دهد.

ایرادات این روش :

1. در این الگوریتم چون نوبت رعایت نمی شود (ورود به ناحیه بحرانی تصادفی است) و به همین دلیل ممکن است باعث قحطی زدگی شود.
2. Busy Waiting دارد.
3. وابسته به سخت افزارخاص است.

سوال 14 ص 272 - کامپیوتری را در نظر بگیرید که دستور العمل TEST AND SET LOCK را ندارد، اما دستورالعملی دارد که می تواند محتویات یک رجیستر پردازنده را با محتویات یک کلمه در حافظه در یک عمل واحد غیر قابل تقسیم مبادله نماید. آیا می توان از این دستورالعمل یک روال enter-region استفاده کرد؟

```
mov reg,1
xchg reg,lock
```

کافی است به جای دستور tsl دستور بالا را بنویسیم

چون عمل دیدن Lock وset وعوض کردن آن دریک دستورالعمل است دستورالعمل اتمیک است پس مشکلی پیش نخواهد آمد.

```
enter- region( ) ;
mov reg ,1;
exchg reg , lock;
cmp reg , 0;
Jnz  enter-region( );
ret;
```

1:29:45

1:33:00

درحالت کلی 3 تاراه حل سخت افزاری داریم :

- swap
- TSL
- ازکارانداختن وقفه ها

مشکل Busy waiting چیست؟

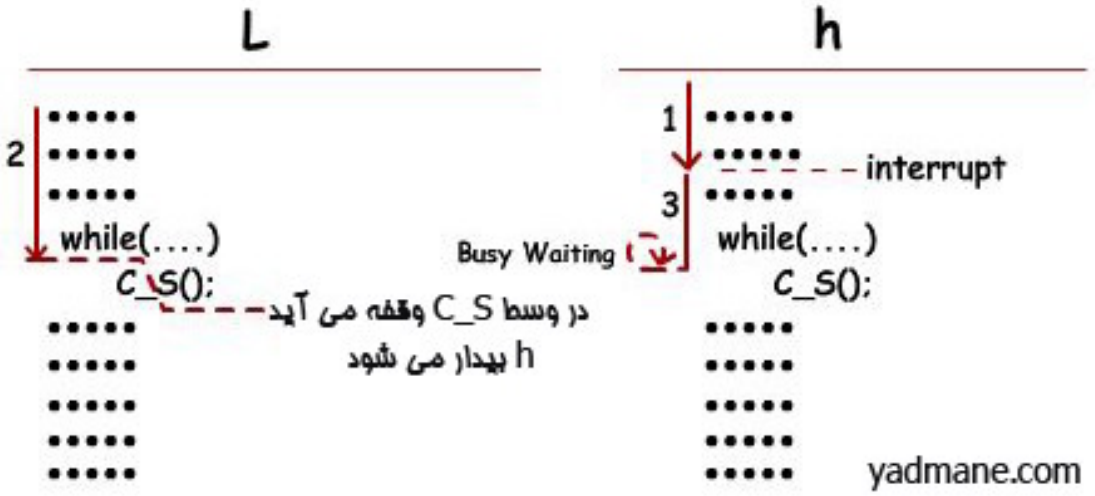
1. اتلاف CPU
2. مشکل اولویت معکوس (صفحه106 کتاب)

اولویت معکوس چیست؟

فرض کنید دو فرآیند H و L داریم

اولویت پایین <- L

اولویت بالا <-H



اگر فرآیند با اولویت پایین تر وارد ناحیه بحرانی شود و یک فرآیند با اولویت بالاتر بیاید و وقفه بیاید اتفاقی که می افتد این است که فرآیند H که CPU دارد و آن را رها نمی کند نمی تواند وارد ناحیه بحرانی شود چون فرآیند L در ناحیه بحرانی است. فرآیند L نیز CPU ندارد که از ناحیه بحرانی خارج شود. و این باعث بن بست و سیکل انتظار ابدی می شود.

1:44:00

خوابیدن و بیدار کردن :

چاره ای است درمقابل Busy waiting و مشکلات آن

مسئله تولید کننده و مصرف کننده :

ص 107 کتاب

یک بافرداریم به سایزN یکی تولید می کند و یکی مصرف می کند باید کاری کنیم تا این عمل ها همگام شوند. یعنی اگر ازبافری که پرنشده بخوانیم و یا در بافری که پر است مقدار بگذاریم.

```
#define N 100                                   /* number of slots in the buffer */
int count = 0;                                  /* number of items in the buffer */

void producer(void)
{
     int item;

     while (TRUE){                              /* repeat forever */
          item = produce_item();                /* generate next item */
          if (count == N) sleep();              /* if buffer is full, go to sleep */
          insert_item(item);                    /* put item in buffer */
          count = count + 1;                    /* increment count of items in buffer */
          if (count == 1) wakeup(consumer); /* was buffer empty? */
     }
}

void consumer(void)
{
     int item;

     while (TRUE){                              /* repeat forever */
          if (count == 0)(1) sleep();           /* if buffer is empty, got to sleep */
          item = remove_item();                 /* take item out of buffer */
          count = count  1;                     /* decrement count of items in buffer*/
          if (count ==N  1) wakeup(producer); /* was buffer full? */
          consume_item(item);                   /* print item */
     }
}
```

این مسئله غلط است زیرا اگر در (1) وقفه رخ دهد بن بست رخ می دهد (هر دو خواب ابدی می روند)

2:02:50

2:07:00

یک پیشنهاد: به جای (if count ==1) بگذاریم (if count >0)

باز هم بن بست ممکن است رخ دهد چرا که اگر در یک کوانتوم تولید کننده بافر را پر کند و به خواب برود آنگاه هر دو درخواب می مانند.

راه حل : ذخیره کردن سیگنال های Wakeup توسط هر دو فرایند است. هر فرایند یک بیت با مقدار اولیه صفر دارد هر گاه سیگنال های wakeup می آید اگر پروسس بیدار باشد این بیت 1 می شود واگر پروسس خواب باشد بیت صفر می ماند اما پروسس بیدار می شود. حال زمانی که پروسس می خواهد بخوابد این بیت را چک می کند اگر صفر باشد می خوابد و اگر 1 باشد آن راصفر می کند.

ص 108 کتاب پاراگراف آخر:

این روش برای 2 فرایند درست کارمی کند اما برای بیش از دو فرایند مشکل دارد.

A ---wakeup--->B----------C

فرض کنید بین A,B و B,C عامل های مشترک وجود دارد. ابتدا A یک Wakeup برای B می فرستد چون B بیداراست بیت آن 1 می شود. حال B می خواهد ازعامل مشترک بین B,C استفاده کند که پراست می خواهد بخوابد اما چون بیت آن 1 است نمی خوابد.

هرچند تننباوم عقیده دارد این مسئله با 2 فرایند درست کار می کند اما حتی با 2 فرایند نیز درست کار نمی کند.

فرض کنید ابتدا تولید کننده شروع به تولید می کند و اولین محصول را تولید می کند یک Wakeup نیز می فرستد بیت مصرف کننده 1 می شود بعد وقفه می آید فرایند دوم چون count صفر نیست ادامه می دهد بعد در دور بعدی count راچک می کند صفر است نمی خوابد اما محصولی وجود ندارد.

سمافورها :

ص 108 کتاب

سمافور به معنی راهنما می باشد

علت بن بست در مسئله تولید کننده و مصرف کننده این بود که سیگنال های wakeup هدر می رفتند. برای حل این مشکل ایده ای برای ذخیره کردن سیگنال های wakeup داده شد که موفق نبود. اما این ایده قابل تامل است و ما باید به گونه ای جلوی هدر رفتن سیگنال های wakeup را بگیریم.

دکسترا با استفاده از این ایده ساختاری به نام سمافور تعریف کرده که در آن wakeup هایی که قرار است هدر برود را می شمرد. پس ایده، شمارش سیگنال های wakeup ی است که قرار است هدر رود.

عملکرد سیگنال Wake up:

1. بیداراست ، سیگنال را ذخیره کن
2. خواب است ، بیدارکن

در سمافور ها معادل wakeup و sleep عمل های UP و DOWN است.

wakeup ~ UP

sleep ~ DOWN

تفاوت سمافور با سیگنال های sleep **و** wakeup :

1. ایده سمافور شمارش wakeup هاست.
2. طرف حساب ما فرآیند ها نیستند بلکه سمافور ها هستند. یعنی فرآیند ها را بیدار نمی کنیم بلکه یک نفر از آنهایی که روی سمافور خوابیده را بیدار می کنیم.

درصفحه 109 کتاب توضیح کامل سمافور ها داده شده است .

سمافور عبارت است از یک ساختار شامل دو فیلد (1- شمارنده 2- صف) و دو فانکشن اتمیک UP و DOWN

وقتی UP می کنیم اگر کسی بود بیدارش می کنیم وگرنه یکی به شمارنده اضافه می کنیم.

وقتی Down می کنیم اگر شمارنده بزرگتر از یک بود یکی کم می کنیم و اگر صفر بود یکی را می خوابانیم.

```
Struct semaphore{
        int count = initial value;
        int queue[N] = initial value;
}s;


void down(semaphore s){
if (s.count>0){
        s.count = s.count-1;
}else{
        place the process in s.queue;
        block the process;
}
void up(semaphore s){
if (s.queue is empty){
        s.count = s.count+1;
}else{
        remove a process from s.queue;
        place the removed process in ready queue; //wake up selected process
}
```

- مقدار شمارنده سمافور نمی تواند منفی باشد.(چون وقتی صفر باشد دیگر کم نمی کند می خواباند)
- درحین عملیات up وDown ابتدا به صف نگاه می کنیم که ببینیم کسی بیدار یا خواب است برای بیدارکردن برای سمافور ها یک صف داریم و اینکه کدام یک را از صف برداریم بستگی به شرایط مسئله دارد.
- وقتی که یک پروسس را بیدار می کنیم آن فرایند ازblocked به Ready می رود وفرایند جاری تا پایان کوانتوم خود به کارش ادامه می دهد.
- سمافورها اتمیک هستند.

اصطلاحات معادل :

Down	**~**	**wait**	**~**	**p**
UP	**~**	**signal**	**~**	**v**

برنامه های کاربر سه کار با سمافور می توانند انجام دهند :

1. تعریف (مقداراولیه)
2. Down
3. UP

چه چیزی سمافور را پشتیبانی می کند؟

اغلب سیستم عامل سمافور را سمافور را پشتیبانی می کند اما اگر سیستم عامل پشتیبانی نکند زبان های برنامه نویسی (کامپایلر) این کار را انجام می دهد. اما سیستم عامل ابزار های بهتری و راحت تری برای این کار دارد تا این دستورات را اتمیک انجام دهد.

23:30

تست: این برنامه چند غلط دارد:

```
1: Semaphor     S = 2
        ...
2:        Main {
          ...
3:      Down (S) ;
        ...
4:      UP (S);
        ...
5:      S.count + + ;
        ...
6:      if(s.count<0)UP(S);
}
```

نکاتی برای حل این تست؟

1. می توان به سمافور مقدار اولیه داد و مقدار اولیه سمافور الزاما صفر نیست.
2. تنها راه دسترسی به سمافور دوعمل اتمیک Down وup است .
3. توسط count نمی توانیم به مقدار آن دسترسی داشته باشیم. غیر از عمل UP و DOWN و مقداردهی اولیه هیچ راه دیگر برای دسترسی به سمافور نداریم. (نه خواندن و نه نوشتن)

می توانیم درسمافور ها اولویت بندی انجام دهیم :

وقتی می خواهیم مسئله رفتن به ناحیه بحرانی را با سمافور حل کنیم باید به 3 چیزتوجه کنیم .

1. چند تا سمافور داشته باشیم.
2. کجا Down و کجا UP کنیم.
3. مقداراولیه سمافور باید چند باشد.

در این مثال به چند سمافور نیاز داریم؟

چون یک صف داریم تنها یک سمافور کافی است. 35:50

✱ برای پیدا کردن مقدار اولیه Mutex باید حداقل وحداکثر راامتحان کنیم اگر Mutex = n باشد. N پروسس باهم می توانند وارد ناحیه بحرانی شوند واگرMutex = Q باشد هیچ فرایندی نمی تواند وارد ناحیه بحرانی شود. اگربخواهیم فقط یک پروسس وارد ناحیه بحرانی شود Mutex =1 مناسب است. 37:30

```
semaphore mutex = 1;
void p1()
{
     while (TRUE){
          down(&mutex);
            //اگر کسی در ناحیه بحرانی است بخواب
          c_s();
          up(&mutex);
            //اگرکسانی خواب هستند یکی را بیدار کن
          n_c_s();
     }
}
```

```
void p2()
{
     while (TRUE){
          down(&mutex);
          c_s();
          up(&mutex);
          n_c_s();
     }
}
```

تست: فرض کنید فرآیند روی یک سمافور می خوابند که ناحیه بحرانی فقط یک فرآیند را می تواند همزمان بپذیرد مقدار سمافور چه مقادیری می تواند باشد؟

پاسخ: فقط مقدار صفر و یک

به سمافری که فقط مقادیر صفر و یک می گیرد باینری سمافور می گویند.

در کتاب استالینگ سمافور می تواند منفی نیز باشد که به تعداد افراد خوابیده برای سمافور عدد منفی می شود. یعنی اگر سمافور3- باشد یعنی سه نفر روی سمافر خوابیده اند.

* بیدارکردن یک فرایند به این نیست که CPU رابه آن می دهیم بلکه آنرا ازblocked به ready می بریم وCPU براساس الگوریتم زمانبندی به آن داده خواهد شد.

* فرض می کنیم mutex count ابتدا 1 است و4 فرایند P7 وp4 وp3 وp2 را داریم. در ابتدا P2 وارد می شود وبا down ازmutex کم کرده و وارد CS می شود چون کسی داخل CS نیست.

فرض می کنیم درحین اجرای عملیات، وقفه ای می آید، P3 قصد ورود دارد، P3 با Down روبرو می شود وچون کسی داخل CS است به ناچار می خوابد در صف انتظار، درکوانتوم بعدی P2 به کارخود ادامه می دهد دوباره وقفه می آید وکوانتوم به P4 داده می شود که باز قصد ورود دارد اما چون CS اشغال است P4 نیز درصف می خوابد دوباره P2 کوانتوم می گیرد و درهنگام خروج با عملیات UP یکی ازفرایندها را از خواب بیدار می کند. این نکته قابل توجه است که مقدار count هیچ تغییری نکرده است . زیرا در صف انتظار فرایند منتظر ورود هستند. اما دراینجا لزوما کوانتوم بعدی به فرایندی که بیدارشده داده نمی شود واین بستگی به زمانبند دارد.

مثلا اگر درکوانتوم بعدی P7 بیدارشود با وجود آنکه کسی داخل CS نیست نمی تواند وارد CS شود بلکه چون سمافور صفر است باید درصف بخوابد وفرایندها به ترتیب نوبت واردCS می شوند. و اگر P7 وارد ناحیه بحرانی شود و وسط کارش کوانتومش تمام شود P4 که قبلا از mutex گذشته وارد ناحیه بحرانی می شود.

جلسه هفتم ۱۳۸۸/۶/۲ ساعت ۱۲:۰۰

حل مسئله تولیدکننده ــ مصرف کننده با استفاده از سمافور:

ص 111 کتاب

مسئله تولید کننده – مصرف کننده را با یک بافر 100 تایی درنظر می گیریم:

ما دو نگرانی داریم:

1. انحصار متقابل(هر دو همزمان سراغ بافر نروند): برای این منظور سمافور mutex با مقدار اولیه 1 ایجاد می کنیم.
2. برای همگام سازی دو سمافور full و empty ایجاد می کنیم.

```
#define N 100                                   /* number of slots in the buffer */
typedef int semaphore;                          /* semaphores are a special kind of int */
semaphore mutex = 1;                            /* controls access to critical region */
semaphore empty = N; //Y in sound               /* مقدار اولیه  برابر با تعداد خانه های خالی است */
semaphore full = 0;  //X in sound               /*  باشد1در ابتدای کار چون هیچ تولیدی نبوده پس این سمافر باید  */

void producer(void)
{
     int item;

     while (TRUE){                              /* TRUE is the constant 1 */
          item = produce_item();                /* generate something to put in buffer */
          down(&empty);                         /* اگر بافر خالی است- تعداد خانه های خالی بافر */
          down(&mutex);                         /* enter critical region */
          insert_item(item);                    /* put new item in buffer */
          up(&mutex);                           /* leave critical region */
          up(&full);                            /* اگر مصرف کننده خواب است بیدارش کن */
     }
}

void consumer(void)
{
     int item;

     while (TRUE){                              /* infinite loop */
          down(&full);                          /* اگر بافر خالی است بخواب */
          down(&mutex);                         /* enter critical region */
          item = remove_item();                 /* take item from buffer */
          up(&mutex);                           /* leave critical region */
          up(&empty);                           /* اگر تولید کننده خواب است بیدارش کن */
          consume_item(item);                   /* do something with the item */
     }
}
```

3. دربین عملیات (جایی که یکی به خانه های پر اضافه کرده ایم اما هنوز از خانه های خالی کم نکرده ایم) ممکن است full.count یکی بیشتر شود که با تکمیل عملیات مجموع خانه های پروخالی برابر n خواهد شد.
4. در روش سمافور دیگر if هایی که در روش های قبلی داشتیم را نداریم بلکه آن if ها در دل شمارنده سمافور تعبیه شده است.

39:40

مانیتور :

1. هر مسئله انحصار متقابل و همگام سازی را می توان با سمافور حل کرد اما کار با آن بسیار سخت است اما مانیتور یک ابزار سطح بالا برای اینگونه مسائل است.
2. مانیتور به معنی ناظر است و یک ابزار نظارتی است.
3. مانیتور یک ساختار برنامه نویسی است شامل متغیرها و زیربرنامه ها.

```
monitor example
 integer i;
 condition c;

 procedure producer (x);
 .
 .
 .
 end;

 procedure consumer (x);
 .
 .
 .
 end;
end monitor;
```

4. اداره سمافورها برعهده سیستم عامل است اما مانیتور توسط زبان برنامه نویسی(کامپایلر) پشتیبانی می شود.

50:50

5. سیستم عامل نمی تواند مانیتور را کنترل کند چرا که مانیتور ازSystem call استفاده نمی کند اما سمافور از System call استفاده می کرد (UP , Down)
6. سیستم عامل هیچ موقع درون برنامه شما را نمی بیند پس تا system call نباشد سیستم عامل وارد عمل نمی شود و هیچگاه کد ها را تفسیر نمی کند.

7. مانیتور یک ابزار راحت است (سطح بالا) اما سمافور سخت است و تشخیص محل UP وDown ها بسیار مشکل است.

مانیتور2 خاصیت دارد :

1. Encapsulation هیچ فرایندی ازخارج ازمانیتور نمی تواند به متغیرهای درون مانیتور دسترسی داشته باشد (چه خواندن وچه نوشتن) واگر بخواهیم به متغیری دسترسی داشته باشیم باید یک تابع درون مانیتور را فراخوانی کنیم .
2. انحصار متقابل : ذاتا انحصار متقابل دارد. یعنی اگر کسی وارد مانیتور شود کس دیگری نمی تواند واردآن شود. فقط یک نفر می تواند داخل مانیتور فعال باشد. (غیر فعال بودن یعنی نه Ready ونه Running است ، block است.) یعنی چند فرآیند می توانند داخل ناحیه بحرانی باشند اما تنها یکی از آنها می تواند فعال باشد.

1:08:00

- درهمگام سازی تاکنون یاد گرفته ایم که اگر منتظر چیزی هستیم بخوابیم. با زوج های Sleep / Wake up که زوج مناسبی نبود چون بن بست ایجاد می کردند زوج دوم با سمافور(Down / UP) که مشکلی نداشتند اما کار با آنها سخت بود . زوج سوم سیگنال های Wait / Signal است که برروی متغیرهای شرطی درون مانیتور اثرگذار است. چون این متغیر های شرطی داخل مونیتور تعریف شده است تنها راه دسترسی به آنها از داخل مانیتور است .
- از آنجایی که تنها فرآیند هایی می توانند یک فرآیند را که بر درون مانیتور خوابیده را بیدار کنند که داخل مانیتور باشند پس باید اجازه داد زمانی که فرآیندی داخل مانیتور خوابیده فرآیند های دیگر وارد مانیتور شوند در غیر این صورت فرآیند خوابیده نمی تواند خود را بیدار کند و باعث بن بست می شود.
- برای پرهیز فعالیت همزمان دو فرآیند درون مانیتور دو پیشنهاد داریم:
  1. فرآیند جدید را اجرا می کنیم و دیگری را معلق می کنیم (کسی را بیرون نمی کنیم)
  2. فرآیندی که سیگنال را داده سریع باید خارج شود (یعنی سیگنال آخرین حرکت درون مانیتور می تواند باشد)
  3. اجازه دهیم که اجرای سیگنال دهنده ادامه یابد و فرآیند منتظر هر وقت سیگنال دهنده خارج شد به اجرایش ادامه دهد.

1:30:00

- درمانیتور روش برنامه نویسی شما تغییری نمی کند یعنی اگر به کدهای صفحه 107و 117 نگاه کنیم شبیه هم هستند اما کد صفحه 112 که سختی است که از سمافوراستفاده کرده است.

```
monitor ProducerConsumer
     condition full, empty;
     integer count;

     procedure insert(item: integer);
     begin
          if count = N then wait(full);
          insert_item(item);
          count := count + 1;
          if count = 1 then signal(empty)
     end;

     function remove: integer;
     begin
          if count = 0 then wait(empty);
          remove = remove_item;
          count := count  1;
          if count = N  1 then signal(full)
     end;

     count := 0;
end monitor;

procedure producer;
begin
     while true do
     begin
          item = produce_item;
          ProducerConsumer.insert(item)
     end
end;

procedure consumer;
begin
     while true do
     begin
          item = ProducerConsumer.remove;
          consume_item(item)
          end
end;
```

در کد بالا اگر قسمت های داخل مانیتور را داخل کد اصلی بگذاریم همان برنامه نویسی قدیمی می شود.

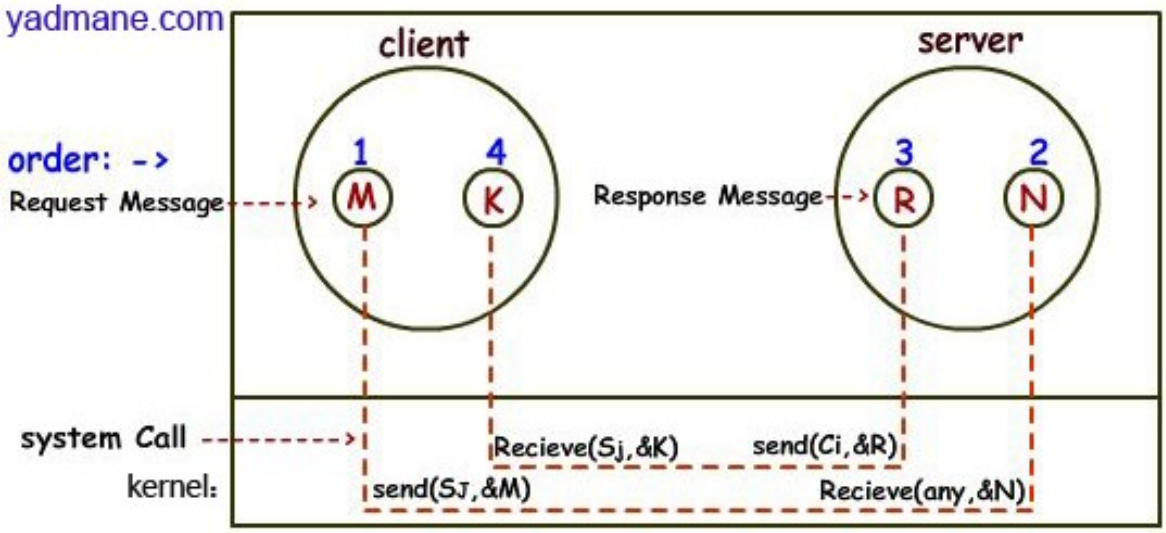
تفاوت down و up سمافور با wait و signal مانیتور چیست؟

1. سمافور شمارنده احتیاج دارد اما مانیتور به شمارنده احتیاج ندارد زیرا هیچ سیگنال Wake up ی هدر نمی رود و نیازی به شمارش wake up های هدر رفته نداریم.
2. اما در مانیتور صف را داریم.
3. مانیتور ساختاری است که همه زبان های برنامه نویسی آن را **Support** نمی کند. مثلا C++ را پشتیبانی نمی کند اما **Java** و C# آن را پشتیبانی می کند.

تبادل پیام :

ابزاری است که به دو منظور به کار می رود:

- تبادل داده
- همگام سازی



در تبادل پیام فرستنده پیام را send می کند اما در این روش بدین صورت نیست که هسته پیغام را به گیرنده به زور بدهد. هسته نمی تواند به دلخواه یک فضا را انتخاب کند و پیغام را آنجا قرار دهد زیرا ممکن است روی Stack یا داده های دیگر قراردهد خود گیرنده باید یک فضا رامشخص کند.

گیرنده D از کجا می داند که عمل receive را انجام دهد؟

فرایندها وقتی به هم پیام می دهند درواقع ازقبل برنامه ریزی شده اند یعنی Ps وPd در کد چنین اقتضایی دارند. یعنی برنامه نویسی که کد را نوشته کد را اینگونه نوشته است و قطعا در مقابل ارسال پیام دریافت کننده ای را هم از قبل مشخص کرده است.

بعضی مواقع ممکن است یک سرویس دهنده درخواست را از هرکسی قبول کند مثل فایل سرور که درخواست های همه را قبول می کند، اما پس از قبول کردن درخواست هنگام پاسخ ، پاسخ را باید به همان شخص فرستنده بفرستد.

Recive(any,&N)

اگر فرآیندی Receive کند اما طرف مقابل send نکرده باشد هسته با فرآیند receive کننده چه می کند؟

1) فرایند را بلوکه می کند.
2) فرایند Busy Waiting می کند.

دیدگاه اول، Blocking :

سیستم های تبادل پیام به 3 دسته تقسیم می شوند:

1) سیستم هایی که هسته ، گیرنده را بلوکه می کند. مزیت این روش سادگی است .

22:10

2) هسته گیرنده را بلوکه نمی کند . کاری که گیرنده انجام می دهد این است که در زمانی که باید منتظر پیام دهنده باشد کار های دیگری را انجام می دهد که ربطی به پیام ندارد. و باید مرتبا سرکشی کند که پیام آمده یا نه که polling چندان جالب نیست.
3) یک نخ دیگر بنویسیم که زمانی که از طرف فرستنده پیغام آمد سیستم عامل یک وقفه دهد و گیرنده برود و پیغامش را چک کند. این روش پیچیده است زیرا یا باید چند نخی داشته باشیم. ویا به صورت یک وقفه عمل کنیم. این روش در ازای از دست دادن سادگی Busy Waiting را نیز حل کند.

- به صورت پیش فرض وقتی نوع پیام را نمی گویند Blocking (حالت 1) درنظرگرفته می شود.

دیدگاه دوم ، Buffering :

دراین دیدگاه معکوس فکر می کنیم یعنی فرستنده فرستاده است اما گیرنده نمی خواهد دریافت کند راه حل ها به این ترتیب است:

1. فرستنده پیغام را می فرستد و هسته می داند که آن را باید به چه کسی بدهد، هرگاه گیرنده receive گفت سیستم عامل پیام را به آن می دهد. مشکل اینجاست که در پیام های متوالی لزوما آدرس نمی ماند وممکن است بعد از مدتی تغییر کند در حالی که فرستنده آن را به همان آدرس قبلی می فرستد و مشکل بزرگتر اینکه فرستنده پیامی را overwrite کند که هنوز receive نشده است. که برای جلوگیری از این مشکلات چاره ای نداریم جز اینکه زمانی که فرستنده پیام را فرستاد تا زمان receive آن را بخوابانیم.
2. راه دوم استفاده از بافر است (Mail box) که می تواند اختصاصی یا اشتراکی باشد البته داشتن یک بافر اشتراکی مناسب نیست زیرا سریع پرمی شود. اگرفرستنده ارسال کند وگیرنده نخواهد دریافت کند. فرستنده آن رادرMail box یا بافر گیرنده قرار می دهد. اما اگر گیرنده بافر راخالی نکند تا زمانی که بافر پرشود. درآن صورت فرستنده دیگر نمی تواند عمل send را انجام دهد . دراین صورت فرستنده مجبور است بخوابد.

اگر مشخص نکردند پیام از چه نوعی است بافر داریم.

- ممکن است در صندوق شما پیام باشد receive کنید و به شما پیام ندهند؟

بلی چون ممکن است پیام از فرستنده ای که ما می خواهیم نباشد.

- ممکن است دو فرآیند سر receive دچار بن بست ش.وند؟

بلی ممکن است فرآیند اول receive می کند و می خوابد حال فرآیند دوم به جای اینکه send کند receive می کند (که البته به دلیل ضعف برنامه نویسی این حالت ممکن است رخ دهد) که در این صورت هر دو فرآیند می خوابند.

- آیا ممکن است دو فرآیند هر دو send کنند و بن بست پیش آید؟

اگر send ها از نوع non-buffer blocking باشند

- ممکن است دو فرآیند send کنند بافر هم داشته باشند و بن بست پیش آید؟

بلی در صورتی که هر دو بافر پر باشند.

۴۷:۳۰

دیدگاه سوم ،قرار ملاقات (Rendezuous) :

ص۱۲۱ کتاب

این روش **بدون بافر** است و**blocking** دارد. هرکس زود تر برسد باید بخوابد. برای حالت های بدون بافر است زیرا اگربافر داشته باشیم دیگرنیاز نیست بخوابیم تا پیام طرف مقابل برسد.

حل مسئله تولیدکننده ــ مصرف کننده با استفاده از پیام :

```
#define N 100                                   /* number of slots in the buffer */

                                void producer(void)
                                     {
                                   int item;
          message m;                                /* message buffer */

                                  while (TRUE) {
   item = produce_item();   /* generate something to put in buffer */
       receive(consumer, &m);   /* wait for an empty to arrive */
       build_message(&m, item); /* construct a message to send */
          send(consumer, &m);      /* send item to consumer */
                                    }
                                    }

                                void consumer(void)
                                     {
                                  int item, i;
                                   message m;

for (i = 0; i < N; i++) send(producer, &m);   (1) /* send N empties */
                                  while (TRUE) {
    receive(producer, &m);            /* get message containing item */
     item = extract_item(&m);          /* extract item from message */
       send(producer, &m);               /* send back empty reply */
    consume_item(item);               /* do some1thing with the item */
                                    }
                                    }
```

این سومین راه حل درست مسئله تولید کننده ومصرف کننده است. (بااستفاده ازصندوق پستی وبلوکه کردن)

(1) درحل این مسئله ابتدا یکصد پاکت خالی می فرستیم که نشان دهنده ظرفیت بافراست .

تولید کننده : یک Item تولید می کند وآنرا درپاکت خالی قرارمی دهد.

مصرف کننده : پاکت را receive می کند محتوی آن را برمی دارد وپاکت خالی را پس می فرستد.

ضعف این روش این است که ابتدا باید 100 پاکت خالی بفرستیم وضعف دوم این است که بافر نداریم و بافرها همان پیام ها ست.

مزیت این روش این است که درسیستم های توزیع شده (تحت شبکه) ازهیچکدام ازابزارها مثل سمافور، مانیتور، TSL و.... نمی توانیم استفاده کنیم . وفقط ازروش پیام می توانیم استفاده کنیم.

1:16:00

مسائل کلاسیک IPC از کتاب خوانده شود

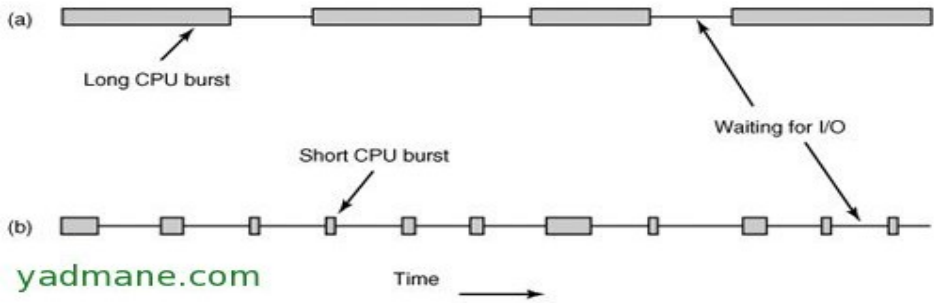
مسئله ص 123 دو بار در کنکور آمده

غذا خوردن فیلسوف ها ، تولید کننده و مصرف کننده خیلی مهم است.

1:20:00

زمانبندی :

CPU Burst : آن قسمت هایی ازفرایند که CPU می گیرند.



در شکل بالا : A که CPU Burst Time هایش بزرگ است را CPU Limited گویند و B که CPU Burst Time هایش کوچک است را I/O limited گویند

تست : اگرهیچ فرایندی برای اجرانباشد CPU چه کارمی کند؟

1. هیچ کارنمی کند
2. Interrupt Rutin انجام می شود
3. Busy Waiting
4. یک فرآیند background اجرا می شود.

نکته : CPU هیچ گاه نمی تواند بیکارباشد.

- **schedullar چه زمان هایی اجرا می شود؟**

1. زمانی که یک فرآیند exit می کند.
2. هنگامی که فرآیندی block می شود.

3. در برخی سیتم عامل ها در هنگام interrupt timer نیز schedullar را اجرا می کنند. مثل سیستم عامل های time sharing
4. وقتی I/O Interrupt رخ می دهد. (چون ممکن است فرآیند اولویت بالایی منتظر آن بوده است)
5. زمانی که فرآیند جدیدی می آید. (چون ممکن است اولویت آن بالا باشد.)

سه مورد آخر در همه سیستم عامل ها عمومیت ندارند.

گروه بندی الگوریتم های زمانبندی:

1. دسته ای
2. تعاملی (محاوره ای)
3. بلادرنگ

معیارهای زمانبندی :

یک سری معیارها عمومیت دارد که همه سیستم های زمانبندی باید داشته باشند . اما یک سری معیارها برای سیستم های خاص مهم است.

معیارهای عمومی :

1. انصاف (عدالت) : البته عدالت به این معنی نیست که همه باهم برابرهستند بلکه هرکس باید سهم عادلانه ای از اشتراک را بگیرد.
2. خط مشی : اعمال سیاست شرایط خاص را درنظر بگیرد.
3. توازن : از همه منابع بهره وری مطلوب و متوازن داشته باشد.

معیار های کمی :

1. **توان عملیاتی (گذردهی) :** تعداد کار های انجام شده و خارج شده در واحد زمان یا true put
2. **زمان برگشت (زمان کامل) :** از لحظه ای که کار داده می شود تا زمانی که کار خارج می شود

turn around time

3. **بهره وری یا راندمان CPU :** درصد استفاده مفید از CPU (زمان مفید به زمان کل)

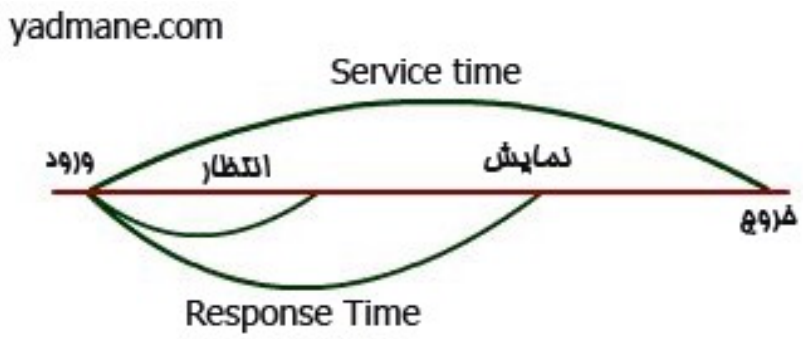
زمان غیر مفید زمان بیکاری و زمان های switch می باشند.

سیستم های تعاملی :

آنچه در این سیستم ها مهم است زمان پاسخ یا response time می باشد.

زمان سرویس : از لحظه ورود کار تا پایان کار

زمان پاسخ : از لحظه ورود کار تا لحظه نمایش یا چاپ .



زمان انتظار : زمانسرویس منهای زمان پاسخ

نکته:

- دربیشتر سئوالات کنکور ازلحظه نمایش تا خروج را حذف می کنیم.
- در اغلب سوالات کنکور زمان response time با turn around time برابر است چون در رابطه با زمان نمایش و چاپ صحبتی نمی شود.

جلسه هشتم دوشنبه ۹/۶/۱۳۸۸ ساعت ۱۲:۰۰

الگوریتم های سیستم های دسته ای:

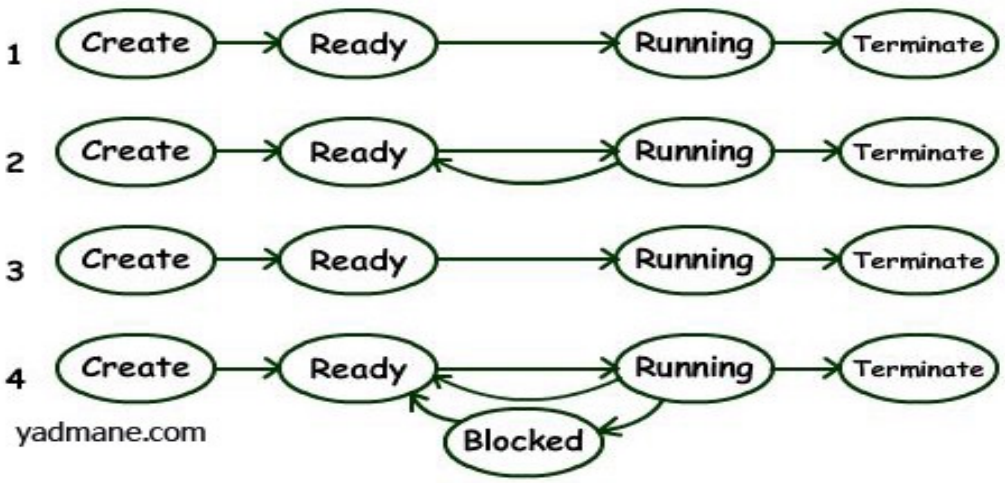
زمانبندی FCFS یا First come First set :

فرایندها دریک صف هستند وبه ترتیب ورود سرویس می گیرند.

خصوصیات این زمانبندی :

1. عدالت
2. ساده وقدیمی
3. انحصاری non preemptive یعنی فرآیند داوطلبانه می تواند CPU را رها کند در غیر این صورت تا زمان پایان کاریا بلوکه شدن دست فرایند می ماند فرایندی که بلوکه شد موقعی که آزاد شد برای گرفتن CPU باید به ته صف برود.
4. بدون قحطی

تست: کدام یک از شکل های زیر FCFS است؟



پاسخ: 3 صحیح است چرا که در 4 و 2 برگشت از اجرا به آماده به معنی غیر انحصاری بودن است

21:00

- درست نیست که بگوییم هرکسی که اول CPU را می گیرد اول تمام می شود چون ممکن است بلوکه شود وشاید خیلی درست نباشد که به آن FIFO بگوییم و به صورت کامل مثل سیستم های قدیمی نیست.
- انتظار بدون حد بالای مشخص را قحطی می گویند.

33:00 •

الگوریتم SJF یا Shortest Job First :

- یک الگوریتم انحصاری است که اولویت رابه کارهای کوتاهتر می دهد. (از لحاظ زمان نه حجم برنامه)
- مشکل : ازکجا بفهمیم که یک کارکوتاه تراست (ازنظرزمانی)
- هیچگاه سیستم عامل به داخل برنامه ها نمی رود ونگاه نمی کند ـ وبه آن مانندیک black box نگاه می کند.
- البته می توانیم براساس رفتار گذشته زمان باقیمانده را تخمین بزنیم (مثلا با الگوریتم Aging)

مشکلات این الگوریتم :

1. نیاز به تخمین دارد و قابل پیاده سازی دقیق نیست.
2. قحطی دارد زیرا هیچ حد بالایی برای انتظار ندارد. زیرا تا زمانی که کارهای کوچک وجود دارد وکارهای کوچک می آید نوبت به کارهای بزرگتر نمی رسد.

مزیت این الگوریتم :

- میانگین زمان انتظار کاهش می یابد.

حداکثر زمان انتظار برای کار های بزرگ افزایش می یابد.

A D C B

0----------8---------9----------11----------14

کار	زمان ورود	زمان سرویس	زمان برگشت turn around time	زمان انتظار waiting time (محاسبه به دو روش)	
A	0	8	8-0=8	0-0=0	8-8=0
B	1	3	14-1=13	11-1=10	13-3=10
C	2	2	11-2=9	9-2=7	9-2=7
D	2	1	9-2=7	8-2=6	7-1=6

TT زمان برگشت = زمان خروج – زمان ورود

WT زمان انتظار = TT زمان برگشت – ST زمان سرویس

AWT میانگین زمان انتظار = ATT میانگین زمان برگشت – AST میانگین زمان سرویس

Average=9.25 AWT=9.25-3.5=5.75

حل داده های بالا به روش fifo

A D C B

0----------8---------9----------11----------14

کار	زمان ورود	زمان سرویس	زمان برگشت turn around time	زمان انتظار waiting time (محاسبه به دو روش)	
A	0	8	8-0=8	0-0=0	8-8=0
B	1	3	11-1=10		
C	2	2	13-2=11		
D	2	1	14-2=12		

ART= (Average Response Time) = **((8-0)+(11-1)+(13-2)+(14-3))/4=10.25**

- معمولاAWT وATT (ART) درSJF کوچکتر ازFCFS می شود. در حالت خاص هر دو حالت برابر می شوند آن هم حالتی که فرآیند ها به ترتیب از کوچک به بزرگ می ایند .
- الگوریتم هایی که زیاد درکنکورمی آیند : RR, SJF, FCFS(FIFO) ,MLFQ(FB) , SRT

- آیا همیشه میانگین SJF ازFIFO کمتراست ؟

a <= b <= c <= d

کار	زمان ورود	زمان سرویس	زمان برگشت turn around time	زمان انتظار waiting time
j1	0	b	a+b-0=a+b	a
j2	0	c	a+b+c	a+b
j3	0	a	a	0
j4	0	d	a+b+c+d	a+b+c

j3 j1 j2 j4

0----------a----------b----------c----------d

ATT = ART = (4a + 3b + 2c + d)/4

AWT = (3a + 2b + c)/4

نتیجه می گیریم AWT و ATT مینیمم هستند زیرا صورت کسر مینیم است زیرا a که ازهمه کوچکتراست ضریب بزرگتری دارد وd که ازهمه بزرگتر است ضریب کوچکتری دارد. با هر ترکیب دیگری که اجرا کنیم صورت از این کمتر نخواهد شد.

در این مثال چون زمان ورود همه فرآیند ها صفر است می توانیم کوچکتر را انتخاب کنیم پس SJF زمان میانگین زمان انتظار فرایند ها را مینیمم می کند اما در حالت قبلی که لحظه ورود برابر نبود فقط میانگین انتظار را کاهش می داد و لزوما مینیمم نخواهد بود.

تست: کدام گزینه د رست است ؟

1. SJF همیشه زمان انتظار همه فرایند ها را min می کند.
2. SJF همیشه زمان انتظارهمه فرایند ها را کاهش می دهد.
3. SJF میانگین زمان انتظار همه فرایندها را min می کند.
4. SJF وقتی همه کارها ازقبل وارد شده اند زمان انتظار کارها را min می کند. ✓

Min ایده آل درحالت غیر انحصاری می باشد یعنی اگر یک کار بزرگ درحال اجراست و در میانه کار کارکوچکی آمد به کار کوتاه پاسخ می دهیم.

حل مسئله قبل با روش غیرانحصاری SRT

Shortest Remaining Time:

- این الگوریتم SJF غیرانحصاری است. بنابراین بررسی های جدید در لحظه ورود فرایند جدید است
- قحطی دارد.
- شدیدترین الگوریتم علاقمند به کارهای کوچک است.
- میانگین زمان انتظار و میانگین زمان پاسخ را حداقل می دهد.(بدون در نظر گرفتن سربار سوییچ)

- اگرسربارSwitch را درنظربگیریم همیشه میانگین زمان پاسخ وانتظار کاهش نمی یابد البته این اتفاق به ندرت می افتد.

A B D B C A

0---------1---------2---------3----------5----------7----------14

کار	زمان ورود	زمان سرویس
A	0	8->7
B	1	3->2
C	2	2
D	2	2

(14 + 4 + 5 + 1)/4=7.5

1:09:50

الگوریتم HRN highest Response Ratio Next **(بالاترین نسبت پاسخ)**

- این الگوریتم انحصاری است.
- ازبین کارها آن را انتخاب می کند که نسبت پاسخ بالاتری دارد.

W : زمان انتظار تاکنون

S : زمان سرویس دهی

نسبت پاسخ = $(W+S)/S = W/S + 1$

برای حل مسئله می توان از (1+) صرفنظر کرد زیرا در پاسخ تاثیری ندارد. اما اگر نسبت پاسخ را بخواهد حتما بایستی به صورت کامل حساب کنید.

معایب :

1. نیاز به تخمین
2. کمی سربار محاسباتی چرا که در هر مرحله باید W/S را حساب کنیم.

A B D C

0----------11----------15----------17----------20

کار	زمان ورود	زمان سرویس	زمان انتظار waiting time
A	0	11	0-0 =(11-0)-11=0
B	6	4	11-6 =(15-6)-4=5
C	8	3	12-8 =(20-8)-3=9
D	10	2	15-10 =(17-10)-2=5

در لحظه 11 باید W/S را برای همه حساب کنیم:

W/S(B) = (11-6)/4 = 1.25

W/S(C) = (11-8)/3 = 1

W/S(D) = (11-10)/2 = 0.5

در لحظه 15 : (با وجود اینکه فرآیند جدیدی نیامده مجددا حساب می کنیم)

W/S(C) = (15-8)/3 = 2.3

W/S(D) = (15-10)/2 = 2.5 --> برخلاف محاسبه قبلی بیشتر شد

AWT=19/4=4.75

- اگر W ها یکسان باشد آن که S کوچکتری دارد W/S آن بزرگتر است ، اولویت بیشتری دارد. درنتیجه رفتار SJF گونه دارد.

4. اگر S ها یکسان باشند : آن که W بزرگتری دارد، W/S آن بزرگتر است ، اولویت بیشتری دارد درنتیجه رفتارFCFS گونه دارد.
5. این الگوریتم ترکیبی ازدوالگوریتم FCFS وSJF است. اگر متناسب با اندازه ات wait کنی نوبتت می شود. به کارهای کوچک به صورت متعادل توجه می کند.
6. این الگوریتم ازSJF وFCFS عادلانه تراست.
7. شرایط دو فرایند با تناسب W/S بررسی شود.

- در الگوریتم هایی که تاکنون گفته شده فقط FCFS قابل اجراست زیرا بقیه نیاز به تخمین دارند.

- جایگاه ART و AWT نسبت به همدیگر

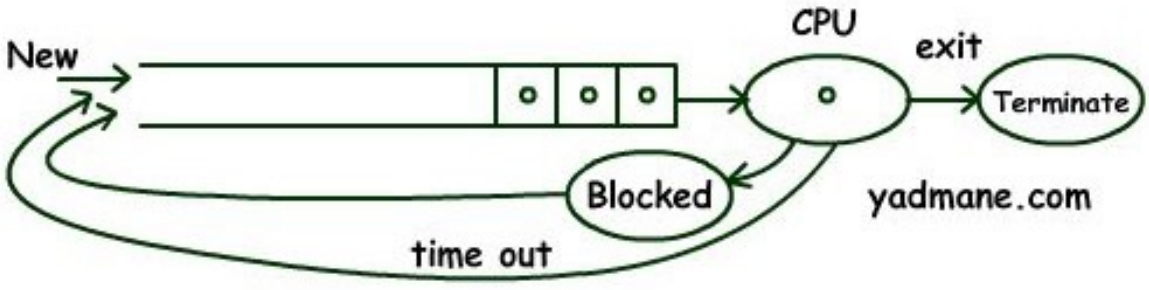
ART , AWT : SRT <= SJF <= HRN <= FCFS

ابتدای صدای 2

زمانبندی سیستم های تعاملی

زمانبندی نوبت گردشی Round Robin (خیلی مهم)

Round robin: preemption at time slice



برای فرایند درحال اجرا سه اتفاق ممکن است بیفتد:

1. Time out
2. Terminated
3. Blocked

خصوصیات این الگوریتم :

1. ساده
2. قدیمی
3. رایج
4. عادلانه (fairness)
5. غیر انحصاری (پس گرفتن درپایان کوانتوم)
6. قحطی ندارد.

نکات:

- الگوریتم RR و FCFS دقیقا قابل پیاده سازی هستند.
- الگوریتم RR میانگین زمان پاسخ راخراب می کند.
- هدف : زمان پاسخ کار های کوچک از یک حدی بیشتر نشود. (در حد مطلوب باشد)

اندازه کوانتوم چقدر باشد مناسب است ؟

برای اینکه ببینیم سایز کوانتوم چقدرباشد مناسب است حالت های اغراق آمیز وغیرعادی خیلی بزرگ وخیلی کوچک رادرنظر می گیریم.

- اگر کوانتوم خیلی بزرگ باشد می شود همان FCFS . یعنی می خواهیم زمان پاسخ کارهای کوچک از یک حدی بیشتر نشود که اگر کوانتوم خیلی بزرگ باشد به این هدف نمی رسیم و در واقع به هدف گارانتی کردن اجرای کارهای کوچک نخواهیم رسید. و راندمان کاهش پیدامی کند.
- اندازه کوانتوم نباید خیلی کوچک باشد زیرا به دلیل سربار سوییچ راندمان کاهش پیدا می کند.
- اگر کوانتوم خیلی کوچک باشد (کوچکتر از کارهای Uniqe) در این صورت CPU کارایی ندارد زیرا CPU فقط بین فرآیند ها سوئیچ می کند و کاری انجام نمی دهد و در واقع کوانتوم را هدر داده ایم .
- درعمل اندازه کوانتوم را بین 20 تا 100 میلی ثانیه می گیرند و گاهی اندازه کوانتوم متغیر است.

21:40

تست: در یک سیستم RR اندازه کوانتوم : $q = 20ms$ و $s = 10ms$ راندمان CPU چقدراست؟

1. 2/3
2. کمتراز2/3✓
3. بیش از2/3
4. به سمت صفرمیل می کند

پاسخ:

گزینه 2 صحیح است زیرا در حالت ایده آل این فرض را داریم که همه فرایند ها کوانتوم خود را کامل استفاده کند و هدر ندهد اما درواقع چنین اتفاقی نمی افتد پس حداکثر راندمان 2/3 است .

نکته : اگر یک فرایند درنیمه کوانتوم خود بلوکه شود بلافاصله سوئیچ می کنیم به فرایند دیگر و کوانتوم هدر نمی رود.

تست: در یک سیستم RR اندازه کوانتوم q و زمان سوئیچ s و میانگین زمان اجرا تا درخواست I/O برابر R است .

اگر $q = s$ و r بزرگتر از q باشد با صرفنظر از زمان سوئیچ به خاطر وقفه I/O راندمان چیست ؟

1. کمتر از 50%
2. 50% (پاسخ مورد نظر طراح)
3. به سمت صفرمیل می کند
4. به سمت 100% میل می کند

پاسخ: تست غلط است چون اگر زمان سوئیچ به خاطر وقفه را درنظر نگیریم بیش از 50% خواهد شد . چون اگر یک فرایند از کوانتوم خود کامل استفاده نکند فرایند بعدی یک کوانتوم می خواهد آنگاه بیش از یک کوانتوم اجرا داریم. اگر CPU burst time مضرب صحیحی از q باشد ب درست خواهد بود.

46:30

- اگر n فرایند در RR داشته باشیم و $q = s$ حداکثر زمان انتظار برای دریافت اولین کوانتوم چقدراست؟

$(n-1)q$

- حداکثر زمان پاسخ فرایند کوچکی که $q = s$ است چقدراست ؟

nq

برای انتظار $(n-1)q$

یک کوانتوم به خاطر خودش q

- حداکثر زمان پاسخ فرایندی که زمان سرویس آن $2q$ باشد چقدراست ؟

زمان پاسخ:

$2nq$

البته اگر زمان I/O را صرفنظر نکنیم باید با زمان سوئیچ جمع شود

همچنین زمان انتظار:

$2(n-1)q$

نکته: در RR وقتی یک فرآیند دو کوانتوم پشت سر هم می گیرد دیگر زمان سوییچ بین دو کوانتوم را در نظر نمی گیریم.

1:00:30

سوالات RR:

- سیستم Time Sharing و RR است . $q = 1$ و $s = 0$ ، سومین کوانتوم راچه کسی می گیرد ($T = 2$).

توجه: در این تست فرایندی که تازه وارد می شود نسبت به فرایندی که تازه کوانتومش به پایان رسیده ، اولویت دارد.

1:02:20

p1 p2 p1 p3 p2 ^ p4 p1 ^ p3 p4 ^ p3 ^

0--------1--------2--------3--------4--------5--------6--------7--------8--------9--------10--

کار	زمان ورود	زمان سرویس
p1	0	3->2->1
p2	0	2->1
p3	2	3->2->1
p4	3	2

توجه: در بعضی تست ها ممکن است بر خلاف RR عادی گفته شود فرآیند تازه وارد اولین کوانتوم بعدی، بعد از ورود را اخذ می کند.

- سیستم Time Sharing و RR است . $q = 1ms$ و $s = 0$

کار	زمان ورود	زمان سرویس به دقیقه
A	0	6->5->3
B	1	2->0
C	6	4->3->2
D	8	2->1

اگر بخواهیم کوانتوم کوانتوم حل کنیم چون کوانتوم 1ms است حل نشدنی است . به روش CPU sharing حل می کنیم یعنی تقسیم CPU به مقدار مساوی بین فرایند ها.

فقط به لحظات ورود وخروج فکرمی کنیم .

A A,B ^ A A,C A,C,D ^ C,D ^ C ^

0--------1------------5--------6----------8-----------11----------13--------14-

1:28:10

الگوریتم های اولویت (معمولا در کنکور نمی آید چون باید در صورت مسئله الگوریتم توضیح داده شود)

سئوال هایی که پیش می آید:

- اولویت ها راچه کسی تعیین می کند؟

policy یا خط مشی ازبیرون توسط سوپر وایزر به ما دیکته می شود.

- چگونه به هدفمان می رسیم ؟

منابع سیستم را بیشتر به کسی می دهیم که اولویت آن بالاتر است . روشی که در پیش می گیریم تا اولویت ها محقق شود مکانیزم نام دارد.

انواع مکانیزم ها :

(صفحه 144 کتاب)

1. کسی که اولویت بیشتری دارد را انحصاری انجام می دهیم تا تمام شود.

   این کار باعث قحطی برای اولویت های کمتری می شود.

   مشکل اینکه یک اولویت بالا می آید اما چون انحصاری است باید منتظر بماند.

2. اگر فرایند اولویت بیشتر ازCPU خواست به آن می دهیم (غیرانحصاری)

   برای اولویت کمتر قحطی دارد.

3. وقتی اولویت بالا می آید تا پایان کوانتوم باید صبر کند،

   این مکانیسم هم قحطی دارد.

4. برای اینکه اولویت پایین ها دچار قحطی نشوند:

اولویت پایین ها رابا گذشت زمان بالا ببریم .

اولویت بالاها را پایین بیاوریم .

اشکال این روش این است که اولویت ها به طور اغراق آمیزی به هم می ریزد.

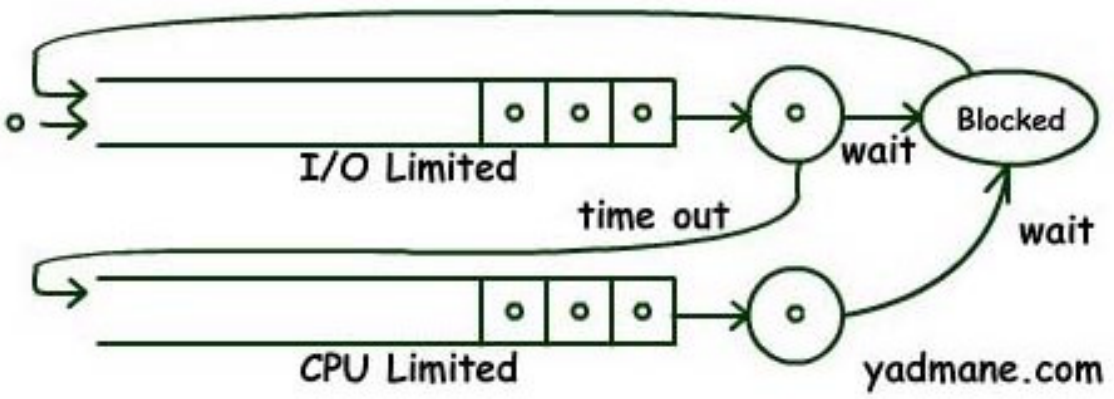
5. به هرکس متناسب با اولویت CPU می دهیم .

سامان 1:38:00

6. به I/O Limited ها اولویت می دهیم چون اولا کم CPU می خواهد.

ثانیا اینها خودشان زمان زیادی رابلوکه هستند.

ازکجا بفهمیم I/O است یا CPU ؟ ابتدا خوشبینانه آنرا I/O درنظر می گیریم . اگر کوانتوم را نیمه مصرف کند پس حدس ما می تواند درست باشد اما اگر کوانتوم راکامل مصرف کند درکوانتوم بعدی آنرا CPU درنظر می گیریم .



7. اگر در کوانتوم قبلی کسرF را مصرف کردید . اولویت جدید آن یک F ام می شود.

این روش فازی است روش قبلی صفر و یکی بود.

8. MLQ (شکل صفحه 146) Multy level queue

چند تا صف 0و 1و 2و .... داریم .اولویت صف ها متفاوت است .

ابتدا صف های با اولویت بالاتر را انجام می دهیم تا به اتمام برسد و اگر وسط یک صف بودیم و فرآیندی وارد صف اولویت بالاتر شد باید آن اولویت بالاترانجام شود.

در داخل هر صف مکانیسم های مختلفی می توانیم داشته باشیم.

این روش قحطی دارد.

جلسه نهم ۱۳۸۸/۶/۱۶ ساعت ۱۲:۰۰

19:00

صف های چند گانه :

ممکن است در کتاب های مختلف به یکی از این نام ها نامیده شده باشد:

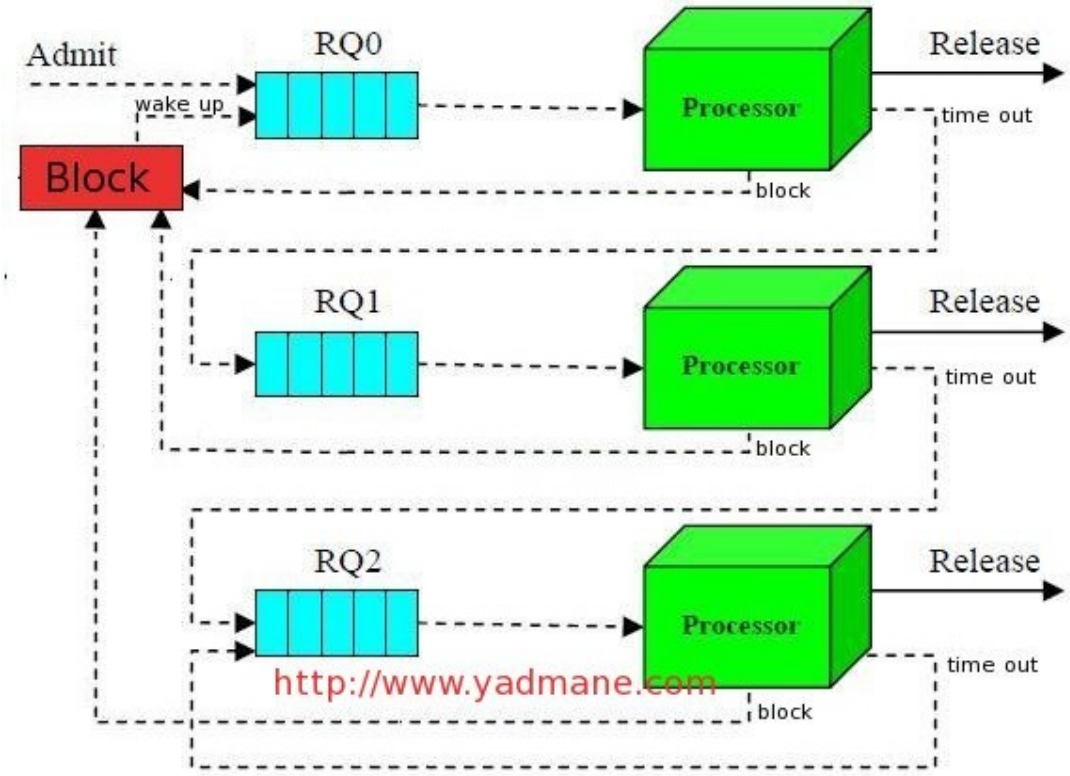
Multiple Queues

MFQ (MLFQ) Multilevel Feedback Queues صف های چند سطحی چند گانه

FB (Feed back) بازخورد

همانطور که می بینید چند صف اولویت دارد که به ترتیب از بالا به پایین اولویت صف ها کم می شود.

کلیه کارهای تازه وارد درانتهای صف اول قرار می گیرند.



اهداف الگوریتم :

1. سربار سوئیچ کم شود. (با بزرگ کردن کوانتوم فرآیند هایی که از کوانتوم های قبلی خود کامل استفاده کرده اند)
2. زمان پاسخ کارهای کوچک مطلوب باشد. (به کار های کوچک اهمیت بیشتری می دهد)

اگر در صف دوم باشیم و کاری وارد صف اول شود باید به صف اول برگردیم.

صف های پایین تر را دیر تر به سراغشان می رویم اما کوانتوم های بزرگتری به آنها می دهیم.

اگر یک JOB صد کوانتومی داشته باشیم درسیستم RR معمولی باید 100 بار سوئیچ کنیم اما اگر از صف های چند گانه استفاده کنیم به شکل زیر خواهد بود.

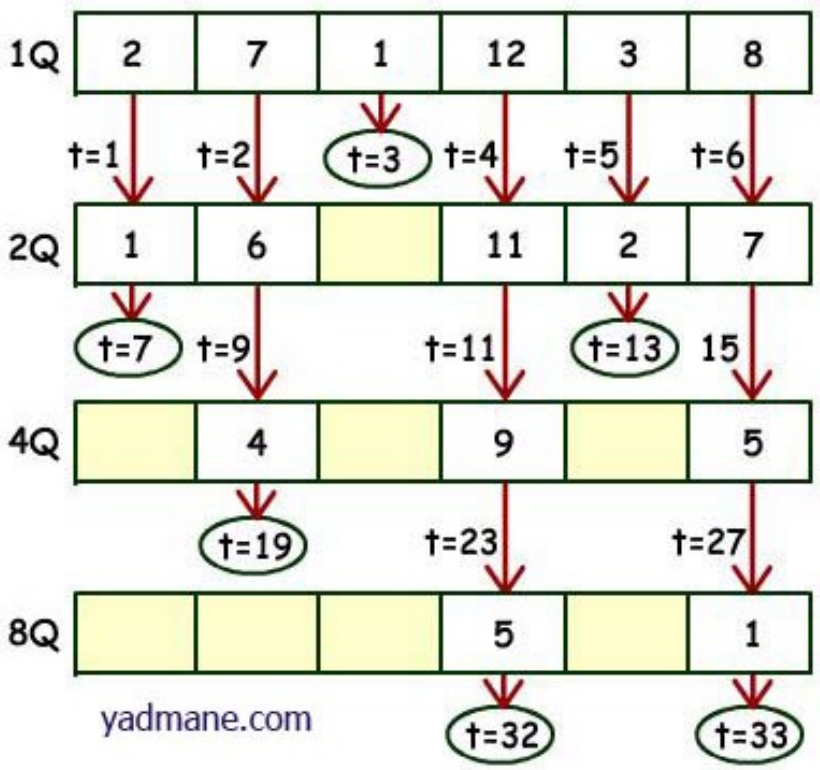
صف	مجموع کوانتوم تا کنون
1q	1
2q	3
4q	7
8q	15
16q	31
32q	63
64q	100

تعداد کوانتوم ها به صورت لگاریتمی کاهش پیدا می کند. (تعداد سوئیچ ها سقف لگاریتم تعداد کوانتوم مورد نیاز می شود)

- هدف دیگری که این الگوریتم دارد این است که به کارهای I/O Limited اولویت می دهد.
- ایراد : کارهای بزرگ دچار قحطی می شوند.
- این الگوریتم به کارهای کوچک اولویت می دهد اما نیاز به تخمین ندارد. ازروی Feed back اندازه کار را متوجه می شود.

37:50

مثال: 6 کار داریم که همگی در زمان صفر وارد شده اند وبه ترتیب 2 ، 7 ، 1 ،12 ، 3 و8 واحد زمانی پردازش احتیاج دارند. با صف های چندگانه بازخورد که صف اول، یک واحد زمانی ، صف دوم و بعدی هر کدام دو برابر صف قبلی CPU تخصیص داده می شود. میانگین زمان پاسخ چند است؟



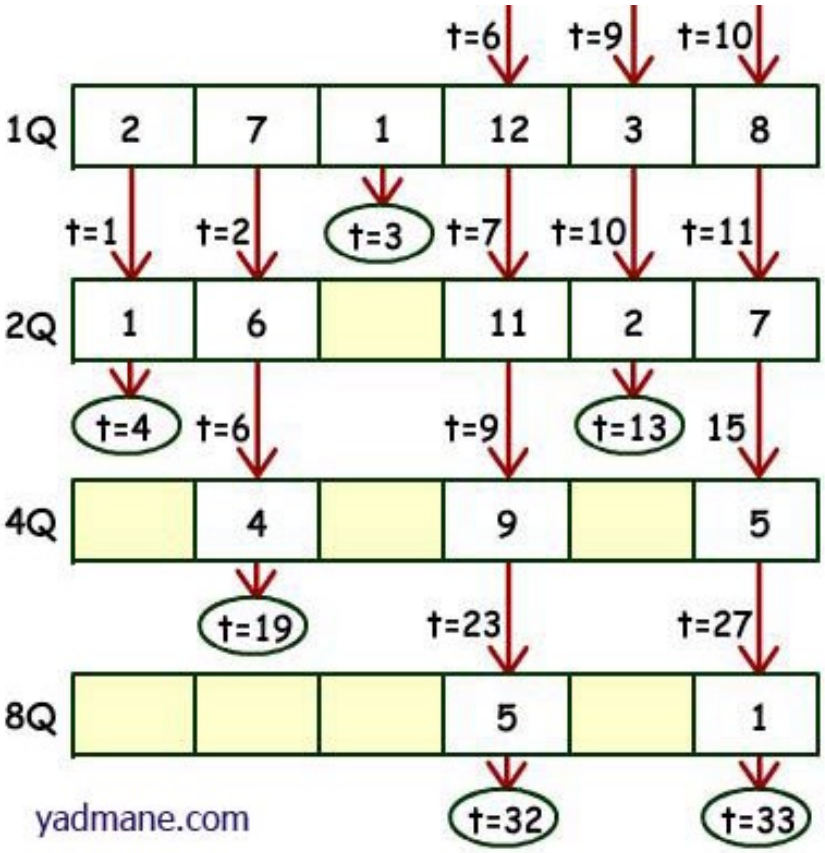
زمان پاسخ = T خروج – T ورود

ART = 3 + 7 + 13 + 19 + 32 + 33.6 = 17.83

48:30

50:00

ـ مثال قبل را با زمان های ورود متفاوت حل می کنیم . زمان های ورود به ترتیب 0 و 0 و 0 و 6 و 9 و 10 می باشد.



زمان پاسخ = T خروج − T ورود

(3 + 4 + (13-9) + 19 + (32-6) + (33-10)) / 6 = 79 / 6 = 13.1

1:18:20

این الگوریتم یکی از الگوریتم هایی است که درعمل از آن استفاده می شود و سیستم عامل های مدرن هم مشتقاتی از این الگوریتم را به کار می گیرند.

این الگوریتم مشتقاتی دارد و ممکن است درسئوال کنکور بگویند کوانتوم صف اول 8ms و صف دوم 16ms ودرصف سوم FCFS باشد هیچ الزامی نیست که کوانتوم ها دو برابر شوند و این صف ها ادامه داشته باشند.

تنوع های مختلفی از FB ها داریم :

5. 2^i–FB

2^i کوانتوم فیدبک: برای حالت هایی که سربار کوانتوم بالاست.

6. i_FB

i کوانتوم feed back می باشد.

7. 1_FB

1 کوانتوم feed back می باشد.

اگر کوانتوم FB یک باشد RR نیست . زیرا باعث قحطی می شود. در صورتی که RR قحطی ندارد.

مشتقاتی از این الگوریتم ساخته اند که قحطی ندارد برای این منظور می توان فرایند ها را پس ازمدتی به صف های با اولویت بالاتر بازگرداند.

1:28:50

الگوریتم SPN

shortest process next

این همان SJF است اما چون درسیستم های غیر تعاملی گفته شده بود در اینجا دوباره آورده شده است. اما در SJF تخمین احتیاج داشتیم که درآن موقع تخمین بحث نشده بود.

درSPN ازالگوریتم Aging برای تخمین استفاده می کنیم.

این الگوریتم به گونه ای میانگین گیری می کند که رفتارهای اخیر وزن بیشتری داشته باشند.

- - مثال الگوریتم Aging با ضریب 1/2

$aT_0 + (1-a)T_1$ $\frac{1}{2}T_0 + \frac{1}{2}T_1$

تخمین $T_5 = \frac{1}{2}T_4 + 1/4\,T_3 + 1/8\,T_2 + 1/16\,T_1 + 1/16\,T_0$

$T_5 = \frac{1}{2}T_4 + 1/4\,T_3 + 1/8\,T_2 + 1/16\,T_1 + 1/16\,T_0$

یا می توانیم بنویسیم:

$T_5 = (\frac{1}{2}T_4 + \frac{1}{2}(\frac{1}{2}T_3 + \frac{1}{2}(\frac{1}{2}T_2 + \frac{1}{2}(\frac{1}{2}T_1 + \frac{1}{2}T_0))))$

- مثال الگوریتم Aging با ضریب ½ می باشد و زمان انجام کار ها به ترتیب زیر می باشند زمان کار بعدی را تخمین بزنید:

60 , 90 , 45 , 30 , 70

(60+90)/2= 75

(75 + 45)/2=60

(60 + 30)/2=45

(45 + 70)/2=57.5

1:45:00

- مثال قبل با ضریب 1/3 و داده های زیر:

$1/3T_0 + 2/3\ T_1$

$T_5 = (2/3\ T_4 + 1/3\ (2/3 T_3 + 1/3\ (2/3\ T_2 + 1/3\ (2/3 T_1 + 1/3\ T_0))))$

60 , 30 , 45 , 90

1/3(60)+2/3(30)=20 +20= 40

1/3(40) +2/3(45)=13.3+30= 43.3

1/3(43.3) +

2/3(90)=14.4+60=74.4

2ابتدای صدای

الگوریتم هایی که از این به بعد گفته می شود کمتر در کنکور آمده و می آیند:

الگوریتم زمانبندی تضمین شده :

Guarantied Scheduling

دراین الگوریتم عدالت تضمین می شود که هرکس سهم خودش را ازCPU بگیرد. الزاما این سهم ها مساوی نیستند. همیشه باید محاسبه کند ببیند چه فرایندی کمتر و کدام فرایند بیشتر از سهم خود ازCPU استفاده کرده وسر همه کوانتوم ها این محاسبات باید تکرارشود به همین خاطر سربار زمانی دارد.

-- الگوریتمRR هم عدالت را برقرار می کند ـ بدون هزینه ـ پس چرا ازاین الگوریتم (الگوریتم زمانبندی تضمین شده) استفاده می کنیم؟

RR در زمانی که یک فرایند خواب است حقش را محاسبه نمی کند وجبران نمی کند.

این الگوریتم هزینه زیادی برای عدالتی می کند در صورتی که همه جا این عدالت مناسب نیست.

17:20

زمان بندی بخت آزمایی (به خاطر اینکه مشکل شرعی نداشته باشد: ارمغان بهزیستی) Lottery

این الگوریتم غیر انحصاری و Time sharing است . به هر فرآیند تعدادی بلیط می دهیم و سر هر فرآیند قرعه کشی می کنیم و هر فرآیندی برنده شود کوانتوم را به آن می دهیم. این الگوریتم نوعی مکانیسم اولویت محسوب می شود چون تعداد بلیط ها می توانند نابرابر باشد.

مثال: یک عدد تصادفی انتخاب می کنیم و ازروی عدد انتخاب شده فرایند مربوط را انتخاب می کنیم.

اولویت	1	2,3	4-6	7-10
بلیط بخت آزمایی	1	2	3	4
فرآیند	P1	P2	P3	P4

مزیت های این الگوریتم :

1. به هرفرآیند متناسب با اولویت زمان می دهد.
2. قحطی ندارد در حالی که اغلب زمانبندی های اولویت قحطی داشتند.(اما درحالت خیلی خیلی بدبینانه قحطی دارد)
3. خیلی سریع خودش را با شرایط جدید وفق می دهد. یعنی وقتی یک فرآیند وارد می شود از همان لحظه ورود شانس دارد و گذشت زمان تاثیری در شانس آن ندارد.
4. مبادله بلیط دارد(ticket passing). مثلا P3 ،سه عدد بلیط دارد و P2 دو عدد تا بلیط دارد. P3 به P2 احتیاج دارد پس وقتی که P3 می خواهد بخوابد بلیط هایش را به P2 می دهد.
8. این الگوریتم در دراز مدت خوب است. اولویت ها از بیرون اعمال می شود.
5. علیرغم اینکه این الگوریتم ظاهرا تصادفی است اما درباطن به فرایندهایی که اولویت بیشتری دارد توجه بیشتری دارد.

3 7:50

زمانبندی سهم عادلانه :

FSS (Fair share scheduling)

اولین الگوریتمی است که به جای فرایندها به کاربرها نگاه می کند.

کاربران	User1	User2
فرآیند ها	A , B , C , D	E

درحالت مساوی صف اجرا به شرح زیر است:

A E B E C E D E ....

درحالتی که وزن دار باشد (WFSS) مثلا کاربر اول 2 برابر سهم داشته باشد صف اجرا به شرح زیر تغییر پیدا می کند:

A B E C D E A B E C D E .....

البته در کتاب های سیستم عامل به هر دوی این ها FSS می گویند و در تست اگر وزن دار بیاید باید آن را مشخص کند

اما در کتاب های شبکه از هم تفکیک شده اند.

این الگوریتم کاملا شبیه RR است فقط به جای فرایند ها به کاربرها نگاه می کند.

41:20

زمانبندی بلا درنگ:

در این زمانبندی مهلت (Dead Line) داشتیم یعنی کار باید قبل از پایان مهلت انجام شود.

به زمانی که فرصت داریم تا کار را تمام شروع کنیم سستی یا laxity می گویند.

Dead line = laxity + process time

• کارهای بلا درنگ

1. نرم --> پرش درفیلم (عدم اجرای آنها خسارت کمتری به بار می آورد)
2. سخت --> ICU

• دسته بندی دیگر

1. قطعی :

- متناوب --> درفواصل زمانی مساوی رخ می دهد.
- غیر متناوب

2. تصادفی :

3) وقایع مشخص نیست کی رخ می دهد مثل پرتاب موشک به هواپیما

درسیستم قطعی وقایع مشخص است که چه زمانی رخ می دهد. مثل یک کارخانه تمام اتوماتیک .

* درسیستم های پریودیک Dead Line را خود پریود مشخص می کند .

$$\sum_{i=1}^{m} \frac{C_i}{P_i} \leq 1$$

سیستم های متناوب (periodic)

Pi پریود واقعه یا پریود رخداد واقعه فرآیند i ام.

Ci مقدار زمان مورد نیاز

مثلا واقعه ای که هر5s رخ می دهد و2s زمان مصرف می کند . 2/5

در سیستم عامل های چند پردازنده ای به جای 1 در فرمول بالا K (تعداد پردازنده ها) بگذاریم.

54:30

الگوریتم نرخ یکنواخت Rate Mono Tonic Algorithm

ص 153

- این الگوریتم غیر انحصاری است.
- این الگوریتم برای متناوب هاست.
- آن که فرکانسش بیشتر است اولویت دارد.

P1: 10 --------Dead line-----> 0.1

P2 : 2 ---------Dead line-----> 0.5

الگوریتم ابتدا زود ترین مهلت EDF

- اگرسیستم تصادفی باشد که اصلا Rate ندارد .
- ابتدا کاری را انجام می دهیم که Dead Line کمتری دارد.

الگوریتم کمترین سستی Least Laxity

- ابتدا آن را انجام می دهیم که سستی کمتری دارد. زیرا مهلت کمتری برای شروع دارد.
- البته این الگوریتم خیلی قابل پیاده سازی نیست چون باید service time را بدانیم که همیشه این امکان وجود ندارد.

۵۴:۳۰

زمانبندی نخ :

زمانبندی نخ الگوریتم نیست بلکه باید یاد بگیریم زمانبندی های قبلی را در دو سطح انجام دهیم .

مثال : فرض کنید دو فرآیند به شکل زیر داریم :

P1	T11 : 2	T12 : 3	
P2	T21 : 1	T22 : 2	T23 : 1.5

زمان ورود فرایند ها : 0

زمان ورودنخ i ام هر فرایند i-1

زمانبندی فرایند ها RR با q=3

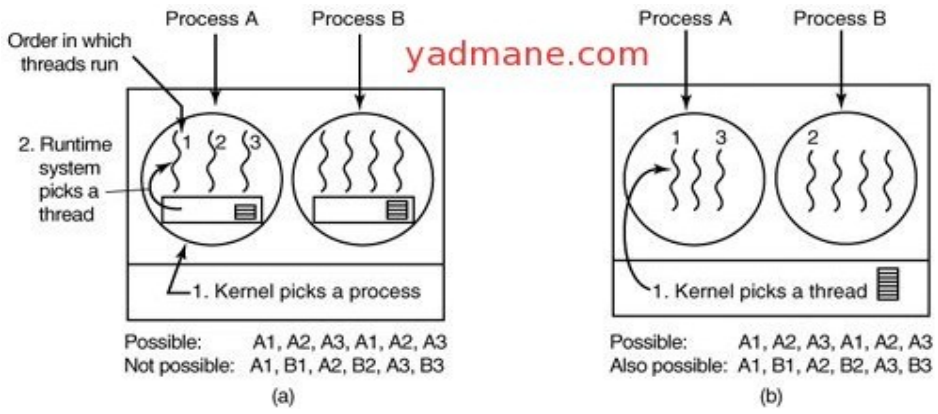
زمانبندی نخ ها FCFS

میانگین زمان پاسخ نخ ها چقدراست؟

```
          P1      |        p2       |    P1   ^|    P2    ^|
T11   ^|  T12  |   T21 ^|    T22   ^|    T12  ^|    T23   ^|
0----------2-------3-------4----------6----------8----------9.5---------
```

ART= (2+4+7+5+7.5)/5 = 5.1

شکل صفحه 155 کتاب نشان می دهد نخ های سطح کاربر و هسته با هم متفاوت هستند.



در سطح کاربر هسته کنترلی روی نخ ها ندارد. (به گزینه های Possible و Not Possible زیر عکس ها توجه کنید)

جلسه دهم پنجشنبه ۱۹/۶/۱۳۸۸ ساعت ۸:۰۰ صبح

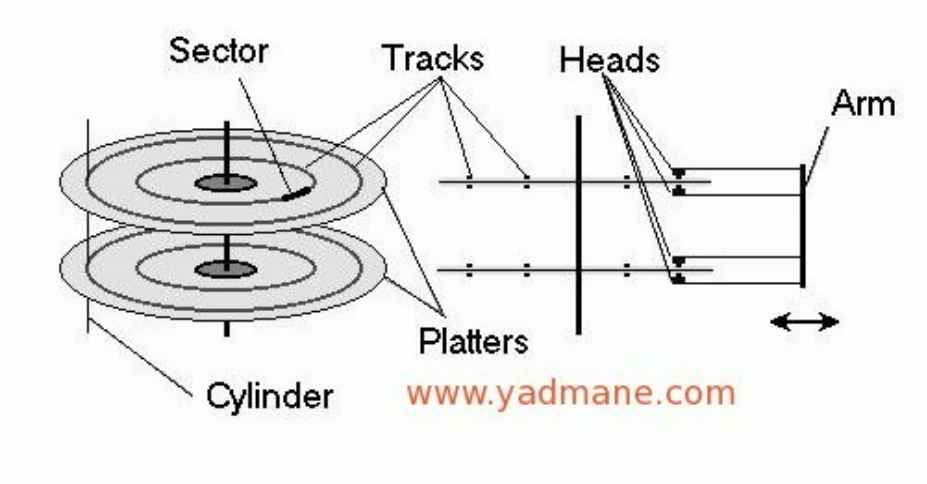
فصل سوم :

1. I/O ها
2. اصول نرم افزاری

تفاوت بین وقفه درخواست I/O و وقفه I/O را قبلا دیده ایم:

1. درخواست I/O : وقتی برنامه کاربر از سیستم عامل یک عملیات I/O را درخواست می کند با یک فراخوان سیستمی (system call) این عمل را انجام می دهد.
2. وقفه I/O : وقتی کار I/O پایان می یابد دستگاه I/O مربوطه به CPU وقفه می دهد.

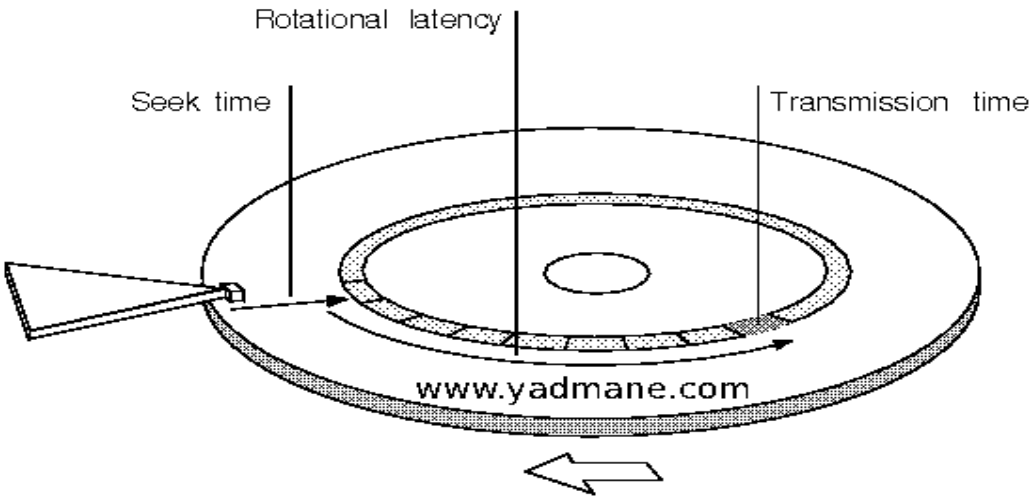
19:10



28:20

- دیسک به دوایر متحد المرکزی به نام track یا شیار تقسیم می شود.
- شیار های زیر هم را سیلندر یا استوانه می گویند.
- با زو های دیسک در جهت شعاع دایره حرکت می کنند.

- زیر و روی دیسک می تواند head قرار داشته باشد.
- Head ها به موازات هم می توانند بخوانند یا بنویسند.



برای خواندن یا نوشتن در بلاک ها باید 3 حرکت صورت گیرد:

1. هد در جهت شعاع دایره باید به سمت شیار مورد نظر حرکت کند. این حرکت مکانیکی را Seek می گویند و به این زمان seek time گویند. تاخیر چرخشی یا درنگ دورانی نیز به آن می گویند.
2. مدت زمانی که طول می کشد تا سکتور مورد نظر حرکت کند و به زیر هد برسد که به آن Rotational Latency می گویند.
3. مدت زمانی که لازم است تا بلاک مورد نظراز زیر head بگذرد را Transfer Time نام می گویند.

- مجموع این سه تاخیر ، کل زمان تاخیر برای انجام عملیات است .
- چون این سه عملیات مکانیکی هستند این زمان نسبتا زیاد است.

33:30

انواع روش های محاسبه Seek Time:

1. محاسبه دقیق
2. محاسبه مینیمم
3. محاسبه میانگین

درمحاسبه دقیق می دانیم هد دقیقا روی کدام شیاراست و می خواهیم به n شیارازk دسترسی داشته باشیم.

فرض کنید هد روی شیار 20 قرار دارد می خواهیم دو بلاک از شیار 12 را بخوانیم . حرکت از هر شیار به شیار مجاور 3 میلی ثانیه طول می کشد. مدت زمان دقیق seek time برابر 8 *3 خواهد شد.

گاهی اوقات ما نمی دانیم ازکجا می خواهیم به کجا باید برویم و باید از Ave یا Min استفاده کنیم .

فرض کنید می دانیم میانگین seek time برابر 10 میلی ثانیه و مینیمم seek time برابر 5 میلی ثانیه می باشد می خواهیم از دو شیار مجاور بلاک هایی را بخوانیم مدت زمان seek time چقدر خواهد بود؟

چون برای رسیدن به شیار اول نگفته است هد کجاست و به کجا باید برویم باید میانگین بگیریم اما برای شیار دوم چون شیار مجاور است باید از مینیمم استفاده کنیم.

37:00

محاسبه Rotational Latency :

معمولا به صورت دقیق محاسبه نمی کنیم و از میانگین استفاده می کنیم.

در اغلب مسائل در این مورد اطلاعاتی نمی دهند و باید میانگین را با استفاده از زمان یک دور کامل و تقسیم آن بر دو محاسبه کنیم.

به عنوان مثال اگر سرعت دیسک 6000 دور در دقیقه باشد:

دور بر ثانیه 100 = 6000/60

میلی ثانیه10 ثانیه = 1/100زمان یک دور =

Average rotational latency = 5 msec

محاسبه transfer time :

زمان ترانسفز = (تعداد کل بلاک درون شیار) / (تعداد بلاک هایی که باید منتقل شود) * (زمان یک دور چرخش)

مثال : می خواهیم یک فایل 24kb ی را بخوانیم . هرشیار حاوی 8 بلوک 2kb ی است اگرسرعت چرخش 6000rpm باشد وMin زمان جستجو 2ms ومیانگین زمان جستجو 15ms باشد و بلوک های فایل پشت سر هم باشند کل عملیات خواندن چقدر طول می کشد؟

برای ترانسفر یک پنجم دور باید بچرخیم .

شیاراول :

Ave. Seek Time + Ave. Rotational + Transfer (8Block)

15ms 5ms 8/8 * 10ms

شیار بعدی :

Min Seek Time +Ave. Rotational + Transfer (4Blook)

2ms 5ms 4/8 * 10ms

15 + 5 + 10 + 2 + 5 + 5 = 42ms

54:20

Raid : Redundant array of Independent disk

آرایه افزونه ای ازدیسک های مستقل

Raid ها برای استقرار ذخیره ساز های پایدار مستحکم و قابل اطمینان (Stable storage) بوجود می آیند مثلا در شبکه نیاز به هارد هایی داریم که قابلیت اطمینانشان بالا باشد.

مکانیسم های مختلفی برای Raid وجود دارد مثل مکانیسم های raid0 تا raid5.

RAM Disk

شبیه سازی دیسک روی رم (ایجاد سیستم فایل روی رم)

ازآنجایی که سیستم های فایل عادت به خواندن ازدستگاه های بلاک بندی شده مثل هاردها را دارند می آییم روی رم شبیه آن سکتوربندی می کنیم تا سرعت کار بالا برود.

درRam Disk چیزی به اسم seek Time نداریم .

1:00:00

الگوریتم های زمانبندی بازوی دیسک

صفحه 345

CPU بسیارسریع است ، RAM کمی کندتر است اما دیسک مکانیکی است و کند است وسرعت PC را پایین می آورد وعلت کندی دیسک نیز حرکت های مکانیکی آن مثل seek time است. از طرفی سیستم عامل ها با دیسک زیاد کار می کنند تا حدی که یکی از سیستم عامل ها DOS یا Disk Operating System نام گرفت و این نشاندهنده نقش دیسک در کار سیستم عامل است. به همین خاطر نمی توانیم نسبت به زمان های سربار آن بی تفاوت باشیم. هر چند کار ما سخت افزار نیست و نمی خواهیم سرعت سخت افزار را بهبود بخشیم اما حداقل از سرعت موجود دیسک استفاده بهینه را ببریم.

ـ فرض کنید 50 درخواست آمده است که باید به خاطر آن شیارهای مختلفی را بخوانیم . چه ترتیبی برای خواندن این شیار ها در پیش بگیریم که مدت seek time کمتری داشته باشیم.

فرض کنید شما نامه رسان یک ساختمان 100 طبقه هستید و در هر طبقه نامه های جدیدی هم به شما می دهند روش های مختلفی داریم :

1. اگر FiFo عمل کنیم ممکن است این شیارها خیلی ازهم فاصله داشته باشند که این باعث خواهد شد Seek Time بالا برود . مثلا اولی درشیار2 ودومی درشیار100 باشد.

بد ترین جواب را در بین الگوریتم ها زمانبندی بازوی دیسک دارد.

2. ابتدا کوتاه ترین (نزدیکترین) زمان جستجو (shortest seek Time First یا SSTF)

مزایای این روش : کاهش Ave Seek Time (لزوما min نیست)

معایب : قحطی دارد. اگر درخواست های متوالی در یک محدوده نزدیک به هم بیاید سایر قسمت های دیسک دچار قحطی می شوند.

3. آسانسور

1:18:00

در این الگوریتم بازو در یک جهت حرکت می کند و در مسیر به تمام درخواست های آن مسیر پاسخ می دهد و دوباره معکوس حرکت می کند.

مزایای الگوریتم آسانسور:

- Ave Seek Time را کاهش می دهد.
- قحطی ندارد
- این الگوریتم را به نام های آسانسور یا scan می خوانند.

Silberschatz می گوید چون بالا ترین درخواست ممکن است 40 باشد ونیازی به رفتن به 100 نیست پس باید نگاه کند بالاترین درخواست کدام است به همین خاطر بهاین الگوریتم Look می گوید.

4. الگوریتم CScan یا Circular Scan

از صفرتا 100 درخواست ها را انجام می دهد دوباره ازپایین به بالا حرکت می کند یعنی درمسیر برگشت به درخواست ها رسیدگی نمی کند.

در آسانسور معمولی اگر درخواستی در پایین ترین طبقات باشد و آسانسور همین الان از آن طبقه عبور کرده باشد باید یک رفت و برگشت منتظر بماند که بیشترین زمان ممکن می شود. اما در آسانسور گردشی فقط یک رفت باید منتظر بماند چون مسیر برگشت به درخواست ها پاسخ نمی دهد هر چند زمانی را صرف بازگشت می کند اما چون به درخواست ها پاسخ نمی دهد این زمان قابل چشم پوشی است یعنی در بازگشت فقط seek time را داریم و زمان ترانسفر و rotational latency را دیگر ندارد.

هدفاین الگوریتم این است که ماکسیمم زمان انتظار را کاهش دهد.

چسبندگی:

همه الگوریتم هایی که تاکنون گفته ایم چسبندگی دارند. حالتی که سیلی ازدرخواست ها برای یک شیار بیاید در این صورت دیگر به درخواست سایر شیار ها پاسخ داده نمی شود که به آن چسبندگی می گویند.

چسبندگی نیز باعث قحطی می شود. البته به این دلیل نمی گوییم همه الگوریتم ها قحطی دارند بلکه می گوییم چسبندگی دارند.

راه حل های پیشنهادی برای حل مشکل چسبندگی:

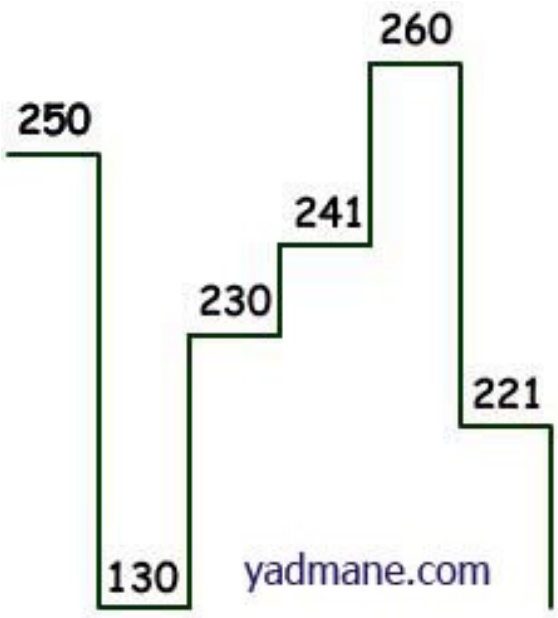
دوصف درنظر می گیریم و هر بار که درخواست یک صف را بررسی می کنیم دیگر اجازه ورود به این صف را نمی دهیم ودرخواست های جدید وارد صف دوم می شوند وقتی صف اول تمام شد سراغ صف دوم می رویم و وقتی درخواست های صف اول را پاسخ می دهیم درخواست های جدید وارد صف اول می شوند.

- FScan: اگردوصف داشته باشیم .
- Nscan: اگر n صف داشته باشیم .

1:45:00

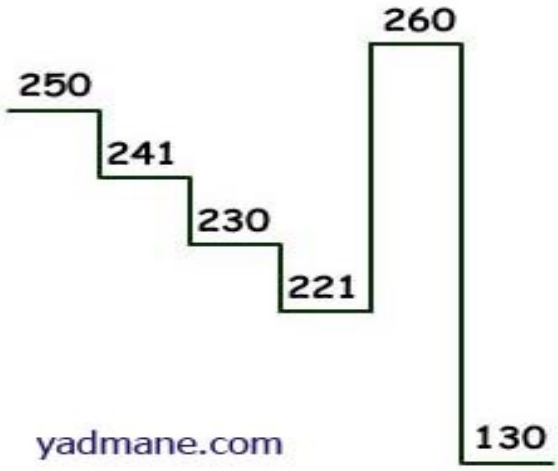
--فرض کنید که درخواست هایی برای شیارهای 221و260 و 241و 230 و به ترتیب از چپ به راست130 وارد شده است . هد روی شیار 250 قرار دارد و جهت قبلی رو به بالا بوده است . زمان حرکت از هر شیار به شیار مجاور 3 میلی ثانیه است. کل زمان جستجو در الگوریتم های FCFS و SCAN و SSTF چقدر است؟

FIFO:



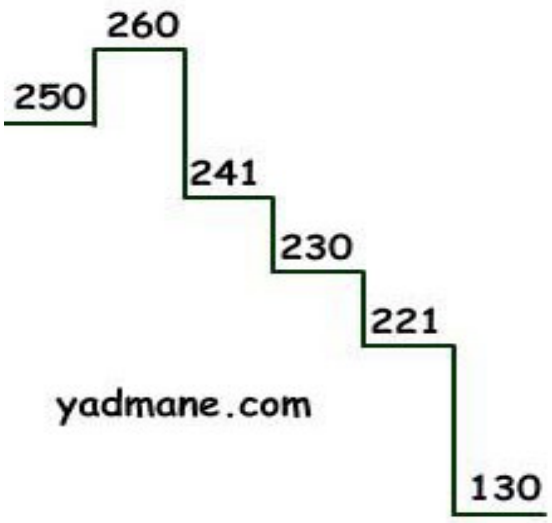
3 * ((250-130) + (230-130) + (241-230) + (260-241) + (260-221))= 289*3 = 867

SSF:



3 * ((250-241) + (240-230) + (230-221) + (260-221) + (260-130))= 594

SCAN:



در مورد آسانسور لازم نیست تکه تکه حساب کنیم روبه بالا ها را یکجا و رو به پایین ها را یکجا حساب مب کنیم:

3 * ((260-250) + (260-130))= 420

ابتدای صدای 2

بن بست یا Dead Lock

صفحه 297

یک مجموعه از فرآیند ها در صورتی منجر به بن بست می شوند که هر یک از فرآیند های درون مجموعه منتظر رویدادی باشد که فقط یک فرآیند دیگر از همین مجموعه می تواند آن رویداد را انجام دهد.

بن بست یک سیکل انتظار است . گروهی که دریک سیکل همگی منتظر هم هستند.

چرا یک فرایند منتظر یک فرایند دیگر می شود؟

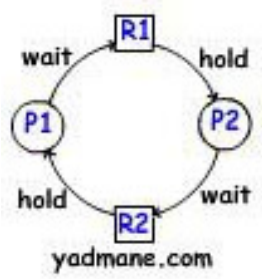
1. انتظار برای آزاد شدن منبعی که در اختیار فرآیند دیگر است.
2. انتظار برای نتیجه محاسبات ازطرف فرآیند دیگر
3. انتظار برای دریافت سیگنال ازطرف فرآیند دیگر

درمثال آخردو فرآیند همدیگر را می شناسند اما در حالت اول دو منبع همدیگر را نمی شناسد و فقط منبع را می خواهند این دونوع ذاتاً باهم متفاوت هستند.

- اگر بن بست به خاطر منابع باشد به آن Resource Dead Lock یا بن بست منابع می گویند.
- اگر فرآیند ها منتظر سیگنال یا پیام از یکدیگر باشند به آن Communication Dead Lock می گویند.
- ترکیب این دو هم ممکن استپیش آید به عنوان مثال یکی منتظر منبع دیگری و یکی منتظر سیگنال ازطرف دیگری باشد

دراکثر کتابها فقط به بن بست منابع پرداخته شده است.

مثال: در شکل زیر بن بست رخ داده است چه راه حلی را می توانیم داشته باشیم:



1 منبع R1 به زور از P1 بگیریم و به P2 بدهیم.

برای برخی منابع ممکن است بتوانیم منابع را به زور بگیریم مثل CPU ولی بعضی منابع امکان ندارد و پس گرفتنشان با تخریب همراه است مثل پرینتر که اگر وسط کار بگیریم صفحه پرینت گرفته شده خراب می شود.

20:50

انواع منابع :

- انحصاری (Non Preemptive) مثل پلاتر --> بن بست خیز
- غیر انحصاری (Preemptive) مثلCPU --> باعث بن بست نمی شود چون غیرانحصاری است. هرچند CPU را می توان انحصاری نیز زمانبندی کرد. اما CPU این قابلیت را دارد.

2 منبع R1 به P1 همراه با P2 سرویس بدهد.

دسته بندی دیگر منابع :

- Mutual exclusive --> دارای انحصار متقابل ، دوبه دو ناسازگار ، CPU وپرینتر

دو نفر با هم همزمان نمی توانند پرینت بگیرند.

- Non Mutual exclusive--> بدون انحصار متقابل ، خواندن ازفایل

پرینتر هم مشکل انحصار متقابل را دارد و هم انحصاری بودن را.

بن بست وقتی رخ می دهد که هر دو مشکل بالا وجود داشته باشد.

31:45

3 یکی را kill می کنیم. کدام یک؟ کم اهمیت تر ؟ جوان تر؟

اگر بن بست رخ داده باشد بدون خون و خونریزی نمی توان ازآن خلاصی یافت.

اگر منابع اولویت دارند آنهایی که اولیت کمتری دارند بکشیم اما اگر اولویت ها برابر است جوانترها را بکشیم زیرا از منابع کمتری استفاده کرده اند وتکرار آنها هزینه کمتری دارد. به این روش ترمیم بن بست یا deadlock recovery می گویند. این recovery ترمیم نیست در واقع روپوشانی بن بست است.

شرایط بن بست:

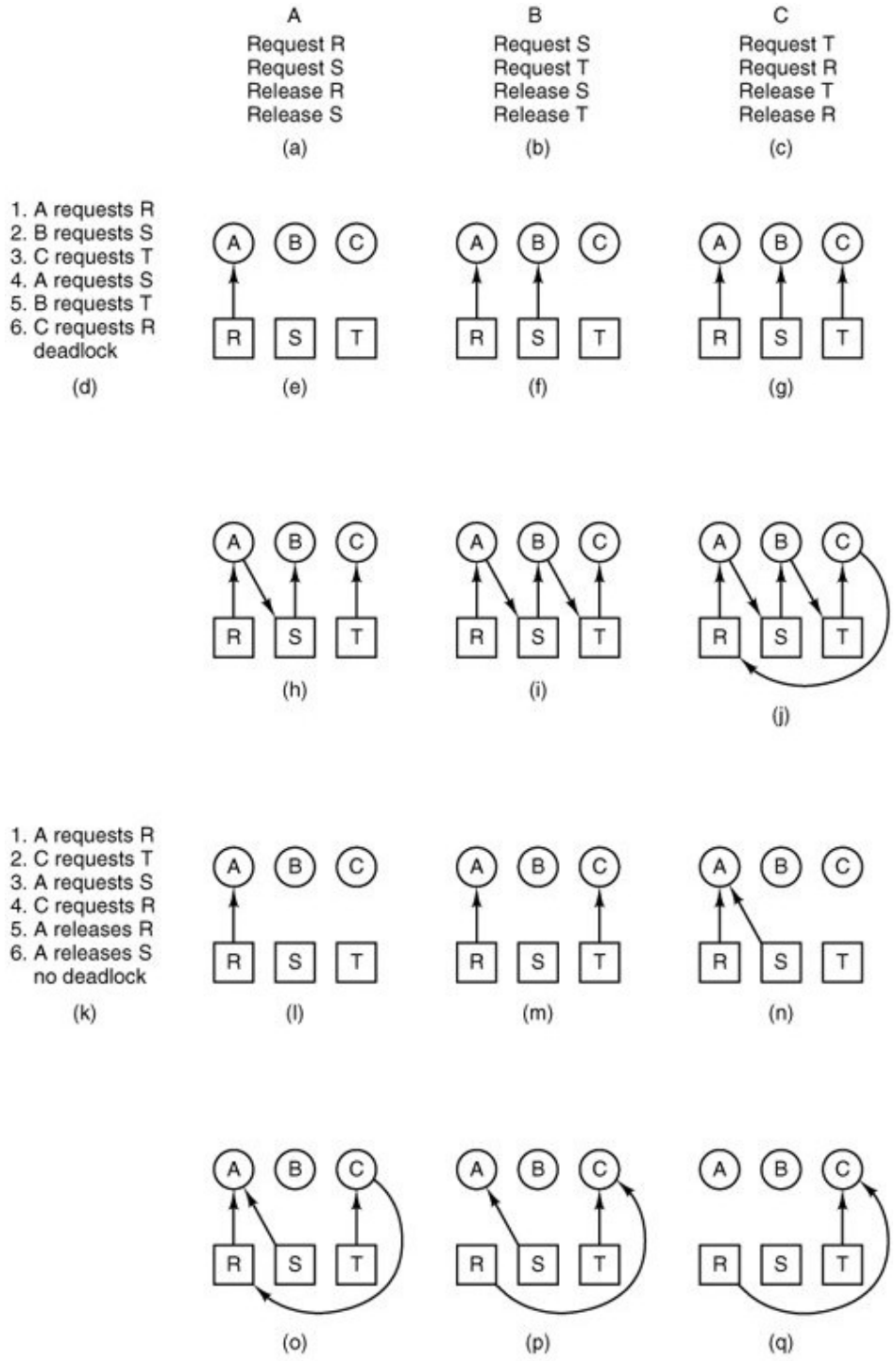
صفحه 298 کتاب

چهار شرط زیر برای وقوع بن بست منابع لازم و کافی است:

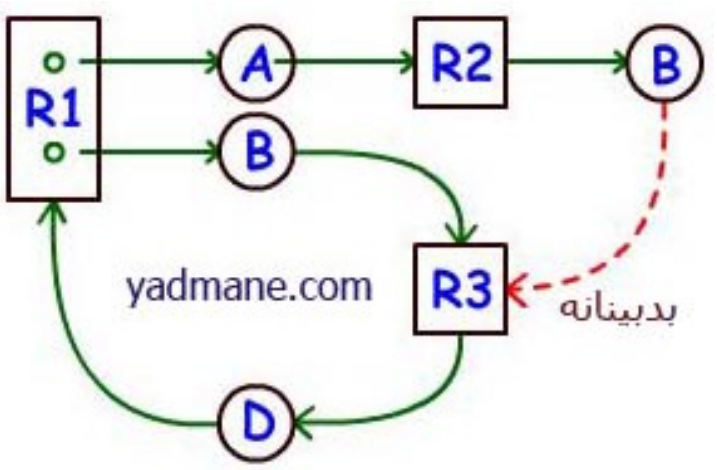
1. شرط انحصار متقابل (دو به دو ناسازگاری)
2. شرط نگهداری و انتظار : partial allocation به معنی تخصیص ناقص یا تخصیص بخشی از منابع در واقع معادل hold & wait است.
3. شرط انحصاری (نداشتن تخلیه پیش هنگام)
4. شرط انتظار چرخشی

سه شرط اول جزء ذات منبع باید باشد تا امکان وقوع بن بست وجود داشته باشد اما شرط چهارم است که رخداد آن باعث بن بست می شود.

گراف تخصیص منابع:



مثال - در شکل زیر آیا بن بست رخ داده است یا نه؟



1. بن بست است؟ خیر
2. بن بست نیست؟ بلی
3. بن بست خواهد شد؟ شاید
4. بن بست نخواهد شد؟ شاید

نگرش بد بینانه این است که بن بست رخ خواهد داد(B منبع R3 را می خواهد) امااگر خوش بینانه نگاه کنیم ممکن است بعضی منابع آزاد شوند و بن بست رخ نخواهد داد.

1:03:00

استراتژی برخورد با بن بست :

صفحه 301

1. استراتژی شترمرغ (ostrich) : اصل این استراتژی این است که بی خیال شویم، انشاءاله رخ نخواهد داد، نفوس بد نمی زنیم .

1:11:00

تنبلوم : شاید اگرشما از مسئله صرفنظر کنید مسئله نیز ممکن است مسئله نیز از شما صرفنظر کند.

حافظ: سخت می گیرد جهان بر مردمان سخت کوش

1:21:00

2. بهتر است این استراتژی آخر آمده بود و زمانی سراغ این استراتژی می رویم که سه راه دیگر نتیجه نمی دهد.

3. استراتژی کشف وترمیم : چگونه OS می فهمد که بن بست رخ داده است ؟ تا قبل از وقوع بن بست این سیستم عین شترمرغ است تفاوت آن بعد از وقوع است . پس از کشف (Detection) بن بست باید Recovery (ترمیم) انجام داد. یعنی فرایند ها را به ترتیب اولویت می کشد. تا از وضعیت بن بست خارج شود.

1:33:00

4. پیشگیری ازبن بست :

این استراتژی خیلی ترسو است .

به خاطر جلوگیری ازبن بست خود را از خیلی از منابع محروم می کند در صورتی که احتمال وقوع بن بست بسیار نادر است و حتی با این استراتژی هیچ تضمینی در جلوگیری از بن بست وجود ندارد.

1 اجتناب : در پیشگیری خیلی خودش را محدود می کند اما هر لحظه فکر نمی کند. دراجتناب به بن بست نزدیک می شویم ، اما با احتیاط ، همچنین وارد هر وضعیت جدیدی که می شویم شرایط جدید را چک می کنیم تا منجر به بن بست نشود.

- **امن** : وقتی شرایط فرار میسر باشد.
- **نا امن** : شرایط فرار نداریم شاید بن بست رخ دهد. ممکن است هزاران بار ازمسیر نا امن بگذریم و بن بست رخ ندهد.
- خواهیم دید که روش های اجتناب نیز جواب نمی دهد زیرا آینده را نمی توانیم پیش بینی کنیم.
- شکست الگوریتم بنکر به علت عدم توانایی ما در پیش بینی نیاز آینده فرآیند ها به منابع است.

بررسی بیشتر الگوریتم کشف و ترمیم :

- **چگونه می توان بن بست را کشف کرد؟ (چه زمان گراف تخصیص منابع را چک کنیم که بن بست شده یا نه؟)**
- **یک پیشنهاد این است که هر زمان درخواست برای منابع از یک حدی بالا تر رفت چک کنیم ببینیم بن بست رخ داده یا نه؟**
- **هر زمان فرآیندی درخواست منبعی می دهد چک کنیم ببینیم آیا این درخواست منجر به بن بست می شود یا نه؟**

البته بهتر است درخواست هایی را چک کنیم که نمی توانیم پاسخ دهیم.

تنها انتظاری که سیستم عامل می تواند تشخیص دهد ابدی است یا خیر بن بست است چون سیستم عامل می تواند گراف های تخصیص را چک کندو سیکل را پیدا کند اما درمورد حلقه های تکرار نمی توانیم تشخیص بدهیم که تا ابد در لوپ می چرخد چون OS از درون فرآیند خبر ندارد.

اما این کار معمولا انجام نمی شود چرا که تعداد پروسس ها و منابع زیاد است (هزاران منبع داریم در واقع منابع فقط پرینترو ... نیست ، فیلد ها و فایل ها نیز نیز منابع هستند.) مثلا ممکن است جدول فرایند خود به عنوان منبعی باعث بن بست شود.

تننباوم مثالی می آوید که pcb باعث بن بست می شود.

دلیل عدم استفاده از الگوریتم کشف و بن بست در سیستم عامل ها وقت گیر بودن جستجو و کشف بن بست نیست دلیل آن پریود زیاد آن است . پریود آن بیش از زمان اجرای دستورات و بسیارپرهزینه است. ضمن آن که این همه هزینه برای کشف یک پدیده نادر است. دست آخر اینکه وقتی بن بست رخ داد آنگاه باید تعدادی از فرآیند ها را kill کند. که اگربه دنبال کشف بن بست نباشیم و رخ دهد نیز تقریبا همان هزینه را دارد . یعنی کاربر خودش فرآیند ها را kill می کند مثلا با restart.

جلسه یازدهم دوشنبه ۱۳۸۸/۶/۲۳ ساعت ۱۲:۰۰

پیشگیری ازبن بست :

صفحه 303 کتاب

پیشگیری از بن‌بست پرهزینه ترین روش برخورد با بن بست است حتی اگر هزینه بسیار زیادی هم بکنیم موفقیتمان صد درصد نخواهد بود.

اگر کاری کنیم که یکی از چهار شرط بن‌بست هرگز هرگز بروز نکند ما از بن‌بست پیشگیری کرده ایم.

روش	شرط
Spoll کردن همه منابع	انحصار متقابل
درخواست تمام منابع در ابتدا	نگهداری و انتظار
باز پس گیری پیش هنگام منابع	انحصاری
شماره گذاری منابع و درخواست آن‌ها به ترتیب (مثلاً صعودی)	انتظار چرخشی

روش‌های پیشگیری بن بست:

هر کدام از روش‌های پیشگیری زیر سعی می‌کند یکی از چهار شرط بن‌بست که در بالا آمده را نقض کند.

شرط اول - انحصار متقابل : می خواهیم کاری کنیم که شرط انحصار متقابل تحقق پیدا نکند یعنی دو نفر بتوانند همزمان ازیک منبع استفاده کنند.

راه حل پیشنهادی استفاده از Spooling است. مثلا پرینتر در دست کس دیگری است. اگر شخص دیگری احتیاج به پرینتر داشته باشد بدون اینکه بداند دیتای آن بر روی دیسکSpool می شود. درواقع ما پرینتر را بر روی دیسک spool کردیم. حتی ممکن است شما پرینت بفرستید و برنامه را ببندید و بعدا و به ترتیب پرینتر پرینت شما را چاپ می کند و دیگر ما منتظر پرینتر نشدیم و برای اینکه پرینتر در دست دیگری است wait نکردیم.

اما با این روش به صورت قطعی نمی توانیم از بن بست جلوگیری کنیم زیرا :

3. بعضی ازمنابع قابل Spool نیستند مثل درآیه های جدول فرایند و PCB ها.
4. خود فضای spool می‌تواند باعث بن‌بست شود. چرا که فضای spool نامحدود نیست و این می‌تواند باعث محدودیت شود.

پس با Spooling نمی توان به صورت قطعی از بن بست جلوگیری کرد تنها با Spooling می‌توان گفت که احتمال بن بست را کاهش می دهیم و ازآنجایی که هزینه چندانی ندارد و همچنین Spooling برای همزمانی استفاده ا زمنابع مفید است آن را انجام می دهیم.

حتی اگر همه منابع را می توانستیم Spool کنیم و spooling مشکل انحصار متقابل را حل می‌کرد باز هم احتمال بن بست وجود دارد چرا که همه بن بست ها به خاطر منابع نیست.

هزینه روش	امکان پیاده‌سازی عملی (در سیستم‌های همه منظوره)	روش	شرط
	اولاً غیر ممکن است . همه منابع قابل spool نمی باشند. مثلاً رکورد های فایل یا PCB. ثانیاً برای منابعی مانند چاپگر نیز به علت محدودیت فضای spool بر روی دیسک شاید spool رخ دهد.	Spool کردن همه منابع بر روی دیسک. مانند spool کردن چاپگر	انحصار متقابل
هدر دادن منابع از لحظه درخواست تا زمان استفاده	غیر ممکن است. چون فرآیند ها در ابتدا باید نیاز آینده خود را پیش‌بینی کنند.	درخواست تمام منابع در ابتدای اجرای فرآیند	نگهداری و انتظار
	غیرممکن است. چون منابع انحصاری اند. در غیر این صورت بن‌بست یوجود نمی آید.	پس دادن منابع فعلی و در خواست مجدد منابع جدید مورد نیاز	
	غیرممکن است. چون منابع انحصاری اند. گرفتن این منابع معادل کشتن آن هاست.	باز پس گیری پیش هنگام منابع	انحصاری
هدر دادن شدی منابع بی کار	غیر ممکن است. چون ممکن است فرآیند ها در هر لحظه به چندین منبع نیاز داشته باشند	هر فرایند در هر لحظه فقط یک منبع در اختیار داشته باشد	انتظار چرخشی
	غیر ممکن است.چون دسترسی فرآیند ها به منابع ترتیب خاصی ندارد. درخواست پیشاپیش منابع برای حفظ ترتیب نیز نیاز به پیش‌بینی آینده دارد.	شماره گذاری منابع و درخواست آن‌ها به ترتیب (مثلاً صعودی)	
	غیر ممکن است. چون دسترسی فرآیندها به منابع، ترتیب خاصی ندارد.درخواست پیشاپیش منابع برای حفظ ترتیب نیز نیاز به پیش‌بینی آینده دارد.	شماره گذاری منابع و درخواست منابعی که شماره آن‌ها بزرگ‌تر از منابع فعلی فرآیند است.	

26:30

شرط دوم - نگهداری و انتظار :

دیگر به منبع گیر نمی دهد بلکه رفتار فرایند ها را کنترل می کند. مثلا تا زمانی که فرآیند منبعی در اختیار دارد حق ندارد منبع جدیدی درخواست کند و یا فرایندها باید تمام منابع مورد نیاز خود را ابتدا درخواست کنند که دراین صورت یا تمام منابع را به آن می دهد یا هیچکدام .

اگر تمام منابع را یک جا به فرآیند بدهیم فرآیند فقط hold می‌کند و هیچگاه wait نمی‌کند در نتیجه hold&wait نداریم و بن‌بست نمی‌شود اما این روش ممکن نیست زیرا اولا منابع را هدر می دهد زیرا ممکن است یک برنامه طولانی در انتهای کارش یک منبع را برای چند لحظه بخواهد ثانیا پیش بینی منابع مورد نیاز ممکن نیست.

33:45

شرط سوم - انحصاری ← غیرقابل اجراست.

اگر منبع غیرانحصاری باشد که باعث بن‌بست نمی‌شود و نیاز به بحث نمی باشد اما اگر انحصاری باشد قابل پیش تخلیه نمی باشد و راه حل پیشنهادی به زور پس گرفتن منبع نیز درواقع کشتن فرآیند است و راه حل نمی باشد.

شرط چهارم - انتظار چرخشی :

روش ها :

4. به فرایندها بگوییم درهرلحظه فقط از یک منبع استفاده کنند . نمی شود چون مثلا فرایند می خواهد با استفاده از CPU داده های RAM را پردازش کند.

نکته: این روش را می‌توان روش مقابله با hold&wait نیز دانست اما تنبلوم آن‌ها را در این دسته آورده است بنابراین

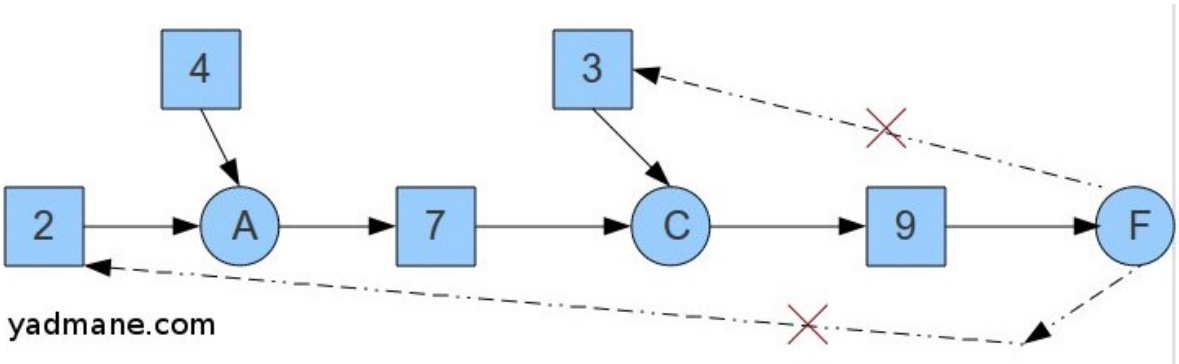
اگر در تست ها آمد باید جزو روش‌های این دسته محسوب کرد.

5. شماره گذاری منابع و درخواست آنها به ترتیب مثلا صعودی چون زنجیره درخواست ها به حلقه تبدیل نمی شود ازبن بست جلوگیری می کند.

با توجه به شکل فرض کنید فرآیند A منبع 2 و4 را گرفته است. سپس درخواست منبع 7 را می‌دهد. در حالی که منبع 7 در اختیار فرآیند C است . همچنین فرآیند C منبع 3 را نیز در اختیار دارد و منبع 9 را نیز درخواست می دهد. منبع 9 در اختیار فرآیند F است.

فرآیند F نمی‌تواند درخواست منبع 3 را بدهد چون سیکل بوجود می‌آید و دچار بن‌بست می شود.

درست است که زنجیره درخواست ها بوجود می‌آید اما چون حلقه ایجاد نمی‌شود باعث بن‌بست نمی شود.



چرا این روش درعمل پیاده سازی نمی شود؟

درخواست منابع به ترتیب ممکن است میسر نمی باشد مگر اینکه منابعی که شماره های کمتری دارند را ازقبل پیش بینی ودرخواست کنیم که خود دومشکل ایجاد می کند:

1. منابع را هدر می‌دهد چون فرض کنید منبع شماره 2 را در انتهای کار بخواهد و در ابتدای برنامه منبع 5 را می‌خواهد اما چون به ترتیب باید منابع را درخواست کند مجبور است در ابتدای کار که منبع 5 را می‌خواهد می بایست منبع 2 را نیز درخواست کند.البته هدر رفتن منابع در این روش به شدت روش قبلی نیست.
2. نیاز به پیش بینی دارد.
3. شماره گذاری منابع و درخواست آنها به ترتیب اما اگر منابع بزرگتر را رها کند می‌تواند منبع با شماره کوچکتر را درخواست کند

این درواقع همان روش قبلی است که یک مقدار محدودیت آن کمتر شده است. اما به هر حال کمابیش همان ایرادات روش قبلی را دارد.

54:45

اجتناب از بن‌بست :

اگر چه این روش نیز موفق نمی‌شود اما زیباترین روش ریاضی است.

الگوریتم دکسترا می‌گوید که من از بانکدار شهرم یاد گرفتم که هر لحظه حرکاتم را سنجیده انجام دهم و وارد وضعیت نا امن نشوم.

الگوریتم بنکر برای یک منبع منفرد :

صفحه 308

یک بانکدار لحظه به لحظه شرایط را بررسی می کند تا در وضعیت پر خطر نباشد.

قاعده بانک این است که اگر مشتری درخواست وام بکند و بانک مبلغ مورد نظر را نداشته باشد بدهد آن مشتری را مسدود می‌کند وقتی مشتری مسدود می‌شود وام هایی را که در اختیار دارد را پس نمی‌دهد تا وام جدید را بگیرد. پس بنکر همیشه باید مراقب باشد مانده موجودی بانک را به گونه‌ای حفظ کند که حداقل یکی از مشتریان تمام اعتبار مورد نیلزش را بگیرد و همزمان همه مشتری‌ها مسدود نشوند.

بانکی را در نظر بگیرید که چهار مشتری دارد. هر مشتری یک سقف اعتبار دارد.

موجودی یا Existing اولیه این بنکر 10 است. وضعیت الف وضعیت اولیه است. در وضعیت ب بانک مقداری کار کرده و مشتری‌ها مقداری وام دریافت کرده اند. با دادن یک منبع دیگر به فرآیند B از وضعیت ب به وضعیت ج می رویم.

فرآیند	تخصیص یافته	حداکثر نیاز
A	0	6
B	0	5
C	0	4

الف

فرآیند	تخصیص یافته	حداکثر نیاز
A	1	6
B	1	5
C	2	4

ب

فرآیند	تخصیص یافته	حداکثر نیاز
A	1	6
B	2	5
C	2	4

ج

تا این مرحله از 10 منبع 8 تای آن استفاده شده و 2 منبع آزاد می باشند

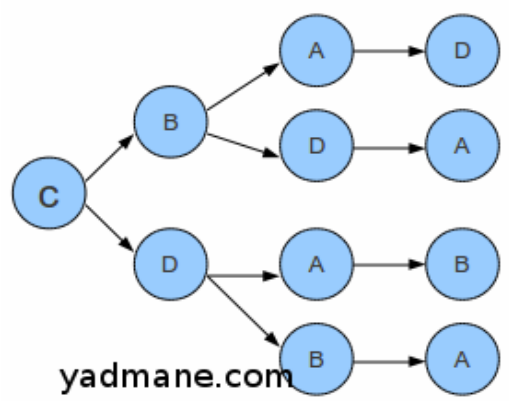
1:06:30

در وضعیت ج تنها یک منبع آزاد داریم که هیچ یک ازفرآیند ها با گرفتن آن کل منابع مورد نیازش را نمی‌تواند دریافت کند و وضعیت نا امن است اما بن‌بست نیست چرا که ممکن است یکی از فرآیندها قسمتی از منابع خود را باز گرداند و بن‌بست نشود.

در وضعیت ب دو منبع آزاد داریم که حداقل یکی از فرآیند ها (C) می تواند با گرفتن کل منابع مورد نیازش Run To Complete می شود و پس از Complete شدن کل منابع مورد نیازش آزاد می‌شود که 4 است با 4 منبع می توان B را Run To Complete اجرا کرد و .... و در نهایت دچار بن‌بست نمی‌شویم پس امن است.

پس راه فرار Run To Complete اجرا کردن تک تک فرآیند هست.

تعداد مسیر های امن مطابق شکل زیر 4 مسیر است :



پارامترها و تعاریف :

Max → Maximum

Alloc → Allocated (used)

Need → Request still needed = **Max - Alloc**

E → Existing Resource

P → Processed Resource = Sum of Allocated

A → Available Resource (Free Resource)

Request → درخواست کنونی فرایند است وبا need متفاوت است

وقتی می‌گوییم امن است یعنی حداقل یک مسیر امن داریم. و وقتی می‌گوییم نا امن است یعنی هیچ مسیر امنی نداریم .

برای اینکه بفهمیم مسیر امن است یا نه باید مسیر را تا آخر مسیر برویم چون ممکن است در گام های بعدی مشکل ایجاد شود.

برای حل مسئله به 3 فاکتورنیاز داریم .

- Alloc
- Need
- Available (برای مقایسه با need)

if Need<Available → Available(new) = Available(old) + Alloc

الگوریتم بانکداری منابع چند گانه

صفحه 310

سه ماتریس داریم :

4. Allocated
5. Need
6. E و P و A

Alloc

فرایند	منابع			
A	3	0	1	1
B	0	1	0	0
C	1	1	1	0
D	1	1	0	1

Need:

فرایند	منابع			
A	1	1	0	0
B	0	1	1	2
C	3	1	0	0
D	0	0	1	0

E=(6342)

P=(5322)

A=(1020)

- **با توجه به داده‌های بالا :**

Alloc= Max-Need → 5 3 3 2

- **با جمع ستونی P بدست می آوریم.**
- از تفریق P و E می‌توانیم Available را بدست آوریم.

Available = E – P

2ابتدای صدای

در جداول صفحه قبل چند مسیر امن وجود دارد؟

A: 1 1 0 0 >= 1 0 2 0

B: 0 1 1 2 >= 1 0 2 0

C: 3 1 0 0 >= 1 0 2 0

D: 0 0 1 0 <= 1 0 2 0

E: 2 1 1 0 >= 1 0 2 0

با توجه به مندرجات بالا مسیر فرار در گام اول D می باشد.

بعد از اینکه D را Run To Complete اجرا کردیم منابع آن آزاد می‌شوند در نتیجه:

Available(new) = Available(old) + Alloc(D) = 1 0 2 0 + 1 1 0 1 = 2 1 2 1

در گام بعدی:

A: 1 1 0 0 <= 2 1 2 1

B: 0 1 1 2 >= 2 1 2 1

C: 3 1 0 0 >= 2 1 2 1

E: 2 1 1 0 <= 2 1 2 1

در این گام هم A و هم E را می‌توانیم انتخاب کنیم اما بهتر است که فرآیندی را انتخاب کنیم که منابع Allocate بیشتری دارد و در نتیجه پس از Run To Complete منابع بیشتری را آزاد می کندپس در اینجا A را انتخاب می کنیم:

Available(new) = Available(old) + Alloc(A) = 2 1 2 1 + 3 0 1 1 = 5 1 3 2

در گام بعدی:

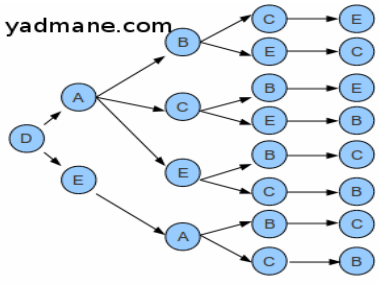
B: 0 1 1 2 <= 5 1 3 2

C: 3 1 0 0 <= 5 1 3 2

E: 2 1 1 0 <= 5 1 3 2

در گام سوم هر سه فرایند باقی‌مانده را می‌توانیم انتخاب کنیم پس همینجا می توانیم تشخیص دهیم که امن است.

محاسبه تعداد مسیر های امن:



تست 2 صفحه 22 جزوه تست: اگر بخواهیم در روش پیشگیری (prevention) برای مقابله با بن‌بست به شرط نگهداری و انتظار حمله کنیم و منابع را در ابتدای کار به طور کامل درخواست کنیم چه مشکلاتی خواهیم داشت؟

1. نیاز به پیش‌بینی نیاز فرایند ها در آینده دارد، منابع هدر می‌رود، Spool کردن همه منابع غیرممکن است، ممکن است که خود فضای spooling بر روی دیسک موجب بن‌بست شود.
2. نیاز به پیش‌بینی نیاز فرایند ها در آینده دارد، منابع هدر می‌رود، Spool کردن همه منابع غیرممکن است.
3. نیاز به پیش‌بینی نیاز فرایند ها در آینده دارد، منابع هدر می‌رود.
4. Spool کردن همه منابع غیرممکن است، ممکن است که خود فضای spooling بر روی دیسک موجب بن‌بست شود.

تست ۲ صفحه ۲۲: برای پیشگیری از بن‌بست از طریق انتظار چرخشی، درخواست منابع به ترتیب عددی ، پیشنهاد شد. در پیاده‌سازی آن با چه مشکلی روبرو خواهیم شد؟

1. نیاز به پیش‌بینی آینده دارد و خود فضای spool موجب بن‌بست می شود.
2. نیاز به پیش‌بینی آینده دارد و منابع سیستم را هدر می دهد.
3. خود فضای Spool موجب بن‌بست می‌شود و منابع سیستم را هدر می دهد.
4. نیاز به پیش‌بینی آینده دارد، خود فضای Spool موجب بن‌بست می‌شود و منابع سیستم را هدر می دهد.

تست ۳ صفحه ۲۲: کدامیک از روشهای زیر مربوط به استراتژی جلوگیری از بن‌بست نمی باشد؟

1. Spool کردن منابع
2. افزایش تعداد منابع
3. شماره گذاری منابع و درخواست منابع به ترتیب شماره توسط هر فرآیند
4. درخواست همزمان کلیه منابع مورد نیاز

توضیح راجع به تست ۴:

در این مسأله بن‌بست وقتی رخ می‌دهد که هر کدام از فرایند ها یک منبع را بگیرد و منبع دوم را بخواهد. اما اگر حداقل یکی تعداد منابع بیش از فرایند ها باشد حداقل یکی از فرایند ها هر دو منبعش را می‌گیرد و دیگر بن‌بست رخ نخواهد دارد.

حال برای اینکه به یک فرمول برسیم فرض کنید که 8 فرایند داریم و هر فرایند 3 منبع بخواهد . حداکثر تعداد منابعی را که با وجود آن بن‌بست احتمال بن‌بست وجود دارد زمانی رخ می‌دهد که هر فرایند دو منبع بگیرد و منبع سوم را بخواهد.

حال برای n فرایند که هر فرایند 3 منبع می‌خواهد اگر 2n منبع داشته باشیم احتمال بن‌بست وجود خواهد داشت.

برای n فرایند اگر هر کدام k منبع بخواهند با وجود :

$n*(k-1)$

باز هم احتمال بن‌بست وجود دارد و اگر بخواهیم بن‌بست نشود باید یک منبع از این تعداد بیشتر داشته باشیم.

به طور کلی اگر :

n : تعداد فرایند ها

m : تعداد منبع یکسان قابل استفاده مجدد

k : حداکثر نیاز هر فرایند به منبع

برای اینکه بن‌بست اصلاً رخ ندهد می بایست:

$m > (k-1)n$

$m > nk – n$

$nk < m + n$

اگر k ها با هم متفاوت باشند چطور؟

اگرتعداد منابع مورد نیاز فرایند ها از 1 تا n باشد اگر هر فرایند یکی کمتر از تعداد منابع مورد نیازش منبع گرفته باشد به یک منبع دیگر نیاز داریم تابن بست نشود.

یعنی مجموع ki ها باید کوچکتر از n+m باشد.

$SUM(k_i) < n + m$

تست۴صفحه۲۲: کامپیوتری دارای m عدد از یک نوع منبع است و n فرایند برای در اختیار گرفتن آن‌ها با هم رقابت می کنند. هر فرایند حداکثر به 2 منبع نیاز دارد . حداکثر تعداد فرآیند ها (n) که بازای آن می‌توان مطمئن بود سیستم دچار بن‌بست نمی‌شود چند است؟

توضیح:

$2n < n + m \quad \rightarrow \quad n < m \quad \rightarrow \quad$ حداکثر $m-1=n$

5. m
6. m+1
7. m+2
8. m-1

تست۵صفحه۲۲: کامپیوتری دارای 120 عدد از یک نوع منبع است و 25 فرایند برای در اختیار گرفتن آن‌ها با هم رقابت می کنند. اگر هر فرایند حداکثر به چند منبع نیاز داشته باشد می‌توان مطمئن بود که سیستم دچار بن‌بست نمی شود؟

$25k < 25+120$

$k=5: 125 < 145$

$k=6: 150 < 145$

1. 3 منبع
2. 4 منبع
3. 5 منبع
4. 6 منبع

تست ۶ صفحه ۲۳: سیستمی شامل 4 فرایند همروند و 2 منبع قابل استفاده مجدد را در نظر بگیرید. به شرط آنکه هر فرایند حداکثر به 2 منبع نیاز داشته باشد، تعداد وضعیت‌های بن‌بست (Deadlock states) در این سیستم حداکثر چند می باشد؟

توضیح: ترکیب 2 از 4 حالت می شود.

1. 3
2. 5
3. 6
4. 8

تست ۶ دوم صفحه ۲۳: سیستمی شامل 3 فرایند همروند و 2 منبع قابل تخلیه پیش هنگام را در نظر بگیرید. به شرط آنکه هر فرایند حداکثر به 2 منبع نیاز داشته باشد، تعداد وضعیت‌های بن‌بست (Deadlock states) در این سیستم حداکثر چند می باشد؟

توضیح: قابل تخلیه پیش هنگام یعنی انحصار متقابل ندارد پس می توان منابع را از آن پس گرفت اصلا بن بست ندارد.

1. 3 حالت
2. 2 حالت
3. 1 حالت
4. 0 حالت

42:00

خنک آن قماربازی که بباخت هر چه بودش و نبود هیچش الا هوس قمار دیگر

57:15

تست ۷ صفحه ۲۳: یک سیستم عامل در حال اجرا،متشکل از چند فرایند همروند را در نظر بگیرید که در حالت بن‌بست نمی باشد. اگر یکی از این فرایند ها در وضعیت بلوکه قرار بگیرند، آیا سیستم عامل می‌تواند تشخیص دهد که این فرایند در انتظار رویدادی است که هرگز اتفاق نخواهد افتاد؟

توضیح: بن بست تنها تشخیصی است که سیستم عامل می تواند درمورد انتظار ابدی بدهد و بجز بن بست هیچ انتظار دیگری رانمی تواند تشخیص دهد مثلاً اگر درون فرایند یک لوپ باشد سیستم عامل نمی‌تواند آن را تشخیص دهد پس جواب در اینجا خیر است. اگر درصورت تست نمی گفت که سیستم درحالت بن بست نمی باشد جواب شاید بود.

1. خیر
2. در سیستم‌های تک کاربره بله ولی در سیستم های چند کاربره خیر
3. در سیستم‌های تک پردازنده ای بله ولی در سیستم های چند پردازنده ای خیر
4. سیستم عامل با کنترل زمان سنج قادر به تشخیص است.

تست۸ صفحه۲۳: سیستمی دارای 5 فرآیند و 4 منبع در حالت زیر قرار دارد:

Alloc

فرایند	R0	R1	R2	R3
P0	3	0	1	1
P1	0	1	0	0
P2	0	1	1	0
P3	1	1	0	1

Need:

فرایند	R0	R1	R2	R3
P0	1	1	0	0
P1	2	1	1	2
P2	3	1	0	0
P3	0	0	1	0

منابع موجود E=(5342)

این سیستم در چه وضعیتی است؟

- امن
- نا امن
- غیرممکن
- بن بست

توضیح: اول منابع تخصیص یافته را با هم جمع می‌زنیم می شود:

P: 4 3 2 2

سپس از کل منابع کم می کنیم:

E - P = (5 3 4 2) – (4 3 2 2) = (1 0 2 0)

بعد از آن خواهیم دید که P3 قابل اجراست و بعد از آن P0 , P4 قابل اجراست و در نهایت امن است.

1:02:10

تست۹صفحه۲۴: سیستمی دارای 5 فرآیند و 4 منبع در حالت زیر قرار دارد:

Alloc

فرایند	R1	R2	R3	R4
P0	3	0	1	1
P1	0	1	0	0
P2	1	1	1	0
P3	1	1	0	1

Need:

فرایند	R1	R2	R3	R4
P0	1	1	0	0
P1	0	1	1	2
P2	3	1	0	0
P3	0	0	1	0

منابع موجود E=(6342)

در چه صورتی شرایط نا امن است؟

توضیح: روش تستی:

اول اینکه می‌دانیم وضعیت امن است چرا که اگر نا امن بود تمام مسیر های زیر نیز نا امن بودند. سر مسیر امن را پیدا می‌کنیم که در اینجا P3 است.باید در گزینه ها گزینه ای باشد که مسیر امن را نا امن کند.

1. فرآیند P0 یک واحد از منبع R2 را درخواست کند.
2. فرآیند P1 یک واحد از منبع R2 را درخواست کند و فرایند P3 آخرین واحد باقیمانده R2 را درخواست کند.
3. فرآیند P1 یک واحد از منبع R2 را درخواست کند و فرایند P4 آخرین واحد باقیمانده R2 را درخواست کند.
4. فرآیند P1 یک واحد از منبع R2 را درخواست کندو فرایند P4 کلیه منابع مورد نیازش را درخواست کند.

تست۱۰صفحه۲۴: سیستمی متشکل از 4 فرآیند P1 , P2 , P3 , P4 و سه نوع منبع قابل استفاده مجدد , S1 , S2 S3 را در نظر بگیرید. تعداد واحد های هر منبع به ترتیب t1=3 , t2=2 , t3=2 است. P1 یک واحد از منبع S1 را در اختیار دارد و تقاضای یک واحد از S2 را کرده است. P2 دو واحد از منبع S2 را در اختیار دارد و تقاضای یک واحد از S1 و یک واحد S3 را کرده است. P3 یک واحد از منبع S1 را در اختیار دارد و تقاضای یک واحد از S2 را کرده است. P4 نیز دو واحد از منبع S3 را در اختیار دارد و تقاضای یک واحد از S1 را نموده است.

1. امن است.
2. نا امن است.
3. بن‌بست رخ داده است.
4. بن‌بست رخ نداده است.

توضیح:

توجه کنید که در اینجا Request ها را داریم و need را نداریم پس نمی‌توانیم پس در مورد امن یا نا امن بودن نظری بدهیم پس دو گزینه حذف می‌شود یعنی می‌توانیم فقط بن‌بست بودن یا نبودن مسأله را بحث کنیم.

Alloc

فرایند	R1	R2	R3
P1	1	0	0
P2	0	2	0
P3	1	0	0

Need:

فرایند	R1	R2	R3
P1	0	1	0
P2	1	0	1
P3	0	1	0

P = 2 2 2

A= 1 0 0

E = 3 2 2

P4 ابتدای مسیر امن است. بعد از اجرای P4 منابع آزاد (102)خواهد شد و پس از آن میس توانیم P2 را اجرا کنیم . پس از اجرای آن منابع آزاد (122) خواهد شد که پس از آن مابقی فرآیند ها قابل اجرا و بن‌بست رخ نخواهد داد.

تست۱۱صفحه۲۵: سیستمی دارای 5 فرآیند و 4 منبع در حالت زیر قرار دارد:

Alloc

فرایند	R1	R2	R3	R4
P0	3	0	1	1
P1	0	1	0	0
P2	1	0	1	0
P3	0	1	0	1

Need:

فرایند	R1	R2	R3	R4
P0	0	1	2	0
P1	2	1	1	2
P2	3	1	0	0
P3	1	0	1	2

E=(5342) کل موجودی اولیه

این سیستم در چه وضعیتی است؟

1. امن
2. نا امن
3. بن‌بست
4. هیچکدام

توضیح:

$P4 \rightarrow P0 \rightarrow P2 \rightarrow$ دیگر نمی‌توان ادامه داد پس نا امن است

جلسه دوازدهم پنجشنبه ۱۳۸۸/۷/۲ ساعت ۸:۰۰

فصل چهارم :

مدیریت حافظه

حافظه سلسله مراتبی از حافظه های مختلف است:

موقتی :

- رجیسترها
- حافظه نهان داخلی
- حافظه نهان خارجی
- حافظه اصلی
- بافردیسک

پایدار و ماندگار:

- Disk
- Type

در هر دو دسته بالا از بالا به پایین سرعت کمتر و ظرفیت بیشتر می شود.

اگرچه حافظه هفت لایه دارد اما ما بیشتر با دولایه سر و کار داریم و هر موقع می گوییم حافظه منظورمان حافظه اصلی است. حافظه نماد حافظه های سریع بالایی است ولایه دیگری که با آن بیشتر سروکارداریم دیسک است که نماینده لایه های کند پایینی است.

هدف از مدیریت حافظه:

استفاده بهینه وبی خطر ازحافظه یعنی حافظه هدرنرود و نباید به حریم یکدیگر دسترسی داشته باشیم.

انواع سیستم های مدیریت حافظه :

تکنیک های دسته اول : در این دسته برنامه باید کامل ویکجا درحافظه بار شود تا برنامه بتواند اجرا شود

1. تک برنامگی
2. چند برنامگی با پارتیشن های ثابت (ایستا) : تکه تکه شدن داخلی خیلی شدید – محدودیت تعداد برنامه‌ها – بزرگترین فرآیند اندازه بزرگترین پارتیشن
3. چند برنامگی با پارتیشن های متغیر یا پویا : به آن تکنیک مبادله یا swapping نیز می گویند. تکه تکه شدن خارجی شدید است.

تکنیک های دسته دوم: در این دسته برنامه باید کامل اما می تواند پراکنده درحافظه بازشود.

1. صفحه بندی (Paging): تکه تکه شدن داخلی ـ ناچیز (قابل صرف نظر کردن)
2. قطعه بندی (Segmentation): تکه‌تکه شدن خارجی ـ متوسط
3. ترکیب صفحه بندی با قطعه بندی (Virtual memory): تکه‌تکه شدن داخلی ـ خیلی کم .

تکنیک سوم : برنامه می تواند ناقص و پراکنده در حافظه بار واجراشود.

حافظه مجازی : مدرن ترین و پیچیده ترین تکنیک است.

می‌توانیم بعضی از قطعات و صفحات را به حافظه بیاوریم و لازم نیست کل صفحات را لود کنیم. مجبور هم نیستیم صفحاتی را که آورده‌ایم با ترتیب خاصی در حافظه بچینیم. این تکنیک باعث صرفه جویی در حافظه می شود. درجه چند برنامگی را افزایش خواهد داد و امکان اجرای فرآیند های بزرگتر از حافظه را خواهد داد.

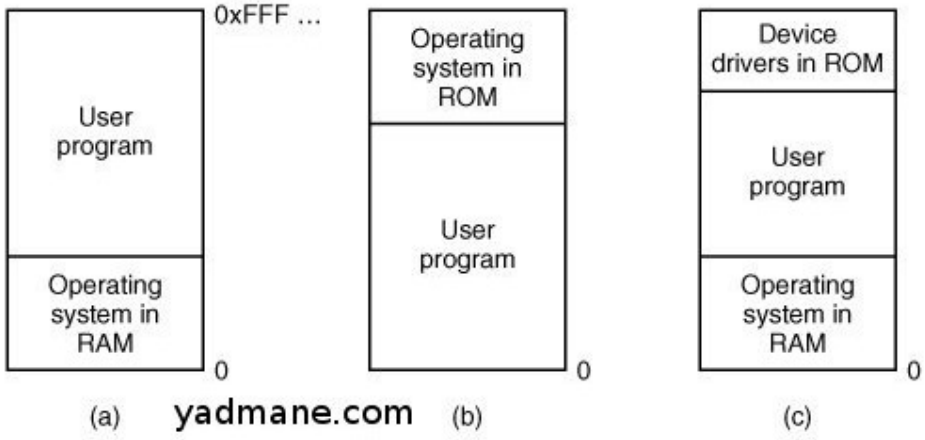
سیستم‌های مدیریت حافظه - دسته اول - تک برنامگی: درواقع تکنیک نیست و بدترین استفاده ممکن از حافظه است.

درتک برنامگی یک برنامه وارد حافظه می شود ، اجرا می شود وخارج می شود وبعد از آن برنامه بعدی می آید.

صفحه 445 کتاب:

در شکل زیر سه روش ساده سازماندهی حافظه شامل یک سیستم عامل و یک فرایند کاربر نمایش داده شده:

شکل c : شبیه سیستم DOS است.



34:30

سیستم‌های مدیریت حافظه - دسته اول - چند برنامگی با پارتیشن های ثابت (ایستا)

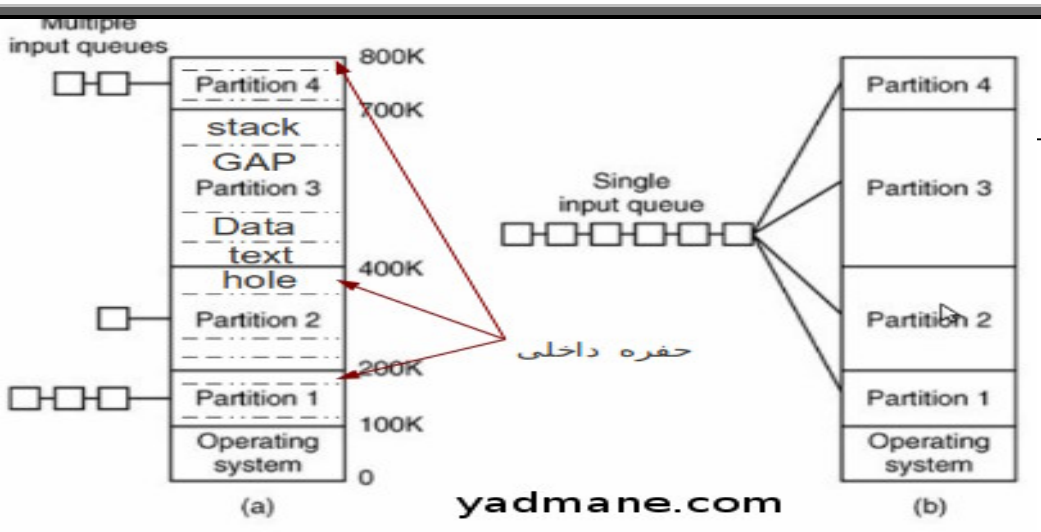
در این تکنیک حافظه توسط Supervisor توسط نرم افزارهای خاص به اندازه های مناسب پارتیشن بندی می شود.

این تکنیک اکنون منسوخ شده و اکنون حافظه پارتیشن بندی نمی شود.

درهر پارتیشن حداکثر یک برنامه می تواند قرار بگیردحتی اگر جا برای یک فرایند دیگر هم باشد.

حتی در دو پارتیشن مجاور نیز نمی‌توان یک برنامه را قرار داد و حتماً یک برنامه باید در یک پارتیشن قرار بگیرد.

- در شکل زیر hole فضا های هدر رفته می‌باشند یعنی فرض کنید پارتیشن 10k باشد در صورتی که برنامه 6k است فضای اضافی به عنوان hole هدر می‌رود به این مشکل که حافظه پر از حفره های بی مصرف می‌شود fragmentation (تکه تکه شدن) می‌گویند.
- چون حفره ها داخل فضایی است که به فرآیند تخصیص می‌دهیم به این نوع تکه‌تکه شدن، تکه‌تکه شدن داخلی



می گویند.

تکه تکه شدن (پارگی) Fragmentation

داخلی : حفره داخل فضایی است که به فرایند تخصیص داده ایم .

خارجی : حفره خارج ازفضایی است که به فرایند تخصیص داده ایم.

- GAP فضای مفید داخل برنامه است که برای رشد استک استفاده می‌شود.
- توجه hole و GAP با هم تفاوت دارند و hole در واقع هدر می رود.

معایب چند برنامگی با پارتیشن های ثابت:

1. تکه تکه شدن داخلی خیلی شدید است
2. محدودیت تعداد برنامه ها
3. اندازه بزرگترین فرایند محدود به سایز بزرگترین پاتریشن می‌باشد.

- اما به هر حال از تک برنامگی بهتر است چرا که در آن واحد چند برنامه می‌تواند اجرا می شود.

روش‌های قراردادن فرایند درحافظه :

1. فرایند ها برحسب اندازه درصف های مجزا قراردهیم .شکل a
2. فرایند ها درصف واحد قراربگیرند. شکل b

شکل a توازن بار ندارد یعنی ممکن است درخواست ها برای پارتیشن های کوچک زیاد باشد.

ایراد b این است که یک فرایند کوچک یک پارتیشن بزرگ را می گیرد وفرایند های بزرگ محروم می شوند. بهتر است شکل b را کمی تغییر دهیم به گونه‌ای که هر پارتیشنی که خالی می‌شود بزرگترین فرایندی را که به آن می‌خورد را به آن بدهیم . در این صورت برای فرایند های کوچک (کوچک نسبت به هر پارتیشن) مشکل قحطی پیش می‌آید. برای رفع این

مشکل باید برای فرایند ها یک شمارنده بگذاریم که هر بار نوبت فرایندی بود اما به دلیل اینکه فرایند بزرگتری وجود داشته که می‌توانسته در آن پارتیشن قرار بگیرد، نوبتش رعایت نشده یکی به شمارنده آن اضافه می کنیم ، هر گاه شمارنده از عددی خاص بیشتر شد اولین پارتیشنی که خالی شد را به آن فرایند بدهیم.

مشکل جابجایی

Relocation Problem

مشکل این جاست که هربارکه یک برنامه را می خوانیم برنامه در آدرس جدیدی Load می شود وحتی در یک بار اجرای یک برنامه ، بارها از حافظه خارج و دوباره درجای جدیدی ازحافظه Load می شود در واقع آدرس base عوض می شود.

یعنی آدرس‌های ما در داخل برنامه آدرس نسبی یا منطقی هستند و این آدرس‌ها باید با آدری base جمع شوند تا آدرس حقیقی یا فیزیکی بدست آید.

1:01:45

مشکل حفاظت

Protection Problem

یک برنامه نباید به آدرس های برنامه دیگردسترسی داشته باشد.

روش حل مشکل جابجایی : تمام آدرس های درون برنامه را با آدرس base جمع می کند . این کار در زمان load برنامه به حافظه توسط سیستم عامل صورت می گیرد. دقت کنید که سیستم عامل نباید تک تک خطوط برنامه را با آدرس base جمع کند بلکه فقط آدرس‌ها را باید با base جمع کند به همین خاطر سیستم عامل باید بداند کجا آدرس و کجا دستورالعمل است . برای این منظوردر انتهای عمل کامپایل، لینکر عما پیوند یا لینک را انجام می‌دهد و جدول لینک ها در header برنامه قرار می گیرد. (این روش امروزه منسوخ شده است)

ایرادات این روش:

- این کار وقت گیراست چون در یک برنامه بزرگ تعداد آدرس‌ها بسیار زیاد است.
- مشکل دوم این است که محل آدرس ها در هر بار اجرا نیز مرتبا تغییر می‌کند و دوباره باید آدرس ها محاسبه شود.
- وابسته بودن سیستم عامل به header فایل.

1:15 :50

پیشنهاد IBM برای حل مشکل حفاظت

حافظه را به بلوک هایی تقسیم می کنند و برای هر بلاک یک کد حفاظتی درنظر می گیرند.

در شکل زیر قسمت‌های همرنگ بلاک های در اختیار یک برنامه است که کد حفاظتی یکسان دارند و هر برنامه یک کلید چهار بیتی دارد

سیستم عامل جداول را آماده می کند واین CPU است که جلوی خطاها را می گیرد.

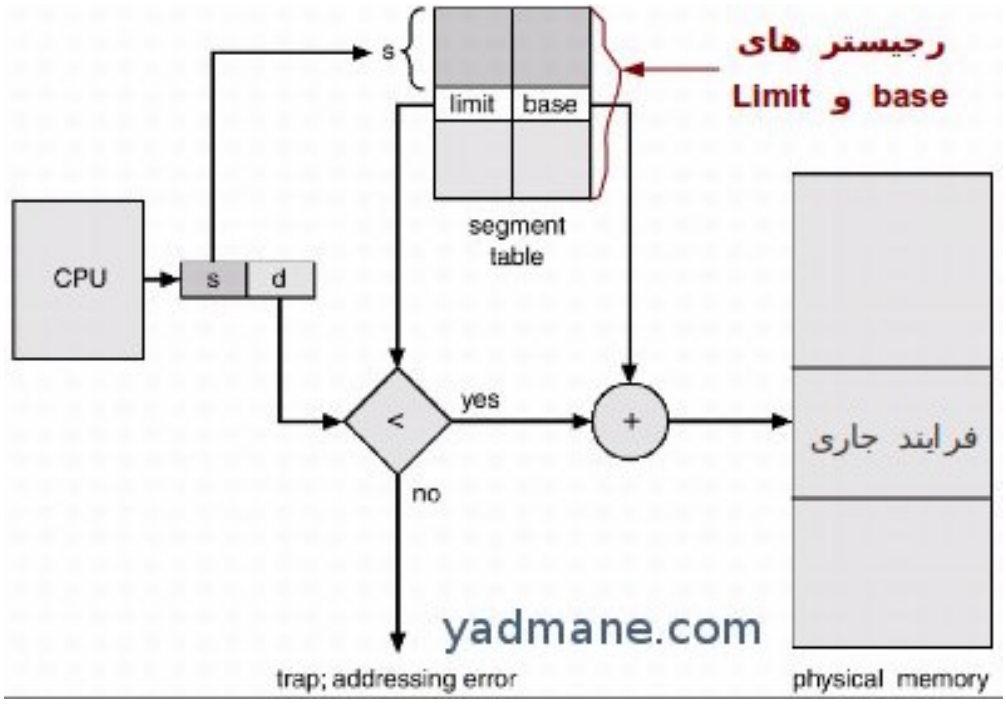
این تکنیک پیچیده است همچنین چون 4 بیت برای آن استفاده شده حداکثر 16 برنامه می تواند همزمان اجرا شود.

1:20 :30

1:35:30

روش رجیسترهای حد و پایه

Base And Limit Register



این روش برای رفع مشکل جابجایی وحفاظت ارائه شده است.

Limit= اندازه (طول) برنامه است .

Exception= وقفه نرم افزار از نوع Program check

وظایف CPU :

1. تبدیل آدرس منطقی به فیزیکی : آدرس نسبی را با آدرس Base جمع می کند (سخت افزاری است) و عدد حاصل را روی Address Bus می گذارد
2. آدرس نسبی را با Limit مقایسه می کند و در صورتی که در بازه نباشد خطا (Protection Fault) رخ می دهد.

Protection Fault وقفه نرم افزاری از نوع Exception یا program check می‌باشد که با وقوع آن در واقع یک تله رخ می‌دهد و باید وارد مد هسته شده ، سیستم عامل Exception handler یا اداره کننده Exception را برای رسیدگی به آن فراخوانی می کند.

- مقادیرBase و Limit را سیستم عامل هنگام Load درPCB قرارمی دهد. وهنگام Dispatch آنها را ازPCB فرایند می خواند و در رجیسترها قرار می دهد.
- مقایسه کننده وLimit مشکل حفاظت را حل کرد.
- Base و جمع کننده مشکل جابجایی را حل کرد.

تست- سیستم عامل چه زمانی مقادیر base و limit را مقداردهی می کند:

1. load
2. اجرای دستورالعمل
3. Dispatch
4. پیوند

سؤال- هر کدام از سخت‌افزارهای زیر را چه کسی مدیریت می کند؟

Disk ⟶ سیستم عامل

Cache ⟶ سخت افزار

Registers ⟶ برنامه کاربر در زبان اسمبلی و کامپایلر در زبان‌های سطح بالا

RAM ⟶ سخت‌افزار و سیستم عامل

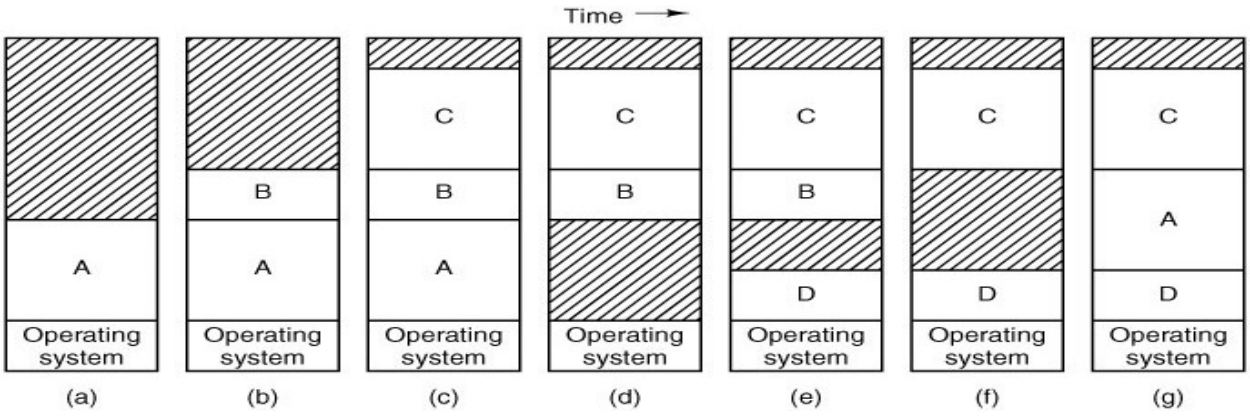
2:00:50

سیستم‌های مدیریت حافظه - دسته اول - روش Swapping - مبادله - پارتیشن بندی پویا یا پارتیشن بندی متغیر

دراین روش در واقع دیگر پارتیشن نداریم و یک برنامه می تواند درصورت وجود حافظه، به اندازه سایز خود حافظه را اشغال کند نکته اینکه این فضای حافظه می بایست پیوسته باشد. پس از خروج فرایند فضای مصرفی آن آزاد می‌شود و فرایند های دیگر می‌توانند جایگزین آن شوند اما به دلیل اینکه سایز فرایند ها یکسان نیست ممکن است در این جابجایی ها فضا های خالی کوچکی ایجاد شود که از کوچکترین فرآیند ما کوچکتر باشد و بنابر این بلا استفاده می‌ماند که به آن حفره می گویند.

به این حفره ها چون خارج از فضای فرایند است تکه‌تکه شدن خارجی می گویند.

یک تفاوت دیگر تکه‌تکه شدن داخلی و خارجی در این است که این حفره ها در خارج از محیط فرایند هستند پس اگر فرایند کوچکی وجود داشته باشد که در آنجا شود می‌توان از این فضا استفاده کرد اما در حفره های داخلی هر چقدر هم که بزرگ باشند چون داخل محیط فرایند هستند نمی‌توان استفاده کرد. به همین خاطر مشکل حفره های داخلی می‌تواند حاد تر باشد چرا که حفره های آن می‌تواند **بزرگتر** باشد.



معایب Swapping :

- تکه تکه شدن خارجی
- عدم امکان اجرای برنامه های بزرگ‌تر از حافظه
- درجه چند برنامگی درحد مطلوب نیست. (به دلیل اینکه برنامه‌ها باید کامل در حافظه بار شوند)
- کل فرایند لازم است درحافظه قرارگیرد درحالی که کل فرایند برای اجرای برنامه لازم نیست.

برای رفع مشکل تکه تکه شدن خارجی Memory Compaction یا Memory Defragmentation پیشنهاد شده است . دراین روش فضاهای پرشیفت داده می شوند تا فضاهای خالی کنار هم قرارگیرند اما این تکنیک وقت گیراست و از آن استفاده نمی شود.

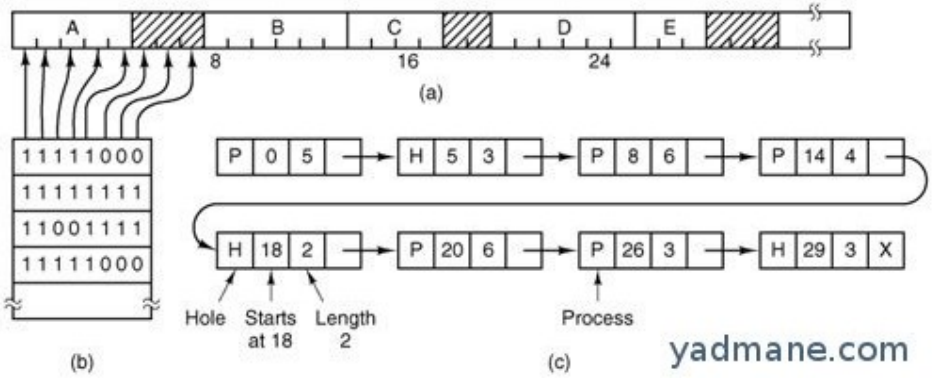
درصفحه 451 مثالی برای نقض این روش آورده شده است.

مشکل برق2ابتدای صدای 2:33:00

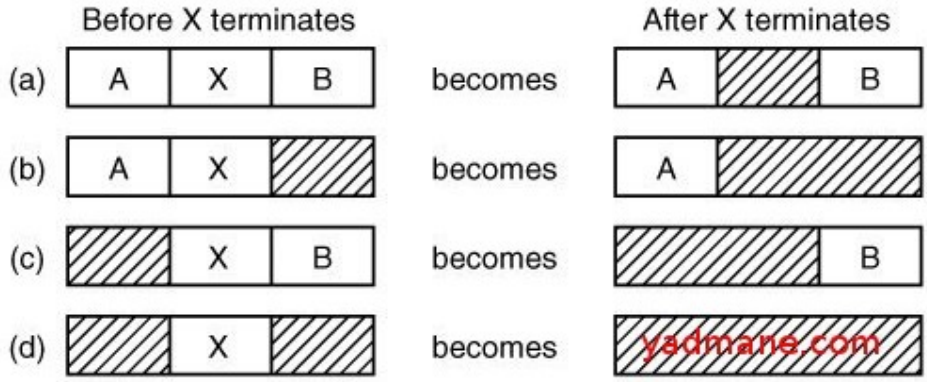
سیستم عامل ازکجا می فهمد کجای حافظه پراست وکجا خالی است ؟ کجا فرایندها هستند ؟ کجا حفره ها ؟

تکنیک های مدیریت حافظه :

1) Bitmap
2) Linked List



- در شکل a حافظه را می‌بینیم، محل حفره ها و فرآیند ها.
- در شکل b روش مدیریت حافظه bitmap نشان داده شده است ، ایراد این روش این است که برای پیدا کردن حفره خالی به اندازه مورد نیاز باید از ابتدای حافظه تعداد صفر های پشت سر هم را که نشان دهنده بلاک های خالی است بشماریم تا حفره‌ای به اندازه مورد نیاز پیدا کنیم که این کار وقت گیراست و یکی از ایرادات این روش است . یعنی باید ازابتدای حافظه شروع به شمارش صفرها کنیم .
- در شکل c یا روش LL تنها کافی است که دنبال hole به اندازه مورد نیاز باشیم ودیگرنیاز به شمارش نیست . دراین روش اگر یک پروسس ازحافظه بیرون رود و درکنار آن حفره باشد باید با حفره های مجاور آن ترکیب شود.



سؤال- فرض کنید از روش bitmap استفاده می‌کنیم و هر بلاک 10 بایت است ، سربار bitmap چند بایت است؟

10 byte * 8 = 80 byte

(1 / 81) * 100

(1 / (80 + 1)) * 100 %

توضیح:

10 بایت 80 بیت می‌شود . یک بیت اضافه هم برای bitmap استفاده می‌شود پس به ازای هر 81 بیت یک بیت سربار است یعنی 1/81 1م سربار bitmap است.

19:00

تکنیک ها یا الگوریتم های تخصیص حافظه

این تکنیک ها عبارتند از:

- First Fit
- Next Fit
- Best Fit
- Worst Fit
- Quick Fit

صورت مسئله: یک فرایند را می خواهیم درحافظه Load کنیم . تعدادی حفره داریم ، فرایند را در کدام حفره بگذاریم بهتراست.

First Fit :

ازابتدای حافظه شروع کن وفرایند را در اولین حفره ای که جا می شود قرار بده .

نکته : در اینجا fit بودن به معنی جا شدن است یعنی بزرگ‌تر از سایز مورد نیاز ماست و معنی fit بودن که در سایز لباس از آن استفاده می‌کنیم را نمی دهد.

مزیت : سرعت بالای آن

عیب : از دید استفاده کارآمد ازحافظه خیلی مطلوب نیست.

مثال : لیست حفره های درون حافظه به ترتیب از چپ به راست عبارتند از:

15k , 10k , 45k , 5k , 8k , 70k , 30k

چهارفرایند به ترتیب با اندازه های 10k , 8k , 12k , 20k از چپ به راست وارد می شوند اگراز الگوریتم First Fit استفاده کنیم لیست جدید حفره ها کدامند.

15	10	45	5	8	70	30
		-20				
15	10	25	5	8	70	30
-12						
3	10	25	5	8	70	30
	-8					
3	2	25	5	8	70	30
		-10				
3	2	15	5	8	70	30

Next Fit :

Next Fit مثل First Fit است اما هر بار به جای اینکه از اول حافظه دنبال فضای خالی بگردد از آخرین تخصیص دنبال حافظه خالی می گردد

مثال : مثال بالا را با الگوریتم Next Fit حل کنید فرض کنید درابتدای کار اشاره گر به اولین حفره پایین حافظه اشاره می کند.

15	**10**	**45**	**5**	**8**	**70**	**30**
		-20				
15	**10**	**25**	**5**	**8**	**70**	**30**
		-12				
15	**10**	**13**	**5**	**8**	**70**	**30**
		-8				
15	**10**	**5**	**5**	**8**	**70**	**30**
					-10	
15	**10**	**5**	**5**	**8**	**60**	**30**

نکته: وقتی حفره ای را به فرایندی اختصاص می‌دهیم اشاره گر در مرحله بعد به حفره جدید ایجاد شده حاصل از باقیمانده حفره قبلی اشاره می کند.

در بدترین حالت یک دورمی چرخد وبرمی گردد به ابتدای حفره های خالی تا جایی که به همان حافظه خالی قبلی برسیم که وقت گیر است . اگرهم که یک دور زد و حفره خالی پیدا نشد که در واقع حفره مناسب نداریم.

: Best Fit

این الگوریتم فرایند را در کوچکترین حفره ای که جا می شود قرارمی دهد (کل لیست را جستجو می کند) .

15	**10**	**45**	**5**	**8**	**70**	**30**
						-20
15	**10**	**45**	**5**	**8**	**70**	**10**
-12						
3	**10**	**45**	**5**	**8**	**70**	**10**
				-8		
3	**10**	**45**	**5**	**0**	**70**	**10**
	-10					
3	**0**	**45**	**5**	**0**	**70**	**10**

41:00

مزیت : استفاده کار آمد از حافظه

ایراد:

1. به علت عملیات جستجو کند است .
2. حافظه را پراز حفره های ریز بی مصرف می کند.

: Worst Fit

دراین الگوریتم فرایند را دربزرگترین حفره موجود قرارمی دهیم . (البته اگر جا بشود)

15	10	45	5	8	70	30
					-20	
15	10	45	5	8	50	30
					-12	
15	10	45	5	8	38	30
		-8				
15	10	37	5	8	38	30
					-10	
15	10	37	5	8	28	30

مقایسه الگوریتم های تخصیص حافظه :

- First Fit و Next Fit هر دو از لحاظ سرعت خوب هستند.
- First Fit از لحاظ استفاده کارآمد از حافظه خیلی خوب نیست.
- 51:50
- ایده Next Fit این بوده که توزیع یکنواختی داشته باشیم تا ابتدای حافظه شلوغ نباشد و جستجو سریعتر باشد.
- ایراد Next Fit این است که ممکن است حفره های بزرگ آخر لیست را از بین ببرد و جا برای فرآیند های بزرگ نباشد.
- درشبیه سازی First Fit بهترازNext Fit است.
- مزیتBest Fit استفاده کارآمد از حافظه است.
- Best Fit و Worst Fit هردو کند هستند زیرا نیاز به جستجو دارند.
- ایراد دیگرBest Fit حافظه را پر از حفره های ریز بی مصرف می کند.
- کندترین این الگوریتم ها Best است هر چند از نظر مرتبه زمانی چنین نتیجه‌ای حاصل نمی شود.
- در نهایت اینکه از نظر استفاده خوب از حافظه و هم از نظر سرعت First Fit از بقیه بهتر به نظر می رسد.
- Quick Fit بین Best Fit و First Fit قراردارد.

بارش باران 1:05:20

گویند بهشت و حور عین خواهد بود آنجا می ناب و انگبین خواهد بود گر ما می و معشوق پرستیم رواست چون عاقبت کار همین خواهد بود

1:09:20

Quick Fit :

Quick Fit همان Best Fit است که سریع انجام می شود. اگرسوالی دادند که گفتند با Quick پاسخ دهید همان پاسخ Best را می دهد. چون همان ایده را دارد. در این روش لیست حفره ها طبقه بندی می شود مثلا حفره های 1تا 4 کیلو بایت ، حفره های 5 تا 8 کیلو بایت و ... (به جای یک لیست n لیست داریم) درواقع نوعی طبقه بندی سایز می باشد.

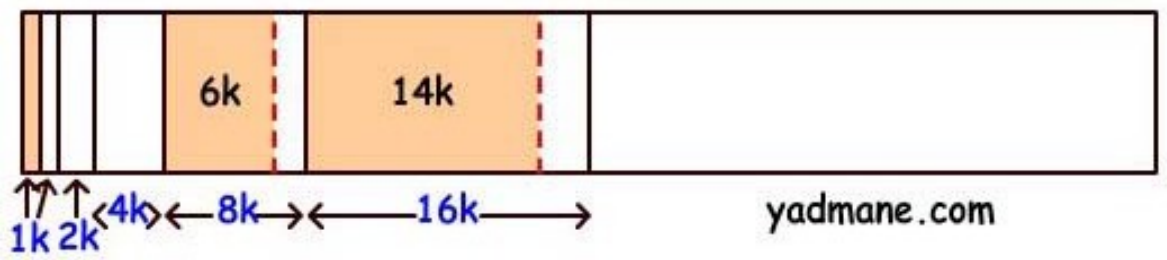
ایراد این روش موقع خروج ازحافظه است خصوصا وقتی که سایز حفره های کناری هر کدام دریک لیست است حال برای تلفیق حفره های کناری باید کل لیست ها را بررسی کنیم.

1:16:30

Buddy System (سیستم رفاقتی) :

با روش های قبلی متفاوت است . به گونه ای عمل می کند که همیشه حفره ها توانی از 2 باشند ، به همین دلیل مرتبا حفره ها را نصف می کند تا همیشه توانی از 2 باقی بماند. (درمسیرBest Fit حرکت می کند)

- درمسیرBest Fit حرکت می کند و می خواهد مثل Best Fit ازحافظه بهینه استفاده کند اما ازنظربهینگی به Best Fit نمی رسد اما از نظر سرعت از آن بهتراست ، چون چیدمان حفره ها ساخت یافته است سرعت یافتن حفره ها درآن زیاد است .



اگر یک خارج شود با یک کناری است ترکیب می‌شود، چون 2 کناری آن نیز خالی است با آن نیز ترکیب می‌شود ...

جلسه سیزدهم دوشنبه ۸۸/۷/۶ ساعت ۱۰:۰۰

رسم شکل 29:50

Overlay چیست ؟

تکنیکی است که بتوان درسیستم های قدیمی برنامه های بزرگتر ازحافظه را اجرا کرد. برنامه نویس باید برنامه را به چند قسمت تقسیم کند و هرگاه نیاز به قسمت هایی شد که درحافظه وجود ندارد خود برنامه با یک فراخوان سیستمی از سیستم عامل می خواهد آن را از حافظه خارج و قطعه دیگری از برنامه را بیاورد.

این کار، کارخوبی نیست چون برنامه و فضای کاربر را درگیر مدیریت حافظه می کند و سیستم عامل نقش ضعیفی در مدیریت حافظه دارد.

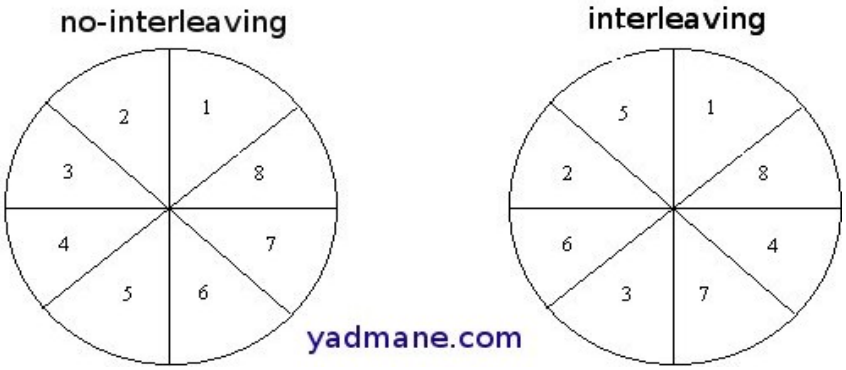
صفحه بندی

1. ساده
2. حافظه مجازی.

- صفحه بندی ساده قصد دارد Fragmentation را مینیمم کند.
- درصفحه بندی ساده تمام صفحات فرایند راباید درحافظه اصلی بیاوریم . اما درمجازی صفحاتی راکه نیاز داریم درحافظه اصلی قرارمی دهیم . وبقیه روی دیسک می ماند.
- صفحه بندی ساده ← آدرس منطقی
- صفحه بندی حافظه مجازی ← آدرس مجازی
- مشکل Fragmentation در هر دو روش حل می شود.
- درجه چند برنامگی : درحافظه مجازی درجه چند برنامگی بالاتر است.
- درحافظه مجازی به راحتی می توانیم فرایند های بزرگتر ازحافظه را اجرا کنیم .
- End استفاده ازحافظه paging حافظه مجازی است.

۴۳:۳۰

Interleaving : (نکته جا مانده از مبحث دیسک)



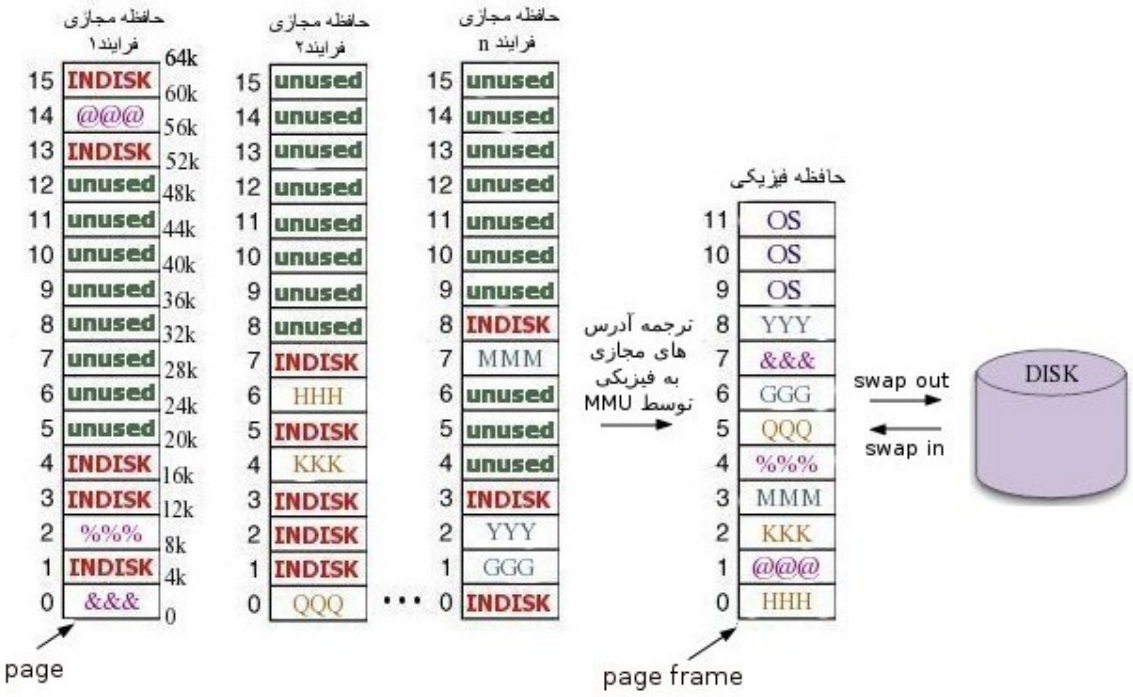
درهنگام خواندن ازدیسک اگرسگمنت های یک برنامه پشت سرهم باشد وقتی یک سگمنت را می خواند تا بخواهد آن را ازدیسک به Cache منتقل کند هد از سگمنت بعدی گذشته است وبرای خواندن سگمنت بعدی باید دیسک یک دور کامل بگردد تا دوباره به آن سگمنت برسد. برای حل این مشکل پیشنهاد شده است قطعات یک برنامه با فاصله در دیسک قراربگیرد.

Interleaving تکنیکی است برای افزایش کارایی دیسک.

ادامه مبحث صفحه بندی:

تفاوت page و segment :

PAGE	SEGMENT
اندازه یکسان	**اندازه متفاوت**
اندازه ثابت	**اندازه متغیر (پویاست)**
اندازه کوچک مثلاً بین 0/5 تا 10 کیلوبایت	**می‌تواند تا چند مگابایت باشد**
چون اندازه صفحات کوچک است تعدادشان زیاد است	**معمولاً تعداد کمی وجود دارد**
از دید برنامه نویسان پنهان هستند	**توسط خود برنامه نویس تعریف می شوند. مدیریت آن‌ها مانند صفحات توسط سیستم عامل است و با overlay فرق می کند.**
در بهینگی است	**در اشتراک زمانی عالی است**



سمت چپ عکس بالا فضای مجازی و سمت راست فضای فیزیکی است.

- هر page می‌تواند used یا unused باشد.
- صفحات used می‌توانند بر روی Disk باشند (in Disk) یا بر روز حافظه (in Memory)
- هرفرایند یک Virtual Memory دارد.
- هرکدام از برنامه ها یک فضای بی نهایت برای خودش درنظر می گیرد. یعنی تا جایی که CPU اجازه می‌دهد که در اینجا چون آدرس‌ها 16 بیتی است بزرگترین آدرس 2 به توان 16 می شود. که این فضا حتی از کل حافظه فیزیکی ما بیشتر است.
- هر برنامه هرمقدار که بخواهد می تواند از این فضا استفاده می کند. مابقی آن نیز که استفاده نمی شود هدر نمی رود چون مجازی است . ضمن اینکه برای صرفه جویی در حافظه سیستم عامل فقط صفحات لازم را به حافظه اصلی می برد و مابقی فرآیند در دیسک و Swap Space می‌ماند تا زمانی که به آن نیاز داشته باشیم.
- روند آوردن صفحات به حافظه تا زمانی که حافظه پر نشده ادامه دارد وقتی که حافظه پر شد و نیاز به حافظه داشتیم باید یکی از صفحات را از حافظه خارج کنیم تا جا برای صفحه جدید باز شود به همین خاطر است که در بین کار با برنامه‌ها مرتبا چراغ هارد خاموش و روشن می شود.

- زمانی که می خواهد یک صفحه را ازحافظه خارج کند باید چک کند اگرروی آن تغییری حاصل شده آن را روی دیسک بنویسد و گرنه نیاز به نوشتن نیست چون سیستم عامل کپی صفحه را زمانی که صفحه از دیسک به حافظه منتقل می‌شد بر روی دیسکنگه داشته و بدین ترتیب یک نوشتن کم می شود.
- هرچند سایز حافظه مجازی محدودیتی ندارد اما سایز یک فرایند از کل فضای دیسک نمی تواند بیشتر باشد.

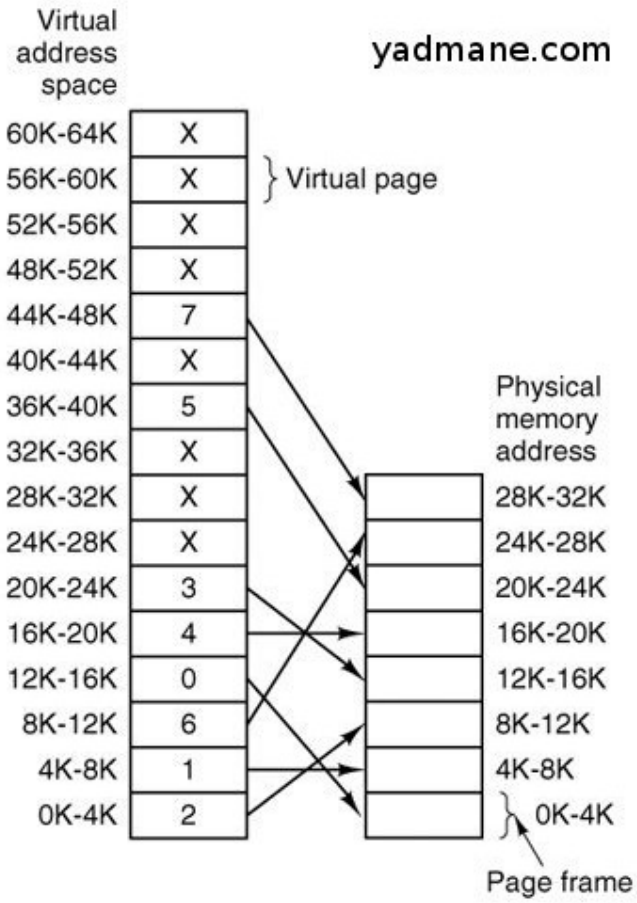
تعاریف همسان :

Physical Address	Virtual Address
Physical Address Space	Virtual Address Space
فضای فیزیکی را به تکه‌های مساوی تقسیم می‌کنیم و نام آن را page frame می نامیم	**فضای مجازی را به تکه‌های یکسان ، نسبتاً کوچک و با اندازه مساوی تقسیم می کنیم ونام آن را page می نامیم**

اگر A = تعدادبیت های آدرس باشد بزرگترین اندازه فرایند ممکن 2 به توان A می شود.

تعداد صفحه = اندازه صفحه / اندازه VM

- لزومی به پرکردن فضای مجازی به صورت مرتب نیست و شما می‌توانید از هر کجای این فضا که خواستید استفاده کنید.

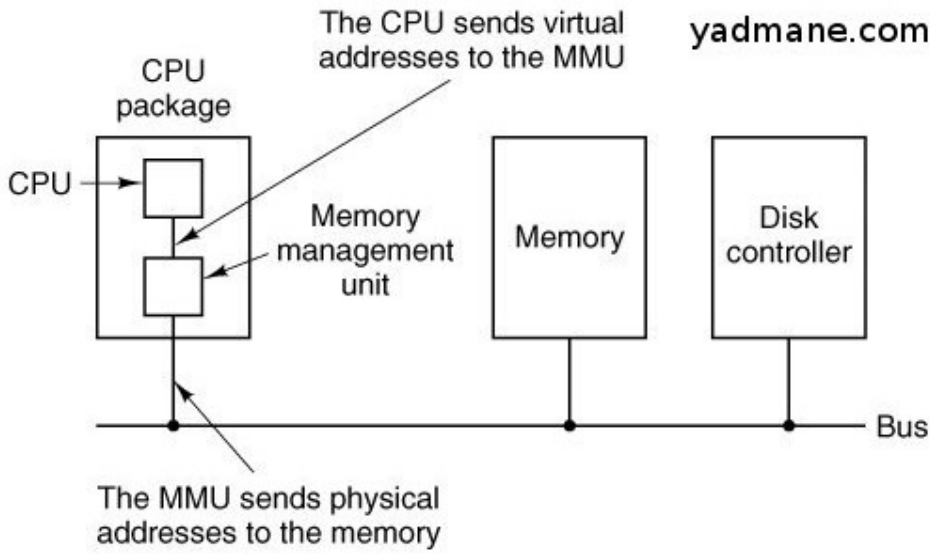


به طورمیانگین بازای هرفرایند نصف صفحه (p/2) خالی می ماند(هدرمی رود) چون تنها صفحه آخرممکن است کامل پرنشود. درواقع Paging مشکل Fragmentation را حل کرده است. ترجمه آدرس ها را سخت افزاری به نام MMU انجام می دهد.

فرض کنید صفحات 4 کیلو بایتی است و فرآیند 30k می‌باشد در این صورت 7 صفحه کامل پر می‌شود و صفحه آخر 2k از آن هدر می‌رود .

این فضای هدر رفته چون داخل فضای فرایند است internal fragmentation می باشد.

آدرس ها توسط سخت افزاری به نام MMU ترجمه می گردد:



مثال- با توجه به شکل دو صفحه قبل آدرس 41FC در فرآیند 2 را به آدرس فیزیکی نگاشت کنید:

P# → 0100111100011100 ← offset

قسمت offset(111100011100) که معادل F1C هگز می‌باشد بدون تغییر منتقل می‌شود.

اما آدرس P# باید با استفاده از جدول صفحه ترجمه گردد که در اینجا 0010 می شود:

PF# → 0010111100011100 ← offset

1:43:30 1:45:40

چرا P# را از سمت چپ جدا کرده ایم:

چون با این کار آدرس‌ها پشت سر هم قرار می گیرند.

چرا آدرس‌ها را به 12 و 4 بیت تقسیم کرده‌ایم:

چون سایز صفحه 4k است می‌شود 2 به توان 12 پس 12 بیت جدا می کنیم.

تعداد صفحات 16 که چهار بیت نیاز دارد

1:55:00

12:10ساعت

ابتدای صدای دوم ساعت 12:30

مثال- آدرس مجازی 8(167243) ازفرایند P1 (در شکل 3 صفحه قبل) را به آدرس فیزیکی ترجمه کنید.

P# → 1110,1110,1010,0011 ← offset

P# = 14 → PF# = 1 → 0001,1110,1010,0011

اگر در مثال بالا صفحه در حافظه مجازی InDisk بود در واقع Page Fault رخ داده بود.

مثال- آدرس مجازی (7F35)هگز یک فرایند را در یک سیستم صفحه بندی با صفحات 2 کیلو بایتی به آدرس فیزیکی ترجمه کنید . بخشی از جدول صفحه به شکل زیر است:

P#	PF#	P/A present / Absent
0	4	0
1	3	1
2	2	0
3	3	0
4	7	1
5	12	1
6	9	1
. . .	. . .	. . .
10	4	1
11	6	1
12	14	1
13	20	1
14	11	1
15	18	1
16	19	1
. . .	. . .	. . .
28	21	1
29	23	1
30	2	0
31	4	0

P# → 01111,1110,0011,0101 ← offset

P# = 15 → PF# = 19

در نتیجه :

P# → 10011,1110,0011,0101 ← offset

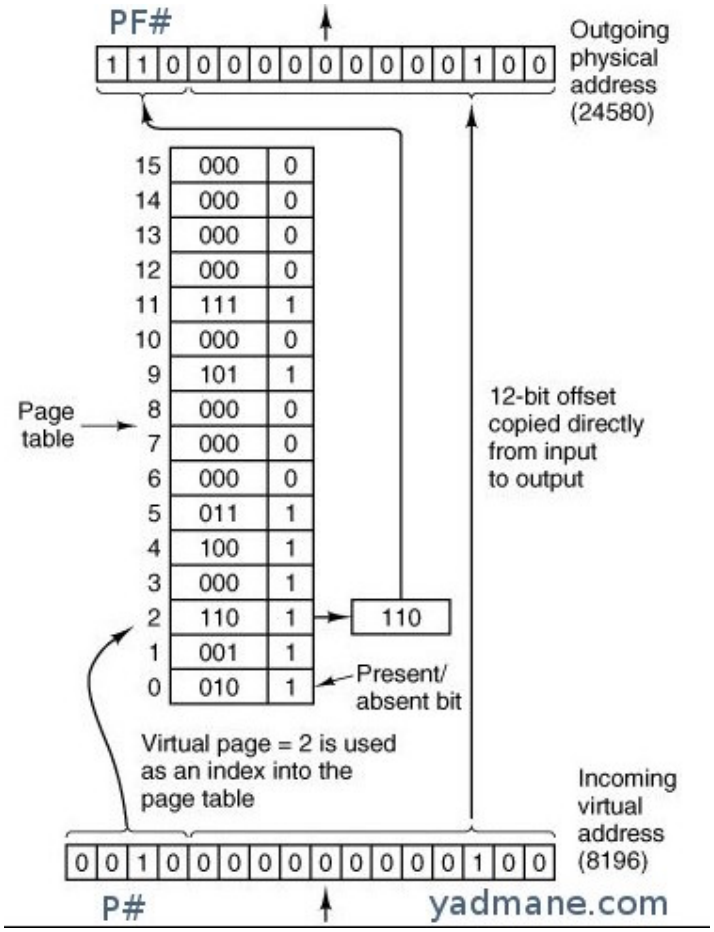
نکته : در جدول مقابل دقت کنید برخی از #PF ها تکراری است که اگر دقت کنید فقط یکی از آن‌ها بیت P/A آن یکی است و بقیه صفر است . پس صفحه در حافظه وجود ندارد و #PF آن بی‌معنی است.

24:30

صفحه 461 کتاب:

در شکل زیر آدرس به دو قسمت تبدیل می‌شود آدرس آفست مستقیم به خروجی ارسال می‌شود و P# توسط MMU به PH# ترجمی می‌گردد که در اینجا P# برابر 2 است به همین دلیل به درایه یا مدخل یا entry شماره 2 جدول صفحه می‌رویم. در درایه 2 اول به بیت حضور و غیاب نگاه می‌کنیم چون بیت 1 است پس page fault رخ نداده پس PF# را بر می‌داریم و به خروجی ارسال می کنیم.

نکته : علت اینکه آدرس آفست تغییر نمی‌کند این است که اندازه page ها با اندازه page frame ها برابر است.



اگربیت حضور و غیاب صفر باشد page fault رخ داده ودیگرکارMMU نیست و وظیفه سیستم عامل است تا به نقص صفحه رسیدگی کند. یعنی اداره نقص صفحه و مراجعه به دیسک کار سیستم عامل است.

32:00

جداول صفحه

بازای هر فرایند یک جدول صفحه داریم. جای جداول صفحه درحافظه است و توسط OS ساخته می‌شود این سیستم عامل است که صفحات را وارد حافظه می‌کند و یا از حافظه خارج می‌کند و این سیستم عامل است که وقتی صفحات را وارد و خارج می‌کند جداول صفحه را update می‌کند اما وقتی برنامه در حال اجراست دیگر سیستم عامل در حال اجرا نیست در این هنگام MMU به صورت سخت افزاری ، با سرعت بالا و اتوماتیک عمل ترجمه آدرس را انجام می دهد.

- توجه کنید که مراجعه به جدول صفحه نیاز به search ندارد چون Index دارد و باعث می شود مستقیم به مدخل صفحه مراجعه کنیم که باعث بالا رفتن سرعت می شود.
- MMU آدرس ابتدای جدول صفحه را در یکی از رجیسترهایش دارد و این آدرس را سیستم عامل هنگام Dispatch در رجیسترها قرار می دهد.
- جداول صفحه انواع مختلف دارند مثل جداول صفحه ساده یا وارونه.

مثال- فرض کنید سیستمی دارای مشخصات زیر است:

Address bus = 32 bit = A

سایز صفحات P = 4KB

هر درایه از جدول صفحه = e = 4Byte

اندازه جدول صفحه را حساب کنید؟

$VM = 2^A = 2^{32} = 4GB$

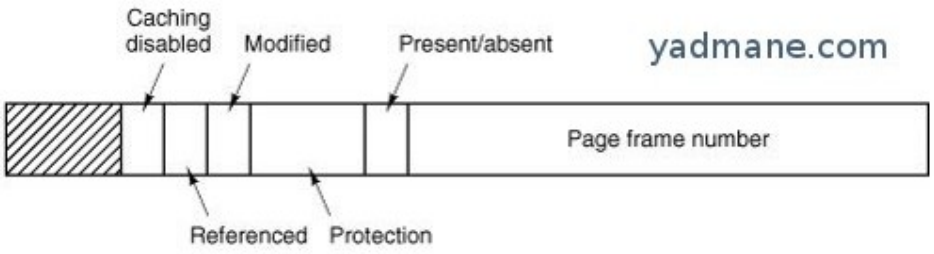
تعداد صفحه = VM / P = $2^{32} / 2^{12} = 2^{20} = 1\ M$ تعداد آرایه

اندازه جدول صفحه = تعداد درایه جدول صفحه * اندازه درایه (e)

$1M * 4B = 4MB = 2^{22}$ بازای هر فرآیند

43:00

ساختار درایه جداول صفحه (صفحه ۴۶۶ کتاب)



بیت حفاظت : از روی این بیت می‌فهمیم صفحه Read only است ،قابل اجراست ، قابل نوشتن است و یا ترکیبی از این خواص را دارد.

بیت M (تغییر یافته یا Modify) : اگر صفحه‌ای که در حافظه است تغییر پیدا کند این بیت یک می‌شود ودر زمان swap out اگر این بیت صفر باشد از نوشتن اضافی روی Disk جلوگیری می کندو یک نوشتن صرفه جویی می شود یعنی فقط اگر صفحه تغییر یافته نیاز به نوشتن درDisk را دارد.

بیت R (رجوع شده) : اگر به صفحه‌ای در حافظه مراجعه شود بیت R یک می‌شود.

7. این بیت ها توسط MMU به روز رسانی می شوند.

مشکلات Paging :

1 : با توجه به مثال قبل بازای هر فرایند 4MB برای هرجدول صفحه نیاز داریم پس اگر100 فرایند درحافظه داشته باشیم فقط 400MB جداول صفحه می شود و جا برای فرایند ها نمی ماند.

2 : دستور mov از آدرس 10000 به آدرس 2000 چند مراجعه به حافظه دارد؟

3 بار :

- دستور را ازکد بردارد.
- محتوای آدرس 10000 رابردارد .
- به محتوای 2000 بریزد.

در سیستم حافظه مجازی مراجعات 2 برابر می‌شود چرا که یکبار نیز باید MMU به سراغ جدول صفحه برود. ضمن اینکه دستورات متفاوتند و بین 1 تا 5 مراجعه به حافظه دارند.

جمع‌بندی مشکلات Paging :

1. جداول صفحه غول پیکرهستند (ازلحاظ حجم) تازه خوشبختیم که CPU های 64 بیتی نیامده اند.
2. عمل نگاشت باید به سرعت انجام شود اما تعداد مراجعات خیلی زیاد است و حجم زیاد این جداول باعث می‌شود سرعت ترجمه آدرس حتی از سرعت دستورالعمل ها کمتر شود.

1:05:30

چگونه این مشکلات را حل کنیم

پیشنهاد اول : از رجیستر استفاده کنیم : این همه غیرممکن است.

پیشنهاد دوم : صفحات unused را درجدول صفحه نیاوریم : نمی توانیم چون Index ها به هم می‌خورد و در این صورت برای دسترسی به صفحات باید search کنیم که بسیار زمانگیر است. در صورتی که با index دیگر به search نیاز نبود.

پیشنهاد سوم : برای اینکه جداول صفحه کوچک شود صفحات را بزرگ کنیم : تکه تکه شدن داخلی زیاد می شود.

برای کوچک کردن اندازه جدول صفحه دو راه پیشنهاد شده است.

1. جداول چند سطحی
2. جداول صفحه وارونه (معکوس)

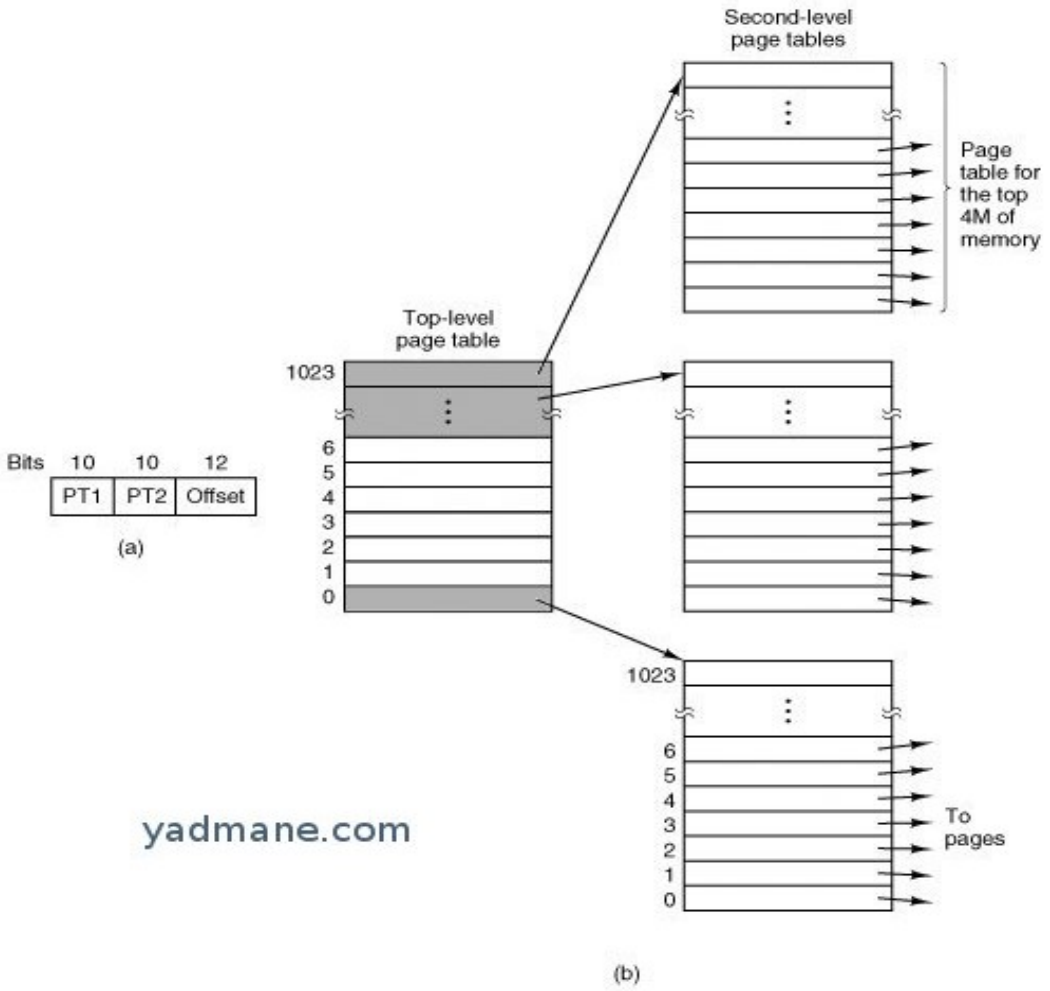
برای افزایش سرعت ترجمه آدرس پیشنهاد زیرداده شده است :

TLB یا Associative Memory

جداول دو سطحی:

در مثال زیر 12 بیت آفست است و 20 بیت برای درایه ها که می‌توانیم 1 میلیون درایه داشته باشیم . درواقع یک جدول داریم با یک میلیون ردیف، آن را تبدیل می‌کنیم به 1024 جدول 1024 ردیفی. البته اجباری در مساوی بودن این اعداد نیست. به هر حال جدول بزرگ را به تعدادی جداول کوچک می شکنیم. که الزامه همه آن‌ها پر نیستند و تعدادی از آن پر می باشد. مثلاً ممکن است در یک فرایند کوچک فقط 3 جدول از 1024 جدول پر شود و 1021 جدول خالی است که نیازی به نگهداری آن‌ها نداریم و در جدول سطح اول درایه معادل آن را خالی می گذاریم.

با این کاربدون اینکه جدوال خالی را نگه داریم Index را به هم نزدیم و جداول را کوچک کردیم.



ابتدای صدای سوم

مثال- فرض کنید آدرس رقم باشد :

11110000111100001111000011110000

اندازه صفحه 4k است پس 12 بیت از راست جدا می‌کنیم به عنوان آدرس آفست

11110000111100001111000011110000

فرض کنید که 1024 جدول 1024 تایی داریم :

پس P# به دو قسمت سطح 1 و سطح 2 تقسیم می‌شود :

11110000111100001111000011110000

پس با توجه به مقدار اندیس سطح 1 که 1111000011 است باید به جدول متناظر با آدرس داده شده برویم و در این جدول به درایه 1100001111 مراجعه می‌کنیم تا PF# را بدست آوریم.

در واقع جدول سطح 1 آدرس جدول سطح 2 را به ما می‌دهد و جدول سطح 2 PF# را به ما می دهد.(اگر صفحه حاضر باشد در غیر این صورت Page Fault می دهد.)

11:30

نکته: صفحاتی که از وقتی آمده تغییرکرده را کثیف می گویند واگرتغییر نکرده باشد تمیز می گویند.

۱. TLBs Translation Lookaside Buffers

جستجو به شکلی که content می‌گیرد (نه آدرس) و content می دهد:

- Content Addressable Memory (CAM)
- Associative Memory

Figure 4-12. A TLB to speed up paging.

Valid	Virtual page	Modified	Protection	Page frame
1	140	1	RW	31
1	20	0	R X	38
1	130	1	RW	29
1	129	1	RW	62
1	19	0	R X	50
1	21	0	R X	45
1	860	1	RW	14
1	861	1	RW	75

فرض کنید جداول صفحه به شکل بالا باشد و فقط آدرس صفحات پر را نگه داریم ، استفاده از این روش نیاز به جستجو دارد چون دیگر index نداریم هر چند تعداد درایه های جدول کم شده اما باز هم جستجو زمانگیر است ، اگر بخواهیم برای این کار از رجیستر استفاده کنیم چون تعداد فرایند ها زیاد است تعداد زیادی رجیستر نیاز داریم که داشتن این همه رجیستر امکان ندارد ، برای حل مشکل ما تنها صفحاتی که اخیرا از آنها استفاده کرده و به آنها نیاز داریم را ملاک قرار می‌دهیم و به تعداد آنها رجیستر به MMU می دهیم. هر درایه یک رجیستر مخصوص به خود دارد. تعداد رجیشتر ها یک مقدار fix است، مثلا عدد128 عدد. اگر اطلاعات درTLB نباشد (TLB miss) به معنای Page Fault نیست زیرا ممکن است در حافظه باشد. اگرآنجا هم نبود نقص صفحه است. اگر در TLB نبود اما درحافظه بود باید آدرس صفحه را به TLB بیاوریم و TLB را Update کنیم.

اما مشکلی که وجود دارد اینکه TLB ، ایندکس نیست و با وجود اینکه سرعت رجیستر ها بسیار زیاد است اما جستجو بین 128 رجیستر بیش از خواندن یک خانه رم می‌شود، برای حل این مشکل ازAssociative Memory استفاده می شود . این کار را جستجوی موازی یا پارالل نیز می گویند.

PAR یا Parallel Accessible Register

در این روش وقتی به دنبال آدرسی می‌گردیم ورودی را همزمان به هر 128 رجیستر می‌دهیم و در یک کلاک همزمان وجود آن آدرس در هر 128 رجیستر بررسی می‌شود یعنی هر رجیستر ورودی را با محتوای خودش مقایسه می کند. در صورتی که همه FALSE برگردانند OR آن‌ها نیز FALSE خواهد شد و یعنی آدرس مورد نظر در رجیستر ها وجود ندارد اما در صورتی که OR رجیستر آن‌ها TRUE باشد یعنی آدرس مورد نظر در یکی از آن‌ها وجود دارد و همزمان آن رجیستر #PF را در خروجی می گذارد.

اهمیت TLB را وقتی می‌فهمیم که می‌بینیم در جداول صفحه دو سطحی بازای هر بار مراجعه به حافظه دو بار هم باید برای جداول صفحه به حافظه مراجعه شود یعنی برای دستوری که پیش از این 3 مراجعه به حافظه داشت اکنون با جداول صفحه دو سطحی 9 مراجعه به حافظه نیاز است. با ایجاد TLB تا 95 درصد مراجعات با سرعت خواندن یک رجیستر آدرس بدست می‌آید و اگر در TLB نبود با احتمال 5٪ بایستی به حافظه مراجعه کنیم که کند است.

-- بر چه اصلی 95% احتمال دارد صفحات مورد نیاز درTLB باشد؟

اصل Locality یا اصل مراجعات محلی

47:00*

54:30

مدیریت نرم افزاری TLB

CISC ← تعداد Instruction ها را زیاد می کند تا سرعت زیاد شود. 400 ـ 300 دستورسخت افزاری

RISC ← تعداد Instruction ها را کم می کند تا سرعت زیاد شود. 25 ـ 20 دستورسخت افزاری

- در RISC دستورات تک سیکلی انجام می شود به همین خاطر دستورات سبک‌تر و سریع‌تر انجام می‌شود واز طرفی 80% دستورات در برنامه‌ها همین دستورات سادهو سبک هستند می باشد.
- در نتیجه 80% دستورات را خیلی سریع تر انجام می دهد و 5% دستورات را ندارد و باید با دستورات دیگر پیاده سازی کرد که زمان بیشتری می‌گیرد و در نهایت آن 95٪ جبران این 5٪ را می‌کند.

نتیجه‌گیری از مباحث بالا:

ما می‌دانیم که MMU آدرس را ترجمه می کند که سخت افزاری است . اولین گام ترجمه آدرس هم این است که P# را به TLB می‌دهد که دو حالت رخ می‌دهد یا hit می‌شود که MMU خودش FP# را سمت چپ آفست می‌گذارد و آدرس را روی گذرگاه آدرس می گذارد. اما اگرTLB Miss شد باید به سراغ جدول سطح 1 بعد از آنجا جدول سطح 2 برود در آنجا اگر Page fault رخ داد OS کار را برعهده می‌گیرد و در غیر این صورت که روال ترجمه آدرس ادامه می یابد.

در طی مراحل بالا از ابتدای کار تا وقتی TLB Miss می‌شود وظیفه MMU است. از زمانی هم که Page Fault رخ می‌دهد هم به بعد وظیفه OS است. اختلاف بر سر زمانی است که TLB Miss می‌شود تا Page Fault رخ می‌دهد که این قسمت وظیفه کیست؟

MMU TLB Miss ? Page Fault OS

|----------|----------|----------|

اگرMMU این کار را انجام دهد چون سخت افزاری است سریع‌تر است اما MMU بزرگ می شود .

اگرOS : سرعت کند می شود.

چون در80% اوقات این اتفاق نمی افتد اگر به عهده OS باشد MMU کوچک‌تر و سریع ترمی شود و در5% کندتر می شود که درمجموع سرعت بهتر می شود.

1:10:30

جدوال وارونه

PF#	Valid	Process#	Page#	M
0	1	2		
1	1	1		
2	1	0		
3	1	1		
	1			

در این جدول صفحه بر خلاف جداول صفحه قبلی به جای اینکه مشخص کند page مورد نظر در چه Page Frame ی است مشخص می‌کند در Page Frame ها چه Pageی قرار دارد (Index روی PF# است). از طرفی تمام Page Frame ها مربوط به یک فرآیند نیست و در نتیجه به جای اینکه بازای هر فرایند یک جدول صفحه داشته باشیم یک جدول برای کلیه فرآیند ها داریم . این جدول صفحه نسبت به جداول قبلی بسیار کوچک است چرا که فقط اطلاعات صفحاتی را نگه می‌دارد که در حافظه هستند.

خصوصیات :

1. کلا یک جدول داریم برای همه فرایندها .
2. ایندکس روی PF# است .
3. شماره فرایند ها هم باید مشخص باشد.

مزیت : **بسیار فشرده و کوچک**

مشکل : به دلیل نیاز به جستجو کند می شود.

چگونه مشکل سرعت را برطرف کنیم :

با استفاده از Hash Function، اما در Hash احتمال تصادف وجود دارد ولی با وجود این بازهم سرعت بیشتر می شود چرا که دیگر دو چیز را می‌دانیم اول استفاده ازTLB و دوم اصل مراجعات محلی که در نتیجه با توجه به اصل مراجعات محلی می‌دانیم که تا 95 در صد به صفحات محدودی نیاز داریم و با پشتوانه TLB می‌دانیم که برای این صفحات محدود مشکل جستجو هم نداریم

ساعت 15:45

جلسه چهاردهم چهارشنبه ۱۳۸۸/۷/۷ ساعت ۸:۰۰

الگوریتم های جایگزینی صفحه : صفحه ۴۷۳ کتاب

فرض کنید که Page Fault رخ داده و همه قاب های صفحه پرهستند اکنون باید تصمیم بگیریم کدام قاب را خارج کنیم. برای این کار چندین الگوریتم وجود دارد مثل : (درجه اهمیت با * مشخص شده است)

3. Optimal*****
4. FIFO*****
5. NRU
6. Second chance**
7. Clock**
8. LRU*****
9. LFU
10. MFU
11. WS
12. Page Buffering
13. Aging**

خود LRU چند دسته است :

5. Link List
6. 64bit counter
7. n*n matrix

1. الگوریتم بهینه یا Optimal

صفحه ای را خارج کن که درآینده دورتری به آن نیاز خواهد شد. ایراد این الگوریتم این است که نیاز به پیش بینی آینده دارد و به همین دلیل قابل پیاده‌سازی نیست اما اگر قابل پیاده سازی بود بهترین الگوریتم بود. از این الگوریتم به عنوان شاخص مقایسه استفاده می‌شود و تنها مکی توانند آن را شبیه سازی کنند.

2. NRU Not Recent Used

صفحه ای را بیرون می کنیم که در پریود اخیر به آن کمتر مراجعه شده است و حتی الامکان تغییر نیافته است.

توجه : این الگوریتم شمارنده ندارد و به هر صفحه‌ای مراجعه می‌شود بیت مراجعه آن یک می‌شود و تعداد مراجعات مطرح نیست همچنین در آخر پریود زمانی تمام بیت‌های مراجعه صفر می شود.

این الگوریتم در ابتدای پریود مشکل دارد چون درابتدای پریود هیچ کدام مراجعه نداشته اند و الگوریتم یکی را اتفاقی بیرون می اندازد که ممکن است صفحه ای باشد که به آن نیاز داریم . آخر پریود هم این الگوریتم ممکن است بد رفتاری داشته باشد چون ممکن است به همه مراجعه شده باشد.

برای حل مشکل آخر پریود می توانیم پریودها را کوتاه‌تر کنیم اما برای حل ابتدای پریود کاری نمی توانیم بکنیم. اینکه به صفحه مراجعه شده یا نه را ازروی بیت R می‌فهمیم (بیت R یکی از بیت‌های جدول صفحه است) که این بیت توسط MMU مقداردهی می شود.

سیستم عامل در دو مورد وارد عمل می شود:

1. Trap ← Page Fault
2. صفرکردن بیت R درآخرهر پریود

برای ایجاد الگوریتم ابتدا باید پریود تعریف کنیم این کار را با تایمر و حدود 20 میلی ثانیه در نظر می گیریم. در 20 میلی ثانیه می‌توان بیش از یک میلیون دستورالعمل اجرا کرد اگر این مقدار کمتر از این باشد سربار زمانی آن زیاد می شود. وقتی که بر سر پریود مشخص شده وقفه آمد این وظیفه سیستم عامل است که بیت‌های R پریود اخیر را صفر کند. مجدداً از اول پریود هر بار که به صفحه‌ای مراجعه می‌شود این MMU است که بیت R آن صفحه را یک می کند.

NRU ساده فقط با بیت R کارمی کند اما درNRU پیشرفته علاوه بربیت R بیت M نیز درنظر گرفته می شود یعنی اگر Page Fault آمد آن صفحه ای خارج می شود که بیت R آن صفر است اگرهمه 1 بودند صفحه ای خارج می شود که بیت M آن صفر است چرا که یک نوشتن به نفع ماست .

صفحات از لحاظ بیت R و M به چهار کلاس تقسیم می شوند:

R	M
0	0
0	1
1	0
1	1

سؤال- با توجه به ردیف دوم جدول بالا چگونه ممکن است صفحه‌ای که به آن مراجعه نشده است تغییر یافته باشد؟

صفر بودن R به این معنی است که در پریود اخیر به آن مراجعه نشده است و در پریود های قبلی ممکن است به آن مراجعه و تغییر یافته باشد.

ـ اگرسخت افزاری بیت R و M را پشتیبانی نکند آیا سیستم عامل می تواند آنرا انجام دهد؟

ازبیت های Protection سخت افزاری استفاده می کنیم.

RWX

<--------

M	R	RWX

سخت‌افزار MMU در جدول خود RWX را دارد. سیستم عامل جدولی ایجاد می‌کند و RWX واقعی را در آن نگه می‌دارد و بیت‌های R و M را نیز به آن می افزاید اما در جدول صفحه واقعی بیت‌های RWX را 000 می‌گذارد. اولین باری که به صفحه مراجعه می‌شود به دلیل exception protection تله رخ می‌دهد در این لحظه سیستم عاما وارد عمل می‌شود و از آنجایی که بیت‌ها RWX واقعی فرایند را دارد RWX را در MMU اصلاح می‌کند و اجازه خواندن صفحه را می دهد و بیت R خود را نیز یک می کند.

برای بیت M نیز همین حالت رخ می‌دهد و هر زمان که خطای نوشتن رخ داد بیت M را نیز به همین روش اصلاح می کند.

3. FIFO

صفحه‌ای را از حافظه خارج کن که قدیمی تر است. (زود تر وارد شده است و یا مدت بیشتری در حافظه بوده)

پیاده‌سازی آن با Link List قابل انجام است.

مزیت : سادگی پیاده‌سازی و هزینه کم

ایراد: الگوریتم خوبی نیست چرا که ممکن است صفحه‌ای قدیمی باشد اما هنوز صفحه پرکاربردی است.

51:30

4. Second Chance

به گونه ای ترکیب FIFO و NRU است.

ایده اش این است که قدیمی ترین صفحه ای را خارج کن که اخیرا به آن مراجعه نشده است. یعنی به سراغ قدیمی ترین صفحات می‌رویم و آن صفحه‌ای را بیرون می‌کنیم که بیت R آن صفر باشد.

این الگوریتم پریود یک نیست.

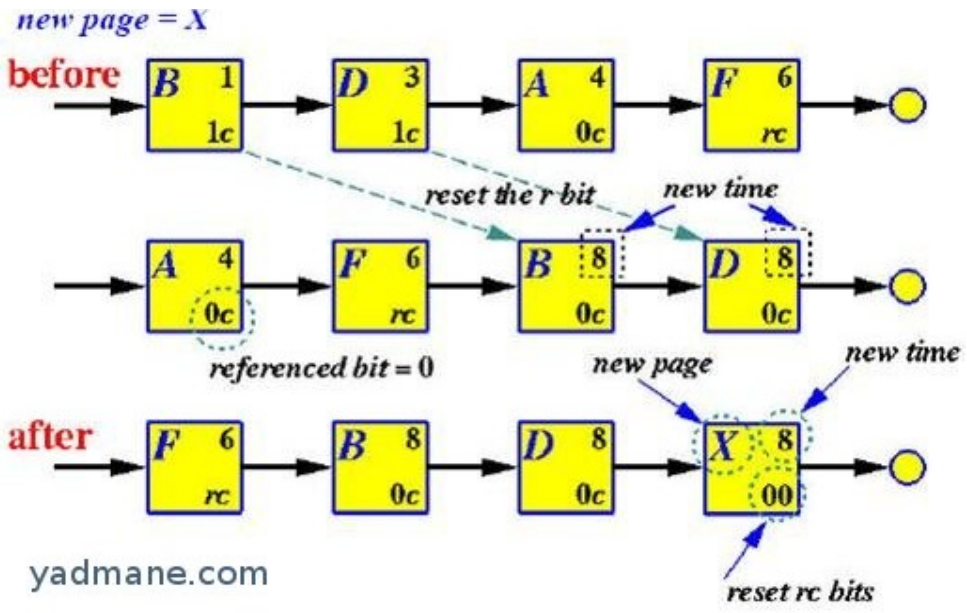
معمولا در کنکور هر بار آمده اولین باری که صفحه وارد می شود را1 می کند.

اگربیت R برای همه صفر یا برای همه یک باشد رفتار الگوریتم همان FIFO می باشد.

در شکل زیر وقتی که بیت R آن یک است بیت R آن را صفر کرده و صفحه را به آخر لیست می بریم. و اگر بیت R صفر بود آن صفحه را خارج می‌کنیم و صفحه جدید جایگزین آن می شود.

چرا به این الگوریتم دومین شانس می گویند؟

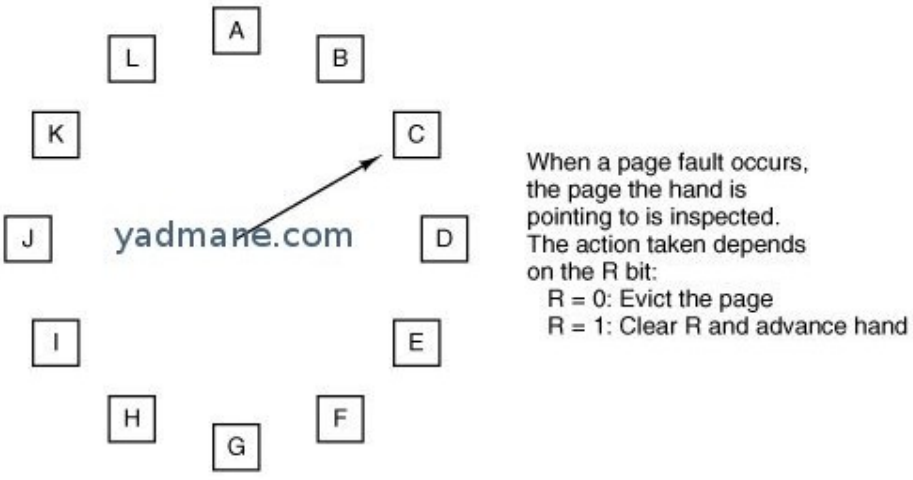
چون وقتی به صفحه‌ای قدیمی مراجعه می‌کنیم که بیت R آن یک است یک شانس دیگر به آن می دهیم. این شانس ممکن است در صورت مراجعات مکرر به صفحه بار ها به صفحه داده شود.

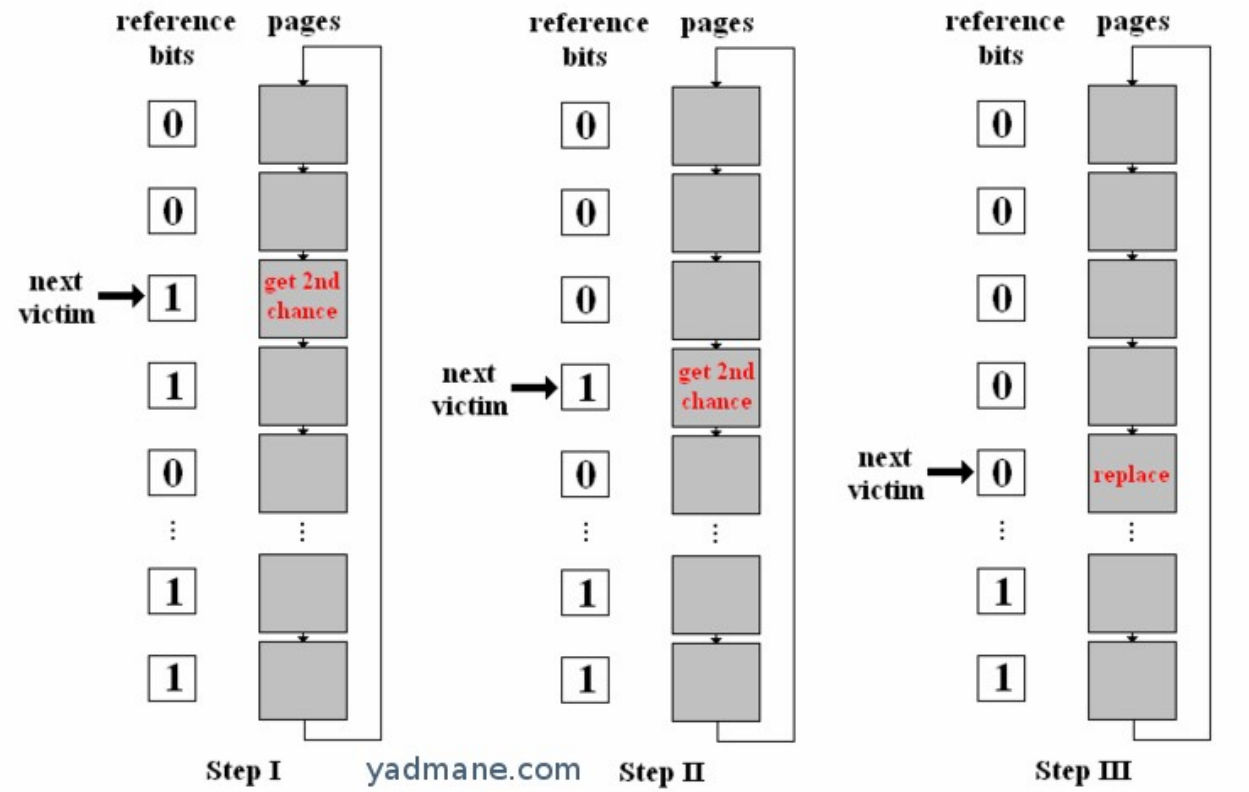


5. Clock

همان الگوریتم Second chance است با این تفاوت که Link List چرخشی است. مهم ترین نکته این است که رفتارهردو الگوریتم کاملا یکسان است . ومزیت Clock در سرعت اجرای آن است .

شکل بعد الگوریتم Clock است که همان رفتار Second Chance را دارد به جای اینکه صفحه را به آخر لیست ببریم اشاره گر را جلو می بریم.





مثال: اگر دنباله مراجعات به صفحات یک فرایند به ترتیب زیر باشد درکدام یک از الگوریتم های زیر 7 نقص صفحه رخ می دهد؟

1 , 2 , 4 , 3 , 1 , …

1- LRU 2- Second Chance 3- Clock 4- FIFO

برای این مسئله نیاز به حل 4 الگوریتم نیست چون مطمئنا Clock یا Second Chance نیست چرا که جواب این دو یکی می باشد ازبین LRU و FIFO چون FIFO راحت تراست آن را حل می کنیم .

6. LRU

ایده اصلی این الگوریتم این است : صفحه ای را بیرون بینداز که از آخرین استفاده آن مدت بیشتری گذشته است.

این الگوریتم به دلیل هزینه‌های سنگین آن قابل پیاده‌سازی نیست اما آنقدر محبوب است که روش‌های مختلفی برای شبیه سازی آن به کار برده اند.

نحوه پیاده‌سازی :

- Link List
- Time

1:19:20

پیاده سازی LRU با Link List :

یک link list ایجاد می‌کنیم ، به هر صفحه‌ای که مراجعه شد ابتدا در link list جستجو می‌کنیم اگر صفحه در link list وجود داشت آن را به انتهای link list منتقل می‌کنیم و اگر در link list وجود نداشت آن را به انتهای link list اضافه می‌کنیم در صورتی هم که بخواهیم صفحه‌ای را خارج کنیم آن را از ابتدای link list انتخاب می کنیم.

فرض کنید دنباله مراجعات به ترتیب زیر آمده زیر باشد:

3 , 4 , 3 , 5 , 8 , 3 , 2 , 4

دنباله Link List در گام های مختلف به ترتیب زیر است:

3				
3	**4**			
4	**3**			
4	**3**	**5**		
4	**3**	**5**	**8**	
4	**5**	**8**	**3**	
4	**5**	**8**	**3**	**2**
5	**8**	**3**	**2**	**4**

مشکل :

1. برای پیاده کردن صفحه نیاز به جستجو داریم .
2. باید صفحه را جابجا کنیم.
3. قابل پیاده سازی نیست چرا که پریود اعمال بالا بیش از Instruction است. یعنی به ازای هر مراجعه باید جستجو صورت گیرد.

- **پیاده سازی LRU بوسیله time :**

زمان مراجعه به هر صفحه را ذخیره کنیم زمانی که Page Fault رخ داد صفحه‌ای که زمانش قدیمی تر است را خارج می کنیم. هرچند نیاز به Search هم دارد اما چون Search بازای Page fault است باز قابل قبول تر است. اما چون بازای هر مراجعه باید ساعت را بنویسیم خیلی بد است .

- **پیاده سازی LRU با64Bit Counter**

چون دقت clock کامپیوتر پایین است نمی‌توانیم از آن استفاده کنیم مثلاً در هر ثانیه 900M دستورالعمل اجرا می‌شود پس clock کامپیوتر مفید نیست برای اینکه دقت بیشتری داشته باشیم ساعت را یک counter درنظرمی گیریم که بازای هر مراجعه یکی می شمارد.

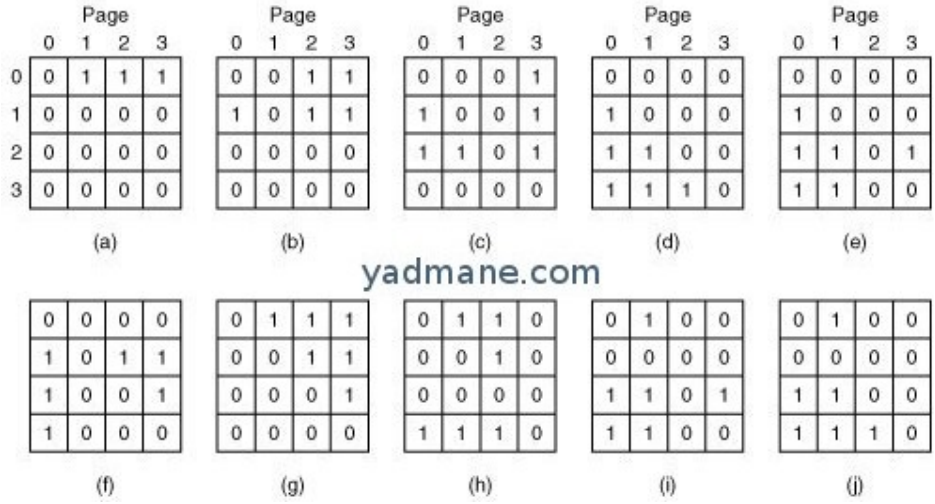
مشکل این روش این است که جدول صفحه نیاز به یک فیلد زمان دارد که درواقع 64 بیت به آن اضافه می شود و جدول صفحه خیلی بزرگ می‌شود (تقریباً 3 برابر می شود). دومین مشکل این است که برای نوشتن آن هر بار MMU باید 8 بایت را بنویسد و این زمان گیر است. اما به هر حال این معقول ترین روش LRU است.

- **پیاده سازی LRU باn*n Matrix صفحه ۴۸۰ کتاب**

این ماتریس n*n بیت است که n تعداد قاب های صفحه است . به هر صفحه‌ای که مراجعه می کنیم سطرمربوطه را 1 کرده و ستون مربوطه را صفر می کنیم موقع page fault کوچکترین سطر انتخاب می شود.

در شکل زیر قبل از شکل a همه بیت‌ها صفر بوده‌اند . حال دنباله زیر آمده و شکل به ترتیب زیر تغییر یافته است:

0 1 2 3 2 1 0 3 2 3



1:43:30 غ

مشکلات :

1. حجم حافظه ای که می خواهد بسیارغول پیکر است.
2. بازای هرمراجعه باید یک سطر و یک ستون را تغییر دهیم.

1:52:30

تمرین- دنباله مراجعات به صفحات یک فرایند به ترتیب ازچپ به راست به شرح زیراست.

2 3 2 1 5 2 4 5 3 2 5 2

فرض کنید که حافظه اصلی دارای سه قاب صفحه است که درابتدا خالی است تعداد نقص صفحه و جایگزینی صفحه را در الگوریتم FIFO ، بهینه ، LRU و ساعت (دومین شانس) بدست آورید.

ترتیب ورود صفحات	2	3	2	1	5	2	4	5	3	2	5	2	
optimal **یا** **بهینه**	2	2	2	2	2	2	4	4	4	2	2	2	PF=7
		3	3	3	3	3	3	3	3	3	3	3	
				1	5	5	5	5	5	5	5	5	**جایگزینی** 3
Page Fault	F	F		F	F& R		F& R			F& R			

ترتیب ورود صفحات	2	3	2	1	5	2	4	5	3	2	5	2	
LRU	2	2	2	2	2	2	2	2	3	3	3	3	PF=6
		3	3	3	5	5	5	5	5	5	5	5	
				1	1	1	4	4	4	2	2	2	**جایگزینی** 4
Page Fault	F	F		F	F& R		F& R		F& R	F& R			

ترتیب ورود صفحات	2	3	2	1	5	2	4	5	3	2	5	2	
FIFO	2	2	2	2	5	5	5	5	3	3	3	3	PF=9
		3	3	3	3	2	2	2	2	2	5	5	
				1	1	1	4	4	4	4	4	2	**جایگزینی** 6
Page Fault	F	F		F	F& R	F& R	F& R		F& R		F& R	F& R	

ترتیب ورود صفحات	2	3	2	1	5	2	4	5	3	2	5	2	
CLOCK	2* →	2* 3*	2* 3*	→2* 3*	5* →3	5* 2*	→5* 2*	→5* 2*	3* →2	3* →2*	→3* 2	→3* 2*	PF=8
		→	→	1*	1	→1	4*	4*	4	4	5*	5*	جایگزینی 5
Page Fault	F	F		F	F& R	F& R	F& R		F& R		F& R		

توضیح در ارتباط با Clock :

- وقتی صفحه جدیدی می‌آید بیت R آن یک می‌شود پس * دارد.
- ← یعنی قدیمی ترین صفحه
- وقتی به صفحه‌ای مراجعه می‌کنیم که * ندارد به آن * اضافه می‌کنیم در این صورت → را دست نمی زنیم چون وقتی به صفحه‌ای مراجعه شده که قبلاً در حافظه بوده یعنی صفحه‌ای خارج نشده و این بدین معناست که قدیمی ترین صفحه تغییر نکرده است.
- اگر همه صفحات * دار باشند یک دور اشاره گر می چرخد و تمام * ها را بر می‌دارد سپس قدیمی ترین صفحه را حذف می کنیم.

روش کنکوری الگوریتم های قبل:

در این روش به تعداد frame ها سطر داریم و وقتی قرار است صفحه‌ای جایگزین شود قبلی خط می‌خورد و صفحه جدید جایگزین می شود:

2 5 2 3 5 4 2 5 1 2 3 2

optimal :

2	4	2
3		
1	5	

تعداد عد دهای داخل کادر= PF = 6

تعداد جایگزینی=3= اعداد خط خورده (قرمز)

FIFO :

2	5	3
3	2	5
1	4	2

PF =9 جایگزینی =6

LRU :

2	3	
3	5	
1	4	2

PF =7 جایگزینی =4

7. Aging (Additional Reference bit)

1) شبیه سازی الگوریتم LRU است.
2) چون LRU الگوریتم محبوبی است اما قابل پیاده سازی نیست سعی درشبیه سازی آن داریم.
3) LRU آیینه Optimal درگذشته است. ایده این است که صفحه‌ای که مدت زیادی از آن استفاده نشده احتمالاً در آینده نیز کمتر از آن استفاده خواهد شد. (الهام گرفتن ازگذشته برای پیش بینی آینده)

الگوریتم Aging چگونه کار می کند؟

برای این کار به جدول صفحه یک byte به نام Aging اضافه می کنیم. سیستم پریود یک است . سر هرclock pulse سیستم عامل باید بایت Aging را یک بیت به سمت راست شیفت دهد ، بیت R را در MSB آن کپی کند و بیت R را صفر کند. این کار توسط OS انجام می شود.

فرض کنید در جدول Step 1 هستیم ، سیستم عامل سر کلاک پالس فراخوانی شده و می‌بیند به صفحات اول و سوم و چهارم مراجعه شده است. بایت Aging را یک بیت به راست شیفت می‌دهد و بیت R را از راست وارد بایت Aging می کند.

P/A	PF#	M	R	Aging Byte
			1	0 1 1 0 1 1 1 0
			0	1 1 0 1 1 1 0 0
			1	0 0 0 0 1 0 0 0

Step 1

→

P/A	PF#	M	R	Aging Byte
			→	1 0 1 1 0 1 1 1
			→	0 1 1 0 1 1 1 0
			→	1 0 0 0 0 1 0 0

Step 2

نکته: ارزش مکانی بیت ها مهم است نه تعداد 1 ها .

ـ اگربیت Aging = 11110000 باشد مفهوم آن چیست؟

3. درچهار پریود گذشته اخیر به این صفحه مراجعه شده است اما نمی دانیم درهر پریود چند بار. صحیح
3. به این صفحه حداقل 4 بار مراجعه شده است. صحیح
3. به این صفحه در4 پریود متوالی مراجعه نشده است. صحیح
3. به این صفحه تاکنون در4 پریود مراجعه شده است. غلط : چون ما از8 پریود قبل تر خبر نداریم.
3. به این صفحه تاکنون 4 بارمراجعه شده است. غلط
3. به این صفحه اخیرا 4 بار مراجعه شده است. غلط

14:00

این الگوریتم هنگام page fault صفحه ای را خارج می کند که Age آن کمترباشد زیرا در گذشته دور تری استفاده شده است.

تفاوت آن با LRU این است که داخل پریود ها را نمی دانیم و از گذشته های دور نیز خبر نداریم.

تست فصل ۶ شماره ۲۴: یک کامپیوتر کوچک حاوی چهار قاب صفحه است . دراولین تیک ساعت ، بیت های ارجاع R قاب های صفحه 0111 می باشند. (بیت ارجاع قاب صفحه اول 0 و مابقی 1 است) ارزش بیت های ارجاع در تیک های بعدی به ترتیب عبارتنداز 1011 ، 1010 ، 0010 ، 1010 (ازراست به چپ) .
درصورت استفاده از الگوریتم Aging وشمارنده 8 بیتی ، پس ازآخرین تیک ساعت ، کدام قاب صفحه به هنگام بروز page fault کاندیدای جابجایی محسوب می شود.

- قاب صفحه اول
- قاب صفحه دوم
- قاب صفحه سوم
- قاب صفحه چهارم

پاسخ : داده‌ها را به ترتیب ورود بیت به بیت وارد می‌کنیم جدول زیر حاصل می شود:

	Aging Byte
اول	0 → 1 0 → 1 1 0 → 0 1 1 0 → 1 0 1 1 0
دوم	1 → 0 1 → 0 0 1 → 0 0 0 1 → 0 0 0 0 1
سوم	1 → 1 1 → 1 1 1 → 1 1 1 1 → 1 1 1 1 1
چهارم	1 → 1 1 → 0 1 1 → 0 0 1 1 → 0 0 0 1 1

8. MRU (Most Recently Used)

عکس LRU ـ صفحه‌ای که اخیراً از آن استفاده شده را اخراج کن. خیلی مسخره است.

35:30

9. LFU

مثل الگوریتم Aging است فقط قبل از اینکه بیت R را صفرکند با بایت LFU جمع می کند.

R		Old LFU Byte	LFU Byte
1	+	0 1 1 0 1 1 1 0 =	0 1 1 0 1 1 1 1

اگربایت LFU مثلا 6 باشد یعنی در6 پریود متوالی یا غیرمتوالی به آن مراجعه شده است و نمی دانیم چند بار.

مشکل این الگوریتم این است که اگر یکی در گذشته دور شمارنده اش بالا رفته و مدتی است شمارنده آن تغییر نکرده در حافظه می ماند.

Aging گذشته دور را فراموش می کند اما LFU این طور نیست .

MFU .10

عکس LFU است آن را بیرون می اندازد که شمارنده آن از همه بیشتر است.

– تست قبل را در نظر بگیرید . کاندیدای خروج هر یک از الگوریتم های LFU ، Aging ، MFU کدام است؟

	Aging Byte	
اول	0 → 1 0 → 1 1 0 → 0 1 1 0 → 1 0 1 1 0	
دوم	1 → 0 1 → 0 0 1 → 0 0 0 1 → 0 0 0 0 1	**MFU**
سوم	1 → 1 1 → 1 1 1 → 1 1 1 1 → 1 1 1 1 1	**Aging**
چهارم	1 → 1 1 → 0 1 1 → 0 0 1 1 → 0 0 0 1 1	**LFU**

Page Buffering .11

FIFO باش اما کمی صبر کن شاید پشیمان شوی . در اینجا دو صف داریم ، صف اصلی و صف بافر یعنی صف آن‌هایی که یک بار کاندید خروج شده اند.

step 1:

Main List :	3	5	4	2	9
Buffer List & m=1 :	1	7	8		
Buffer List & m=0 :	6	11			

مراحل به ترتیب زیر است:

1. یکی از صفحات بافر را اخراج می‌کنیم (در این مثال 6 را واقعا خارج می کنیم زیرا زود تر از همه وارد شده و بیت M آن نیز صفر است.)
2. یکی از صفحات لیست اصلی را کاندید اخراج می‌کنیم که در اینجا 3 کاندید اخراج می باشد. (واقعاً اخراج نمی‌کنیم بلکه بافر می کنیم)
3. به جای آن صفحه جدید که 10 می‌باشد را وارد می کنیم.

step 2:

Main List :	5	4	2	9	10
Buffer List & m=1 :	1	7	8	3	
Buffer List & m=0 :	11				

اگر 3 مجدداً استفاده شود به لیست بالا برمی‌گردد و یکی از لیست بالا بافر می شود.

تغییر صف توسط سیستم عامل صورت می‌گیرد و MMU اطلاعی از این صف ها ندارد و اگر صفحه در صف اصلی نباشد مثل این است که در جدول صفحه نیست و Page Fault رخ می‌دهد در این زمان سیستم عامل است که وضعیت را بررسی کرده و می‌بیند صفحه بیرون از حافظه نیست سپس MMU را update می کند.

در واقع بافر درهمان حافظه است و تنها در ساختمان داده لیست ها تغییراتی داده شده است.

56:30

نکات طراحی سیستم های صفحه بندی:

صفحه 484 کتاب

درست است که Paging باعث شده است که نیاز نباشد همه برنامه را به داخل حافظه Load کنیم . اما یک مقدار حداقلی باید در قاب ها باشد به همین دلیل است که وقتی بعضی برنامه ها را نصب می کنیم می گوید حداقل به n مقدار RAM نیاز داریم . درست است که به تمام صفحات یک برنامه نیاز نداریم اما تعدادی از صفحات هستند که همین الان با آنها کارمی کنیم وهنوز با آنها کارداریم ، به این صفحات working set یا مجموعه کاری می گویند.

Trashing یا کوبیدگی :

اگر تعداد قاب هایی که به یک فرایند می دهیم کمتر از working set آن باشد آنگاه به دلیل Page Fault های مکررTrashing رخ می دهد چون مدام باید صفحات یک برنامه وارد وخارج شود وسرعت به شدت کاهش می یابد.

وقتی یک نرم‌افزار را اجرا می‌کنیم ابتدای کار کمی طول می‌کشد چون می‌خواهد Working Set خود را وارد حافظه کند.

اندازه و محتوای مجموعه کاری ثابت نیست.

1:03:15

12. WSClock (working set clock)

این الگوریتم شبیه Clock است یعنی عقربه به قدیمی ترین صفحه اشاره می کند با این تفاوت که در Clock در صورتی صفحه را خارج می‌کردیم که بیت R آن صفر باشد اما در اینجا در صورتی صفحه را خارج می کنیم که جزو مجموعه کاری نباشد.

چگونه working set راتشخیص دهیم؟

W(k , t)

w مجموعه صفحاتی که در لحظه t در k مراجعه اخیر به آن‌ها مراجعه شده است.

2 درهرلحظه از زمان t مجموعه ای ازصفحات وجود دارند که درk مراجعه اخیر به حافظه مورد استفاده واقع شده اند این مجموعه رامجموعه کاری می گوییم.

درعمل از الگوریتم Aging استفاده می کنیم اگر در k مراجعه اخیر MSB شامل یک بود آن صفحه جزو مجموعه کاری است وآن را نگاه می داریم.

مثال: k = 4 است کدام یک جزو مجموعه کاری است؟

1	1	0	1	0	0	0	0	true
1	0	0	0	0	0	0	0	true
1	1	1	1	0	0	0	0	true
0	0	0	0	0	1	1	1	false

تست- مراجعات فرآیندی به صفحاتش به ترتیب زیر است :

1 2 4 3 1 7 5 2 3 1 3 2 |7 9 3 2 1 6 5

اگر k=5 باشد بین مراجعات 12 و 13 working set کدام است؟

{1,2,3}

نکته: کلیه 12 روش صفحه بندی را که تا کنون گفته‌ایم صفحه بندی درخواستی بوده است یعنی وقتی صفحه‌ای را به حافظه می آوریم که Page Fault رخ داده باشد. در مقابل این روش‌ها پیش صفحه بندی است یعنی به جای اینکه اجازه دهیم Page Fault رخ دهد تا صفحه را بیاوریم از قبل پیش‌بینی می کنیم که چه صفحاتی را بیاویم، این باعث افزایش سرعت می‌شود اما مشکل پیش‌بینی است. البته می‌توانیم از اصل locality و روش‌های دیگری تا حدی برای این کار کمک بگیریم.

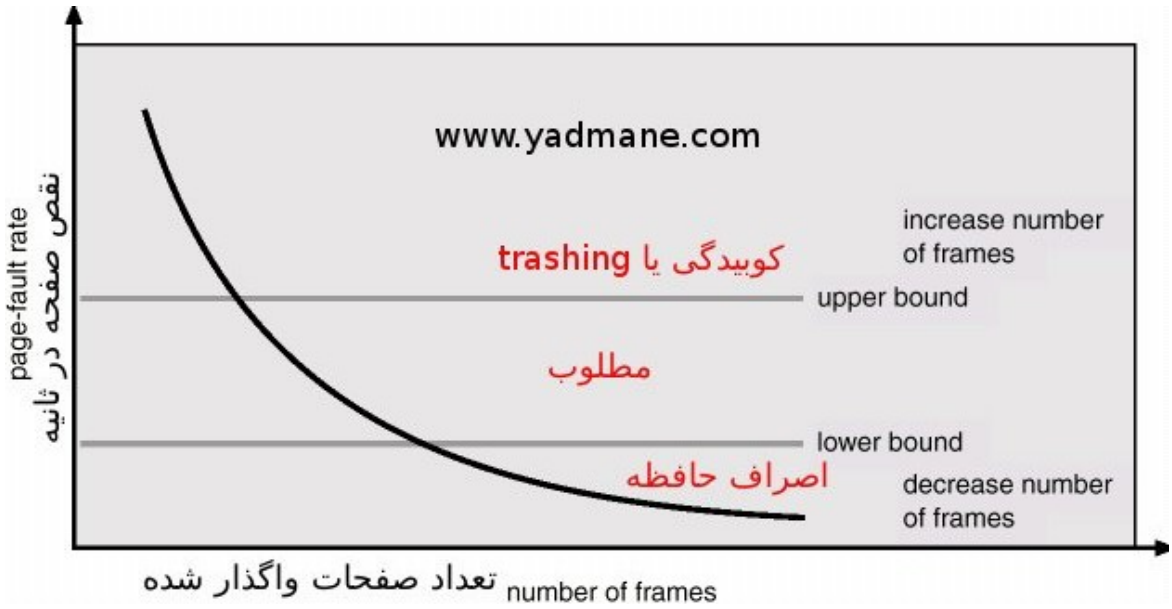
1:18:00 1:20:30

سیاست های تخصیص

1) Local ← صفحه ای که خارج می کنیم برای خود فرآیند باشد.

2) Global ← صفحه ای که خارج می کنیم از کل فرایند ها باشد.

درحالت کلی global بهتراست زیرا working set اندازه اش ثابت نیست اگر ازسیاست local استفاده کنیم working set هر فرایند ثابت می شود و ممکن است گاهی باعث کوبیدگی و گاهی باعث اتلاف حافظه شود. البته در سیاست Local به روش های دیگر می توانیم working set را تغییردهیم . مثلا به روش سرکشی بیاییم تخصیص ها را عوض کنیم، الگوریتمی که این کار را انجام می دهد الگوریتم PFF یا Page Fault Frequency نام دارد. دراین الگوریتم ازآنهایی که page fault هایشان کم است می گیرند به آنهایی که Page Fault شان زیاد است می دهند. دقت کنید که این الگوریتم جایگزینی نیست بلکه الگوریتم کنترل بار است.



شکل بالا نشان می‌دهد افزودن صفحات یک فرآیند از یک حدی بالاتر تأثیری چندانی در کاهش نقص صفحه ندارد و از یک حدی پایین‌تر به صورت نمایی نقص صفحه را افزایش می دهد.

دو تعریف مهم:

Belady's Anomaly :

در برخی الگوریتم ها ممکن است با افزایش قاب برخلاف انتظارتعداد PF ها افزایش می یابد. مثلا الگوریتم FIFO ناهنجاری های بلیدی دارد. اما برخی الگوریتم ها مثل LRU و Optimal این مشکل را ندارند.

الگوریتم هایی که شبیه FIFO به نوعی خاصیت خروج به ترتیب ورود را دارند مثل Second Chance ، ساعت و Page Buffering این خاصیت را دارند.

الگوریتمهای پشته:

الگوریتم هایی که این مشکل را ندارند مثل LRU یا Optimal ازنوع Stack هستند الگوریتم هایی ازنوع استک هستند که اگر n قاب داشته باشید یا n+1 قاب، برای قاب ها قاعده زیربرقرار باشد:

قاب ها زمانی که n قاب داریم زیر مجموعه قاب ها زمانی که n+1 قاب داریم خواهد بود

اندازه صفحه :

سربار حافظه درpaging :

1. سربارتکه تکه شدن داخلی : p/2
2. سربارجدول صفحه هر فرایند : S/P*e

p : اندازه صفحه

e : اندازه جدول صفحه

S: میانگین اندازه فرآیند ها

S/P : میانگین تعداد صفحه فرآیند ها

سربار کل فرآیند:

P/2 + Se / p →

می‌خواهیم سربار مینیمم شود پس نسبت به p مشتق می گیریم:

$$-se/p^2 + 1/2 = 0$$

در نهایت بهترین اندازه صفحه برای اینکه سربار صفحه بندی minimize یا Optimum باشد:

$$p = \sqrt{2se}$$

مثال : اگر e=4Byte و s=2M باشد P را حساب کنید:

S= 2M= 2^{21}

$p = \sqrt{2se}$

→ P = 2^{12} = 4KB

قطعه بندی یا Segmentation

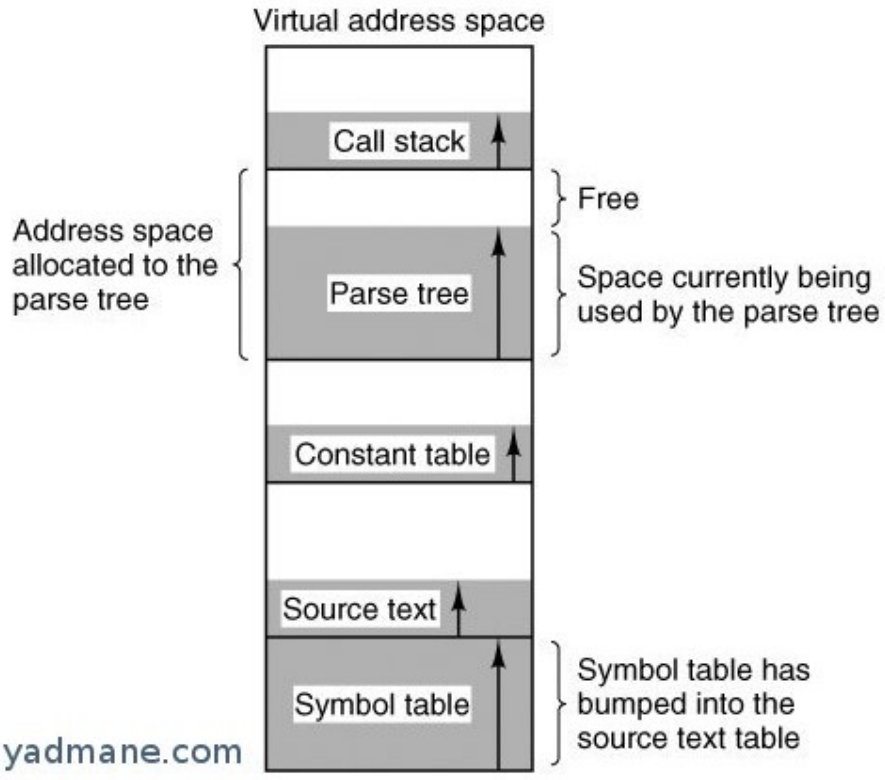
صفحه بندی ایراداتی دارد که باعث گرایش به قطعه بندی شده است.

ایرادهای صفحه بندی:

- برنامه نویس کل فضا را یک تکه می بیند ونمی تواند فرایند را به بخش های مجزای منطقی تقسیم کند .

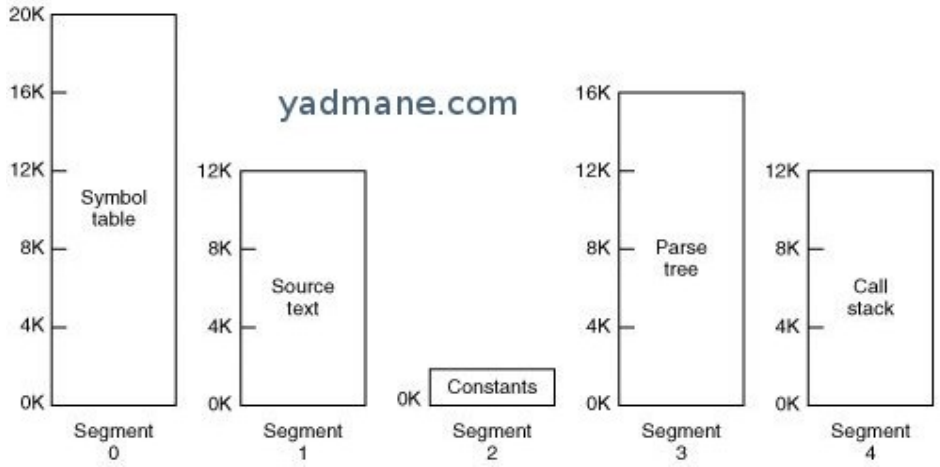
(ماژولار نیست)

- برنامه نویس نمی تواند بخش های دلخواهی ازفرایند را به اشتراک بگذارد و بخش هایی از آن را به اشتراک نگذرد.
- برنامه نویس نمی تواند بخش های مختلف فرایند را به گونه دلخواه و انعطاف پذیرحفاظت کند.
- اداره جداول و ساختمان داده هایی که رشد می کنند ساده نیست (امکان overlap وجود دارد)



در شکل بالا کامپایلری را در نظر بگیرید که جدول سمبل ها ،سورس برنامه ، جدول ثابت ها ، درخت پارز و استک خود را به ترتیب در فضای حافظه گذاشته است. تمامی این جداول و پشته ها در حال رشد هستند ، چون این فضا یک تکه و خطی است امکان دارد با رشد این جداول بر روی هم overlap پیدا کنند. این خود برنامه نویس است که باید از این اتفاق جلوگیری کند چون فضای در اختیار آن یک تکه و خطی است.

با قطعه بندی مثل شکل زیر می‌توانیم این فضا را به طور منطقی از هم تفکیک کنیم:



مزایای این روش:

1. در این روش می‌توانیم به تعداد دلخواه قطعه با اندازه پویا ایجاد کنیم و چون اداره قطعات بر عهده سیستم عامل است اگر قطعات بزرگ شوند سخت‌افزار موقع دسترسی Protection Fault می‌دهد و این مشکل را سیستم عامل باید برطرف کند.
2. چون در قطعه بندی قطعات جداگانه تعریف می‌شود می‌توانیم قطعات دلخواه را به االشتراک بگذارم و در واقع اشتراک انعطاف پذیر خواهد بود.
3. چون قطعات مجزا تعریف می‌شود می‌توانم برای هر قطعه دسترسی های خاص آن را تعریف کنیم. یعنی حفاظت به صورت کنترل شده و انعطاف پذیر خواهد بود.

در واقع چهار ایراد گفته شده درمورد paging در یک ایراد خلاصه می شود. ماژولار نیست.

موارد مقایسه	صفحه بندی	قطعه بندی
آیا برنامه نویس باید از این تکنیک آگاه باشد؟	خیر	بلی
چند فضای آدرس خطی وجود دارد؟	1	زیاد (به تعداد صفحات)
آیا کل فضای آدرس می‌تواند از اندازه حافظه فیزیکی تجاوز نماید؟	بلی	بلی
آیا رویه ها و داده‌ها می‌توانند جدا باشند و به صورت متفاوت محافظت شوند؟	خیر	بلی
آیا می‌توان جداولی را که اندازه آن‌ها کم و زیاد می شود، به سادگی با این تکنیک تطبیق داد؟	خیر	بلی
آیا به اشترک گذاشتن رویه ها بین کاربران ممکن است؟	خیر	بلی
چرا این تکنیک ابداع شده است؟	به منظور داشتن یک فضای آدرس خطی بزرگ بدون نیاز به استفاده فیزیکی بیشتر	امکان جداسازی برنامه‌ها ، داده‌ها و شکستن آن‌ها به فضا های آدرسی که منطقاً مستقل هستند را فراهم نموده و نیز به استفاده اشتراکی از قطعات و حفاظت کمک می کند.

چهار خاصیت قطعه بندی:

1. اشتراکی sharing
2. پیمانه ای Modularity
3. حفاظت protection
4. سادگی simplicity

ایرادات قطعه بندی :

1. تکه تکه شدن خارجی - که اندازه این حفره ها ممکن است خیلی بزرگ باشد.
2. ایراد دیگری که دارد این است که باید یک قطعه کامل Load شود تا برنامه اجرا شود.

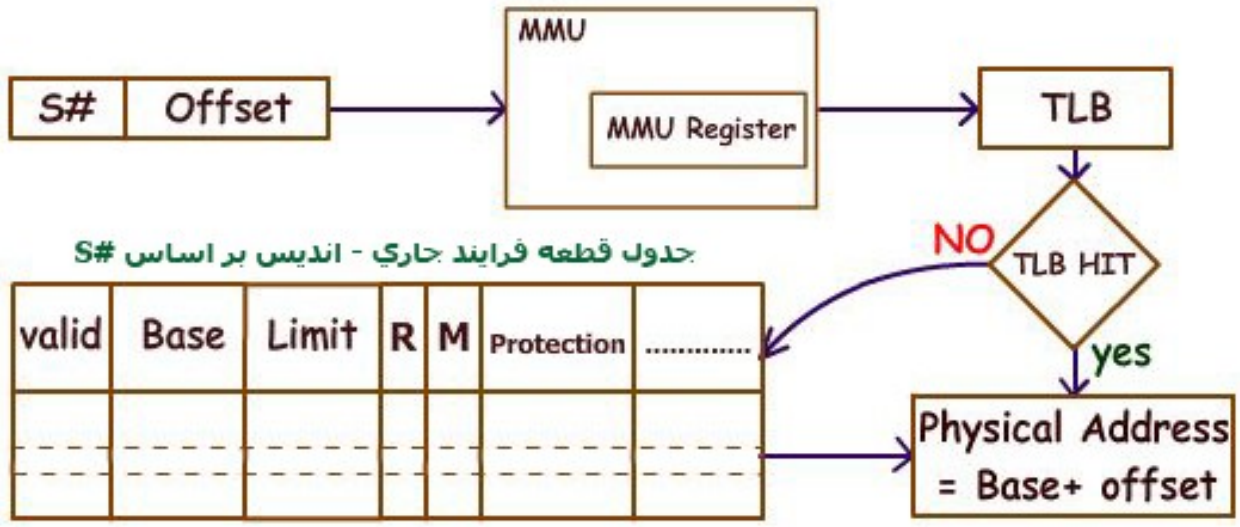
قطعه بندی به روایت دکتر حقیقت:

- ساده ← کل برنامه باید بیاید.
- حافظه مجازی ← کل فرایند لازم نیست بیاید.

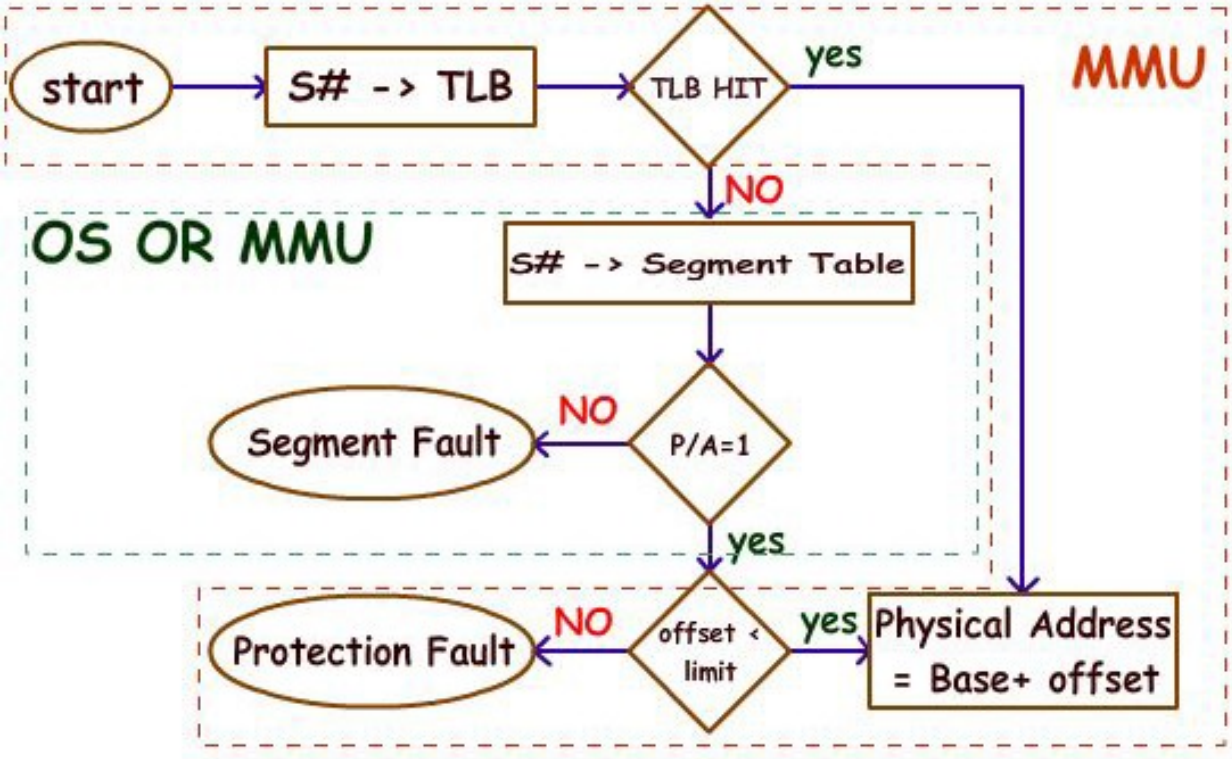
آدرس مجازی از دو قسمت Offset و Segment تشکیل می شود.

Segment# =12 bit	Offset = 20 bit

اگر 20 بیت به آفست و 12 بیت به segment اختصاص پیدا کند هر فرآیند حداکثر قطعه می‌تواند داشته باشد. هر قطعه نیز می‌تواند حداکثر 2^{20} بایت باشد.



در شکل قبل، در جدول قطعه فرایند چون سایز قطعات متفاوت است نمی‌توانیم شماره فریم داشته باشیم که در صفحه بندی استفاده می شد، بلکه باید آدرس Base و Limit قطعات را در جدول نگهداری کنیم.



35:30

هرگونه Fault ی که رخ دهد برعهده سیستم عامل است که این تله را بررسی ورفع کند.

39:40

ترکیب صفحه بندی با قطعه بندی

می‌خواهیم مزایای صفحه بندی و قطعه بندی را با هم داشته باشیم برای این کار قطعه ها را صفحه بندی می‌کنیم. اول برنامه نویس فرایند خود را به قطعاتی تقسیم می کند آنگاه سیستم عامل و سخت افزارهر قطعه را صفحه بندی می کند.

- در صفحه بندی p/2 بازای هر فرایند اتلاف حافظه داشتیم اینجا p/2 بازای هر قطعه
- هر فرایند یک جدول قطعه دارد وهر قطعه هم یک جدول صفحه دارد.

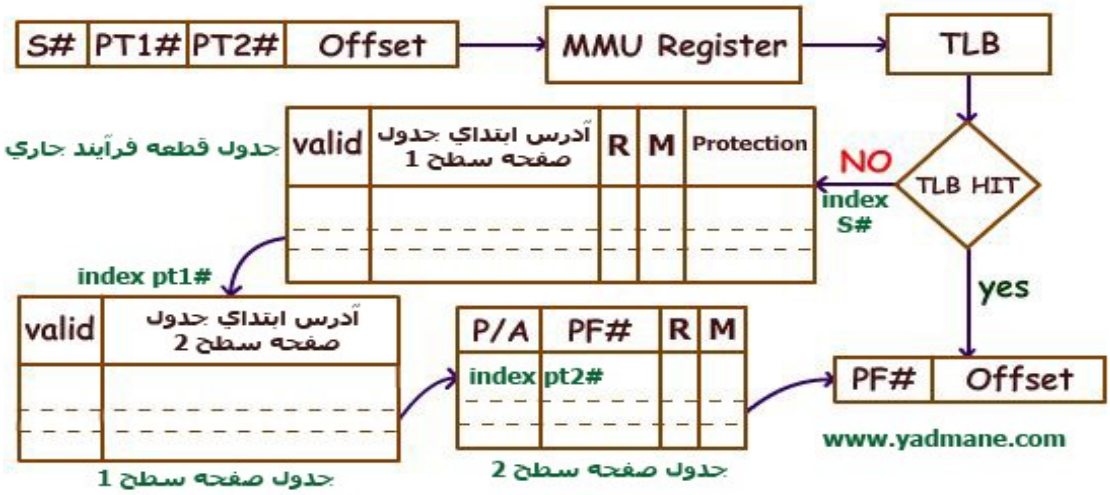
آدرس مجازی در ترکیب قطعه بندی و صفحه بندی :

در آدرس مجازی ترکیب قطعه بندی و صفحه بندی دیگر base و limit نداریم چون دیگراین قطعات نیستند که در حافظه بار می‌شوند بلکه صفحات در حافظه بار می شوند.

S#	P#	Offset

ترکیب قطعه بندی و صفحه بندی با جداول دو سطحی

مثال- ترکیب قطعه بندی و صفحه بندی با جداول صفحه دو سطحی:



55:30

چند قانون:

فرض کنید تعداد بیت‌های VA مطابق شکل زیر باشد:

S#	P#	P#	Offset
n بیت	p بیت	q بیت	k بیت

همچنین :

$A = n + p + q + k$

آنگاه داریم:

1. اندازه VM برابراست با $2A$ ← حداکثر اندازه یک فرایند.
2. حداکثر تعداد قطعات یک فرایند 2^n
3. حداکثر اندازه قطعه: 2^{p+q+k}
4. اندازه هرصفحه: 2^k
5. هرفرایند حداکثر 2^{n+p+q} صفحه دارد.
6. هرقطعه حداکثر 2^{p+q} صفحه دارد.
7. هر فرایند یک جدول قطعه دارد.
8. هر فرایند حداکثر 2^n جدول صفحه سطح یک دارد.
9. هر قطعه یک جدول صفحه دارد.
10. هر قطعه حداکثر 2^p جدول صفحه سطح دو دارد.
11. هر فرایند حداکثر 2^{n+p} جدول صفحه سطح دو دارد.
12. جدول قطعه 2^n درایه دارد.
13. جدول صفحه سطح یک 2^p درایه دارد.
14. هرجدول صفحه سطح دو 2^q درایه دارد.

1:15:30

محاسبه زمان های دسترسی به حافظه :

مثال۱: فرض کنید یک سیستم صفحه بندی با جداول ساده با زمان دسترسی به حافظه 10ns و زمان دسترسی به 1ns TLB مفروض است . اگر نسبت اصابتTLB یا TLB hit ratio برابر0.8 فرض شود . زمان ترجمه آدرس و زمان کل دسترسی به یک کلمه در حافظه را بدست آورید.

زمان ترجمه آدرس:

$H_{TLB} * T_{TLB} + (1-H_{TLB})(T_{TLB}+T_{Mem}) = T_{TLB} + (1 – H_{TLB}) T_{mem}$

$T_{Translate} = 1ns + (1 – 0.8)\ 10\ ns = 3ns$

$T_{کل} = T_{ترجمه} + T_{mem}$

$T = T_{Translate} + T_{mem} = 3ns + 10ns = 13ns$

مثال۲- در سئوال قبلی جداول صفحه را دو سطحی فرض کنید.

زمان ترجمه آدرس:

$T_{TLB} + (1 – H_{TLB})*2\ T_{mem}$

$T_{Translate} = 1ns + (1 – 0.8) *2*10\ ns = 5ns$

$T_{کل} = T_{ترجمه} + T_{mem}$

$T = T_{Translate} + T_{mem} = 5ns + 10ns = 15ns$

1:29:30

مثال۳: در مثال 1 فرض کنید حافظه مجهز به Cache نیز هست . هم داده ها وهم جداول صفحه می توانند درCache باشند.(اگر بگویند جداول صفحه حتماً در Catch می‌باشد نحوه محاسبه متفاوت می باشد) احتمال اصابت Cache یا hcache معادل 0.9 می‌باشد ، زمان دسترسی به Cache را 3ns فرض کنید.

$T_{mem-Cache} = T_{cache} + (1 – H_{cache})\ T_{mem} = 3\ ns + (1-0.9) * 10\ ns = 4ns$

زمان ترجمه آدرس:

$T_{TLB} + (1 – H_{TLB})\ T_{mem-cache} = 1ns + (1 – 0.8) * 4ns = 1.8ns$

اگر قید نکرده بود که جداول صفحه می‌تواند در Cache باشد و جداول حتماً در Cache بودند به جای 4 در فرمول بالا 3 باید می گذاشتیم.

$T_{کل} = T_{ترجمه} + T_{mem-cache} = 1.8ns + 4ns = 5.8ns$

مثال ۴: در مثال بالا فرض کنید که احتمال PF =0.01 باشد و زمان بارگذاری صفحه از روی دیسک 1ms باشد زمان کل دسترسی چقدراست . فرض می کنیم جداول صفحه روی دیسک نیست .

زمان ترجمه آدرس:

$T_{TLB} + (1 - H_{TLB}) T_{mem-cache} = 1ns + (1 - 0.8) * 4ns = 1.8ns$

$T_{کل} = T_{ترجمه} + [T_{mem-cache} + 0.01 * T_{Disk}] = 1.8ns + [4ns + (0.01 * 10^6)] = 10005.8\ ns$

ساعت15:55