Subject:
Year. Month. Date. ( )

در بن‌بست هستیم یا نیستیم؟

انتخاب P_1 یا P_3 چون هر دو صفر هستند

		A	B	C		
انتخاب	P_1 ←	0	1	0	available	
"	P_3 ←	3	1	3		
"	P_2 ←	5	1	5		
"	P_4					
"	P_5					

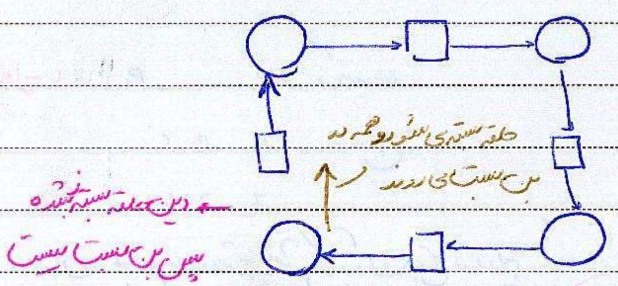
پس System در بن‌بست نیست، همگی True شدند

2. Request $_3$ = 0 0 1 می‌رسد ← آیا سیستم در حالت امن؟ (جدول از صفحه قبل)

ماتریس request در P_3 0 0 1

انتخاب P_1 0 1 0 ← بعد هیچ کدام را نمی‌توانیم به آخر برسانیم پس بن‌بست می‌رسیم

← P_2 و P_3 و P_4 و P_5 ← در بن‌بست هستند



می‌دانیم بن‌بست وجود دارد که در بن‌بست است می‌توانیم از این اطلاعات برای شکستن حلقه استفاده کنیم

← سیستم خلوت ← ready Queue ← خلوت‌تر است

" شلوغ ← " " " ← دارای request زیادتر است

الگوریتم بن‌بست را کدام باید بیشتر اجرا کنیم؟

در سیستم خلوت، در حالی که شلوغ است اصلاً وقت بررسی را ندارند

← شاید دلیل خلوت شدن بن‌بست باشد پس منطقی است که این جا بیشتر اجرا شود

PAPCO

Subject:
Year. Month. Date. ( )

جلسه بیست و یکم (دوم) 12، 9، 87

تمرین OS برای جلسه آینده

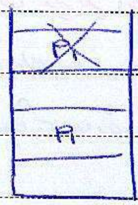
مدیریت حافظه: memory management

← تحویل تمرین‌های بن‌بست ← تحویل مرحله دوم ← پنج‌شنبه 28، 9 ← ساعت 8

تخصیص فضای حافظه، آزادسازی حافظه، پیگیری فضاهای استفاده شده، پیگیری فضاهای آزاد

← کارهای مدیریت حافظه

جابجایی relocation



هر فرایند را می‌توانیم در هر جای حافظه به آزادی قرار دهیم

← کد، Data، پشته، PCB ..... ← یعنی همه اینها رفته داخل حافظه

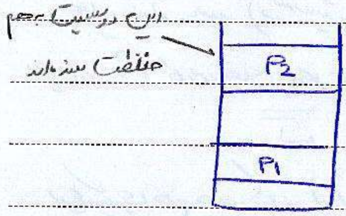
این‌ها در حافظه قابل جابجایی هستند چون در هر جای حافظه می‌توانیم قرار دهیم

ولی هر چیزی را می‌شود جابجا کرد چون ممکن است در کد آدرس دقیق به کار برده باشیم و در برنامه هم مثل ما می‌خواهیم کمتر از

شاید بیشتر استفاده کنیم (نسبت به Base address)

تا کجا به جای اول ← از مقادیر مطلق آدرس استفاده نشود

حفاظت protection



فرایندها نباید به فضاهای یکدیگر دسترسی داشته باشند

این محل‌ها محافظت کنند

اگر این‌ها بخواهند از یک مشترک استفاده کنند امکانش هست

می‌توانند از بعضی چیزها نه همه چیز را بطور مشترک استفاده کنند ولی محافظت شده

اشتراک sharing ← از این طریق می‌توانند هم data رد و بدل کنند

← حافظه دارای سازمان ← logical organization

physical "

Subject:
Year. Month. Date. ( )

باید این دو تفاوت قائل بشویم

در تخصیص حافظه ممکن است پیوسته نباشد و هر بخش فرایند یکی باشد ← فیزیکی

ولی در سازمان منطقی به نظر می‌رسد که پشت سر هم است

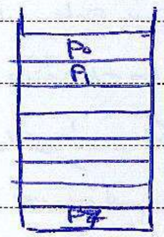
چیزی که واقعاً هست ← physic

چیزی که به نظر می‌آید که هست ← logical

روش‌های مدیریت حافظه

۱. روش ایستا (static

در این روش حافظه از قبل تقسیم بندی شده ← بخش بندی ثابت (partitioning)



← هر بخش‌ها مساوی هستند و حداکثر می‌توانیم 8 فرایند داشته باشیم که فضاهایش هم محدود است

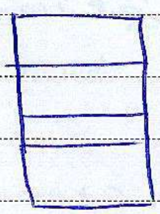
این روش خیلی محدود کننده ولی خیلی ساده است

در سیستم‌هایی که کاربرد عمومی دارند نمی‌توان از این استفاده کرد چون ممکن برنامه‌ها زیاد باشند یا اینکه یک برنامه خیلی بزرگ باشد (در سیستم‌های general purpose)

این روش کاربردهای خاص دارد که تعدادشان هم زیاد هست ← روش خیلی ساده مدیریتی دارد ← 8 تا دارد که این فضاها مال کی هست

← سیستم‌هایی که کاربرد خاص دارند (embeded

نوع دیگری هم هست که اندازه partition ها ثابت است ولی اندازه‌هایشان با هم فرق دارد



بخش بندی ثابت

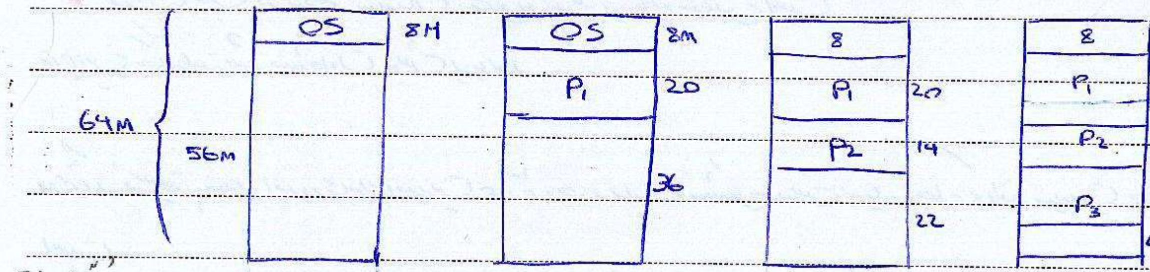
اندازه‌های متفاوت

PAPCO

Subject:
Year. Month. Date. ( )

۲. روش پویا Dynamic

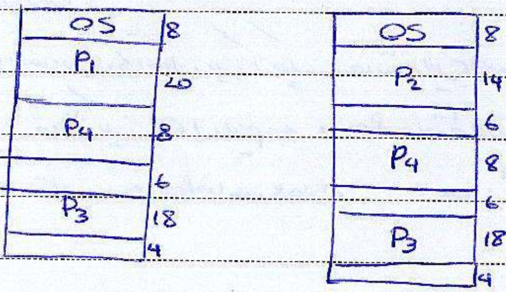
بخش بندی ها به طور متغیر است ← فضای خالی به صورت یک فضای پیوسته است که به هر صورت می توان آن را بخش بندی کرد.



مثلاً فرزندان معلق می کنیم

افزایش زمان →

در این جا فرایند P4 می آید و فضای 8M می خواهد ← نداریم پس P4 منتظر می شود ← پس از مدتی فرایند P2 را معلق می کنیم و فضای P2 خالی می شود، حالا که می خواهد P2 دوباره وارد شود می بینیم که P1 تمام شده و فضایش آزاد است



الان اگر فرایندی بیاید که 16M فضای حافظه بخواهد ← حافظه نداریم (پشت سر هم) ← ولی داریم که پیوسته نیست ولی نمی توانیم به آن بدهیم

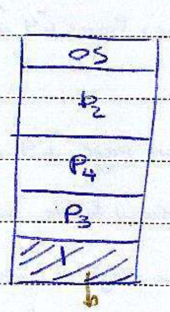
این پدیده که هر چه جلو برود باعث می شود حافظه بیشتر تکه تکه شود fragmentation می گویند

fragmentation →

← فضای حافظه داریم ولی به درد ما نمی خورد

برای حل این مشکل ← هر چند وقت یکبار عمل فشرده سازی انجام دهد ← یعنی همه را به یک سمت منتقل کند تا فضای پیوسته آزاد شود

compaction



internal fragmentation

این کار به سختی انجام می شود ولی این جا ارزش ندارد ولی این جا بیشتر ارزش دارد

← عیب [illegible] و مشکل [illegible] [illegible] [illegible] زمان زیادی می گیرد

→ فضای خالی پیوسته می شود

هزینه زیاد است ← مقدار زیادی از اطلاعات که باید جابجا منتقل کنیم

زمان زیاد (زمان انتقال)

← Fragmentation ← در سرعت دیسک تاثیر دارد چون فضاها پشت سرهم قرار ندارد هرچی فاصله فضاها در دیسک زیاد باشد سرعت کمتری میشود ولی در حافظه تاثیری ندارد چون عنصری الکتریکی است

← پس در حافظه کلی از این روش استفاده می کنند

* در این بحث به کار می‌بریم سازمان منطقی و فیزیکی برهم منطبق است.

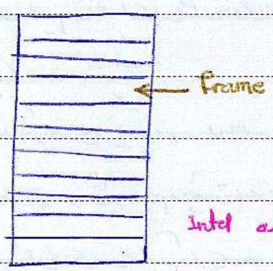
تا این جا گفته شد مشکلات روش پویا بود.

روشی که در سیستم های امروزی برای مدیریت حافظه وجود دارد و مشکل روش پویا را حل می کند مدیریت حافظه paging است.

← مهمترین روش مدیریت حافظه

مدیریت حافظه صفحه بندی Paging

← از نوع روش های پویا است.



حافظه از نظر OS :

قسمت های مساوی ولی خیلی ریز و کوچک (حافظه به این صورت تقسیم بندی شده)

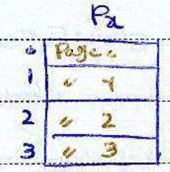
مثال frame : حافظه یک 1G داریم ← frame ها می توانند 4K باشند

Intel ← مربوط به پردازنده

$\frac{2^{30}}{2^{12}} = 2^{18} \Rightarrow$ 256K تعداد frame ها

اندازه frame ها از پیش تعیین شده است.

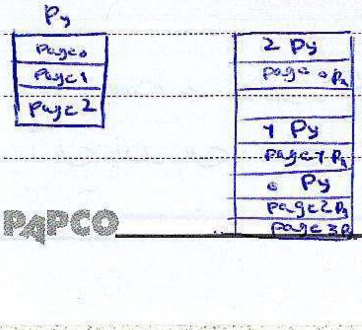
← نسبت به حافظه خیلی ریز است به طوری که در آن هیچ فرایندی جا نمی شود



فرض کنیم فرایندی داریم به فضایی می خواهد اگر واحد را frame بگیریم تقسیم بندی کنیم ← مثلا 4 تا frame فضای می خواهد

پیوستگی (فضای پیوسته)

پس فرایند 4 تا page است ← می خواهیم این را ببریم در حافظه ← لزومی ندارد حتما frame پیوسته پیدا کنیم



بلکه هر frame به جای هر page می تواند قرار گیرد

هر page در هر frame می تواند قرار گیرد

پس دیگر نیازی به fragmentation نداریم

Subject:
Year. Month. Date. ( )

در این جدول مسئله پیوستگی فضا از بین رفت پس دچار مشکل پرش بوده نمی‌شود هرجا frame خالی ببینی می‌توانی از آن استفاده کنی.

← پیوستگی را از بین بردیم پس چطور اجرا می‌شود از Page ها به هم Jump می‌کند؟

سوال هایی که پیش می‌آید

← اختصاص دادن frame

← آزاد بودن frame

← وضعیت پر بودن حافظه؟ اگر حافظه اشغال شد یک فرآیند دیگر چطور باید کار کنیم

می‌توانیم بخش های از فرآیندی که دارد اجرا می‌شود که الان به درد نمی‌خورد از حافظه خارج کنیم

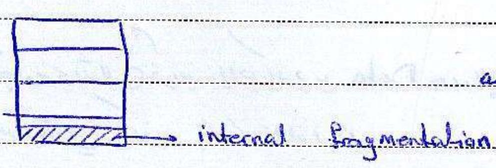
← آیا لازم است کل فرآیند در حافظه باشد ← الان فرض می‌کنیم که کل فرآیند در حافظه است ولی بعدا این فرض را هم حذف می‌کنیم

Fragmentation

خود paging به گونه‌ای fragmentation ایجاد می‌کند

هر فرآیند را که الان به page ها تقسیم کردیم لزوما page آخرش استفاده شده نیست و دارای بخش خالی باشد یعنی page آخر خالی داشته باشد

این فضاهای خالی که fragmentation است ولی قابل استفاده نیست به هیچ وجه ولی در عمل مقدارش خیلی کم است پس می‌گوییم اشکال ندارد



در داخل فرآیند اتفاق افتاده

← OS از کجا بداند هر frame مال کی هست ← یکی باید این را بداند ← OS برای هر فرآیند جدول صفحه درست می‌کند و از این می‌فهمد که page ها کجا هست؟

جدول صفحه Px

Page	frame
0	3
1	6
2	8
3	9

→ page 0 من کجاست

Py

Page	frame
0	6
1	4
2	5

پس بفهمید هر page کجاست

PAPCO

Subject:
Year. Month. Date. ( )

حالا سیستم عامل از کجا می فهمد این جدول کجاست ← pointer (آدرس شروع) جدول را در PCB نگه می دارد

برای حافظه آدرس شروع مهم است و ممکن است کل جدول در حافظه نباشد

OS همیشه می داند کدام frame ها در حافظه خالی هست.

جلسه بیست و پنجم 87/9/17

امتحان یکشنبه 1/10/87 ساعت 12:30 تا 13:30

مدیریت حافظه صفحه بندی Paging

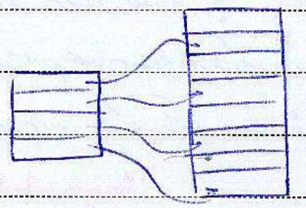
حافظه به frame ها تقسیم شده

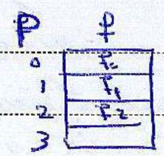
فرایندها را به page ها تقسیم می کنیم

که اندازه frame = page و در حافظه قرار می دهیم

برای اینکه بدانیم page ها کجاست از یک جدول استفاده می کنیم

← جدول صفحه





ترتیب frame ها مهم نیست ← مشکل فضای پیوسته وجود ندارد

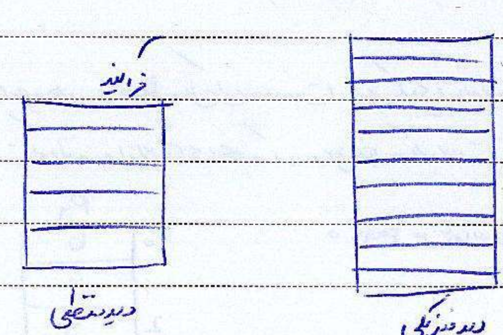
اگر frame ها تمام شدند باید یکی را بیرون کنیم

پیوستگی از بین رفته ← چطور می توان فرایند را اجرا کرد یا Data ها را پیوسته دید

پردازنده این ها را به پیوسته تبدیل می کند ← کمک افزاری

دید فیزیکی به منطقی تبدیل شود

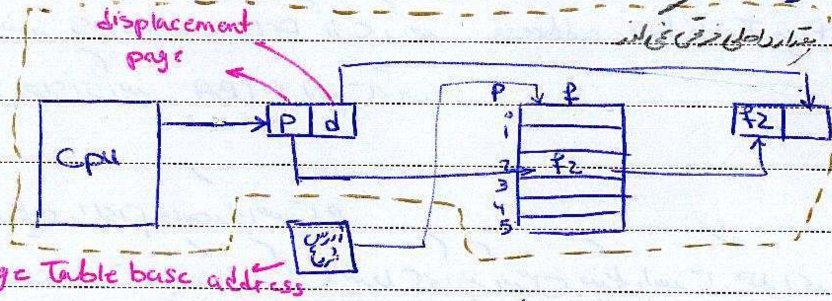
یعنی آدرس منطقی به آدرس فیزیکی



PAPCO

این مکانیزم عملکرد سخت افزاری است و اگر نرم افزاری باشد خیلی سرعت برنامه کم می شود.

تبدیل آدرس پیوسته به آدرس فیزیکی →

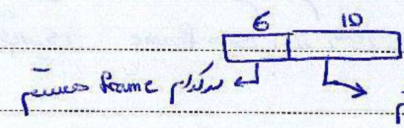


← CPU در اجرای برنامه‌ها آدرس تولید می‌کند به حافظه می‌دهد
مربوط به Instruction
مربوط به Data

فرض می‌کنیم آدرسی که برنامه‌نویس می‌بیند منطقی است ← پیوسته
در عمل CPU آدرس فیزیکی تولید می‌کند

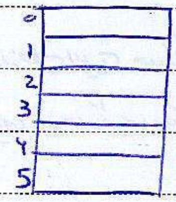
برای حافظه 64K ← 16 بیت آدرس داریم.
64 تا frame داریم 1K => هر frame

هر آدرسی که تولید می‌کنیم می‌خواهیم در کدام frame می‌افتد ← هر آدرس



آدرس ← به طور منطقی نگاه می‌کنیم یعنی 16 بیت را می‌بینیم
به این صورت نگاه می‌کنیم → وقتی نگاه می‌کنیم یک 6 بیت یک 10 بیت
کدام page
کجای page

باید page در کجای حافظه است ← از جدول صفحه مربوط به فرآیند استفاده می‌کنیم
صفحه پیوسته است ولی در حافظه پیوسته نیستند
آدرس‌هایی که می‌بینیم پیوسته است ولی چیزی که به حافظه داده می‌شود ناپیوسته است.



OS کمک افزار را باید به روز خودش کند ولی نشست این کار هم همین کار را کرده در حین اجرا صفر شمار ولی کار خود را انجام داده و تاییدی خود را گذاشته

← این کار کاملاً مخفی است و هیچ کس نمی‌بیند ولی OS می‌داند جدول صفحه خودش را درست کرده ولی در حین اجرای فرآیند OS وجود ندارد.

Subject:
Year. Month. Date. ( )

point) هر فرایند جدول مخصوص خود را دارد و فرایندها به محدوده هم دسترسی ندارند

وقتی که فرایندها عوض می‌شوند OS با توجه به PCB آن فرایند Page Table base address را تغییر می‌دهد. OS قبل از اجرای فرایند PTBA را ست می‌کند

← سخت‌افزاری که این تبدیل را انجام می‌دهد چیست؟

سخت‌افزار داخل CPU است و چیزی که از آن بیرون می‌آید آدرس فیزیکی است و همه‌ی سخت‌افزارها داخل CPU هست ولی PTBA در CPU نیست جدول در حافظه است.

اندازه هر Page را طراح CPU تعیین کرده و اختیاری نیست و ما اختیاری نداریم

مشکلات حل شده:

- دیگر Fragmentation نداریم (داریم به طور مختصر و در آخرین Page هر فرایند)
- مشکل پیدا کردن فضای حافظه را نداریم

مزایا:
- قابلیت انتخاب هر frame ایجاد شده
- جابجایی frame های آزاد آسان شده ← به وسیله صفر یک کردن حافظه‌ها مثلاً اگر حافظه 64K باشد یک رشته 64 تایی می‌تواند این کار را انجام دهد.

اندازه Page از کجا آمده؟ دو نکته باید در نظر بگیریم

1. اندازه جدول صفحه ← اندازه Page کوچک باشد جدول صفحه خیلی بزرگ می‌شود
2. تکه تکه شدن یا fragmentation ← بخشی از حافظه از دست می‌رود و این مقدار خیلی قابل توجه می‌شود

بعد از حل این مشکلات یک مشکل جدید بوجود می‌آید: ما قبلاً مشکل مدیریت حافظه داشتیم و به ازای هر آدرس یک بار به حافظه می‌رفتیم ولی الان به ازای هر آدرس دو بار باید برویم ← یکی برای page Table
یکی برای آدرس اصلی

این مسئله خیلی سرعت را پایین می‌آورد ← حافظه گلوگاه CPU است و سرعت را کم می‌کند

این سیستم کارایی بسیار پایینی دارد
performance

PAPCO

سیستم ها مدیریت حافظه خیلی هم نیست، سرعت است که تعیین کننده است

بیایید مشکل را حل کنیم:

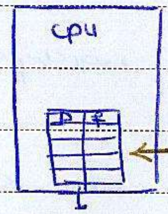
حل مشکل سرعت پایین:

آدرس هم مستقیم به حافظه نمی رود بلکه اول به سراغ cache می رود و در صورتی سراغ حافظه می رود که در cache نباشد

از همین ترفند یعنی cache استفاده می کنیم

که بخش کوچکی از حافظه را دارد درصد $\frac{30}{100}$ یا $\frac{1}{100}$ ولی همیشه اول cache را چک کنیم

برای حل مشکل داخل خود پردازنده جدولی درست کنیم که بخشی از page table درآن باشد از نوع cache که خیلی کوچک می شود)



اسم جدولی که برای حافظه پردازیم برای cache یا جدول TLB

بردیم سرعت با خوب می شود سرعت بالا است مثل reg

مشکلش این است که اندازه اش کوچک است

TLB: Translation Lookaside buffer (بافر دم دستی ترجمه)

که برای cpu دم دستی است.

در پردازنده Intel ← 32 تا سطر دارد ولی کیفیت بالاست براحتی می تواند 1000 تا page داشته باشد

در TBL ← 98% آدرس ها جواب می دهد و تنها در 2% مجبور است سراغ page table برود

که اکثر دسترسی ها با سرعت بالای انجام می شود

TLB به OS هیچ ربطی ندارد و OS اصلاً نمی داند که TBL اصلاً وجود دارد و با جدول صفحه کار دارد

کلش در سخت افزار انجام می شود

برای تبدیل آدرس (سخت افزار)

1. آزمایش TLB ← اگر موجود بود تبدیل انجام می شود

2. اگر موجود نبود جدول صفحه و TLB را update کردن ← یعنی اینها همزمان پیدا کردن می رود در TLB

Subject:
Year. Month. Date. ( )

← TLB از دید OS مخفی است ← سخت افزاری میبیند

← Paging از دید برنامه کاربردی مخفی است ← OS می بیند

(سهم زمانیش شروع شود)

یک نکته) فرض کنیم تا الان P2 کاری کرده الان P1 به کار بیفتد داخل TLB چی هست؟
هیچی چون هرچی داخلش بوده مربوط به P2 است و کلاً پاک می شود و اولین بار به هر آدرسی که ارجاع
می کنیم در TLB نیست ولی همین که نیست چون این آدرس در داخل PLB قرار می گیرد

← با شروع هر فرآیند (شروع سهم زمانی) TLB خالی است و TLB به تدریج پر می شود.
← مجبوریم کار خود را از طریق جدول صفحه انجام دهیم

به هر صفحه‌ای در جدول صفحه مراجعه کنیم که در TLB نباشد ← می آید داخل TLB

خود OS هم برای خود page table دارد ولی معمولاً این چیزی که در حافظه قرار می گیرد پیوسته است.

مثال) سهم زمانی 10ms

1 billion Instruction / sec ← یک بیلیون دستورالعمل در ثانیه اجرا می شود

$\frac{10^9}{10^2}=10^7$ ← در هر سهم زمانی 10^7 بار تا Instruction اجرا می شود

← به تعداد مراجعات (این تعداد از این بیش‌تر است چون ممکن است بیش از یک بار دسترسی داشته باشد)

TLB در کامپیوترهای بیرون تقریباً چقدر است؟

این نوع ذخیره سازی در حجم بالا [illegible] data جستجو

content Addressable Memory → مثل cache

اطلاعات در TLB را چطور پیدا کنیم؟

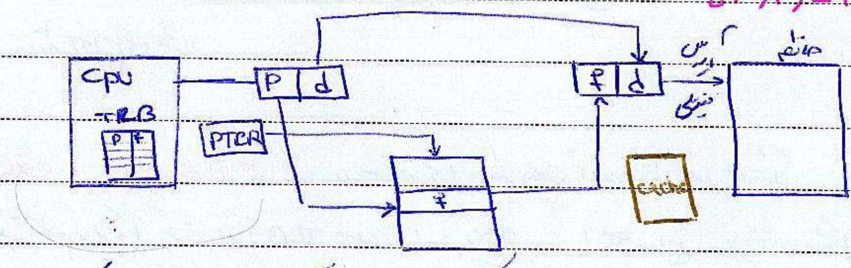
2		
4		
5		
1		
8		
.		

به ترتیب نیست چون تنها page های
که به آن دسترسی پیدا کردیم در TLB است
هیچ راهی جز این نیست که تمام TLB را چک کنیم و به ازای
هر دسترسی باید کل TLB را چک کنیم ← 32 تا مقایسه کننده داشته باشیم که (سخت افزاری)
با هم موازی کار کنند ← به تعداد خانه‌ها
← یکی از دلایل کم بودن خانه‌ها همین است.

PAPCO

Subject:
Year. Month. Date. ( )

جلسه بیست و پنجم 87,9,24



یکی از reg های داخلی پر بار شده که از PCB load می‌شود که آدرس جدول صفحه می‌باشد. چون مشکل سرعت پایین داشتیم از TLB استفاده کردیم.

مثال ۱) حافظه: زمان دسترسی 50ns

TLB „ „ 2ns

hit ratio (نسبت اصابت): 98%

میانگین زمان دسترسی را حساب کنید (به حافظه)

Teff

چون همیشه دسترسی به حافظه زمان یکسانی ندارد ← بعضی وقت‌ها از TLB بعضی وقت‌ها از طریق جدول صفحه

$T_{eff} = 0.98(2ns + 50ns) + 0.02(2ns + 50ns + 50ns)$

اول بررسی TLB که در هر دو حالت انجام می‌شود — جدول صفحه — خود حافظه

$T_{eff} = 51ns + 2ns = 53ns$

اگر فقط یک دسترسی به حافظه داشتیم 50ns وقت می‌خواستیم ولی الان فقط 3 ns اضافه شده کمتر از 10% است و این خوب است تا 10% هم خوب است معمولاً 10% را حفظ در نظر می‌گیرند

← معمولاً سیستم‌های واقعی این طور نیستند، حافظه خیلی کند است و باید راهی پیدا کنیم این را تندتر کنیم (50ns در مقابل 2ns خیلی زیاد است، سرعت حافظه نسبتاً خیلی کند است)

برای حل این مشکل از حافظه کش یا cache استفاده می‌کنیم. cache سرعت بالایی دارد به قیمت گران که اصلاً با حافظه قابل مقایسه نیست. بخشی از حافظه را در cache قرار می‌دهیم (به جای بخش حافظه کاری مثل کد، data، Table ... پشته) مبنای کار با حافظه خیلی است، امیدواریم بیشتر وقت‌ها آن چیزی که می‌خواهیم در cache پیدا کنیم.

PAPCO

Subject:
Year. Month. Date. ( )

← ساختار cache ، سخت افزاری است و OS از وجودش بی‌اطلاع است و OS با حافظه کاری کند و تمام مدیریت‌ها روی حافظه انجام می‌شود

حافظه cache (حافظه نهان): زمان دسترسی خیلی بهتر از حافظه است مثلاً 10ns

نسبت اصابت‌ها هم چیزی که کمتر از TLB است: 90% ← 90% مواقع در cache مشکل حل می‌شود.
برای کمک حافظه دیگر بهتر مدیریت حافظه ندارد ← در اصل بهتر از این است.

cache با حافظه دیگر برنمی‌شود و در اصل می‌خواهیم بگوییم 50ns به چی تبدیل می‌شود.

حل مثال قبل با استفاده از cache ← زمان دسترسی به حافظه ... خیلی تقریبی

$T_{cache} = 0.9(10ns) + 0.1(10ns + 50ns) = 9 + 6 = 15ns$

زمان دقیق از این بیشتر است و خیلی چیزها را در نظر نگرفته‌ایم

$T_{eff} = 0.98(2ns + 15ns) + 0.02(2ns + 15ns + 15ns) = 17.3\ ns$

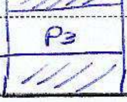
تأثیر cache در جایی است که به حافظه دسترسی داریم. عدد واقعی از 17.3ns بیشتر است چون بعضی چیزها را در نظر نگرفتیم:
ممکن است در cache چیزی را پیدا نکنیم می‌رویم سراغ حافظه. این طور نیست بلکه اگر در cache پیدا نکردیم از حافظه به cache می‌آوریم و در اصل ما ارتباط بین cache و حافظه اصلی را در نظر نگرفتیم که این زمان بر است. CPU با حافظه رابطه مستقیم ندارد و از cache به حافظه ارتباط دارند.

cache معمولاً دو level دارد ← L1 cache داخل CPU ← سرعت خیلی بالا و ظرفیت خیلی کم
L2 cache روی board اصلی

این زمان دسترسی به حافظه هنوز هم خوب نیست یعنی همچنان نسبت به CPU سرعت پایین‌تری دارد و هنوز هم گلوگاه سیستم است

مدیریت حافظه به روش paging که مشکلات fragmentation و مدیریت را حل کنیم

← همه فضاهای خالی را OS باید حساب و کتاب کند و داشته باشد.



S_1 b_1, l_1 S_2 b_2, l_2

base ← آدرس شروع ، طول فضای خالی S_3 b_3, l_3

P_4 وارد می‌شود و فضای خواهد و ما بیشتر از یک choice داریم

PAPCO

می‌توانیم هر سه تا از این فضاهای خالی را به آن بدهیم. فضاهای خالی را باید search کنیم و ببینیم تمام فضاهای خالی

از همه به P4 نزدیک‌تر است که برای این کار باید کل فضاهای خالی را search کنیم که این search خیلی طول

می‌کشد ← روش best fit (بهترین برازش) search طولانی و بهترین انتخاب

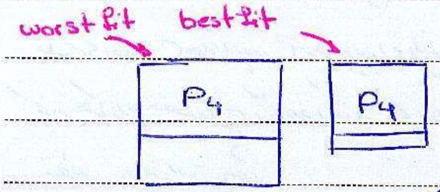
روش دیگر این است که به جای search کردن به اولین جایی که رسیدیم که P4 می‌توانست در آن قرار گیرد آن را

انتخاب می‌کنیم با این کار search را کوتاه‌تر کردیم first fit (اولین برازش)

← جستجوی کوتاه و بدون بهترین انتخاب

روش دیگری هم هست ← کل بیشترین را پیدا می‌کند search می‌کند و بزرگترین را پیدا می‌کنیم یعنی اینکه

از اول sort کردیم ← روش worst fit بدون جستجو



منطق worst fit این است: اگر بزرگترین فضا را برای قرار دادن

فرایند استفاده کنیم فضایی که باقی می‌ماند هم به اندازه‌ی کافی بزرگ است که

بتوان برای فرایند دیگر دوباره آن را استفاده کرد

روش اثبات اینها از benchmarking است ← روش و تحلیلی دقیق کننده است.

← مجموعه‌ای از برنامه‌های واقعی که زیاد استفاده می‌شوند.

روی این برنامه‌ها چیزی که می‌خواهیم را تست کنیم اجرا می‌کنیم و اندازه‌گیری می‌کنیم و ببینیم کدام یک از حافظه بهتر می‌تواند

استفاده کند ← مقدار هدر

← مقدار هدر رفته حافظه ← بیش از 1/3 حافظه هدر می‌رود.

با این روش فهمیدند best & first بهتر از worst کار می‌کنند

1/3 حافظه تلف می‌شود

روشی که بیشتر از همه استفاده می‌شد روش first fit است.

← paging تمام مسئله fragmentation را حل می‌کند ولی مدیریت پیچیده فضاهای پیوسته را حل می‌کند و یک

مدیریت ساده می‌آورد. دیگر احتیاج به فضای پیوسته نمی‌خواهیم

مشکل paging که مزیتش بود با ترفندهایی حل کردیم و مدیریت غالب در سیستم‌های امروزی است

مدیریتی که الان گفته می‌شود در پردازنده Intel از آن استفاده شده. مدیریت حافظه به پردازنده چسبیده است چون

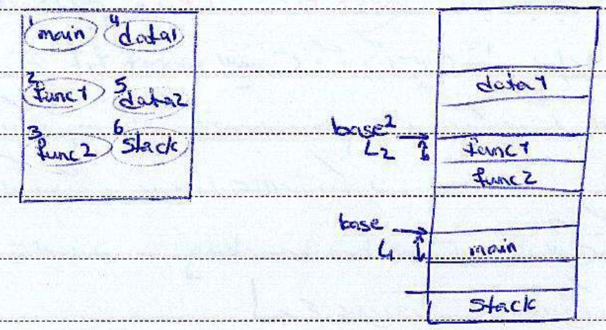
تمام تبدیلات را CPU باید انجام دهد باید پشتیبانی cpu را مدیریت حافظه داشته باشد ← segmentation

جلسه بیست و پنجم ۹,26

مدیریت حافظه، قطعه بندی Segmentation

این مدیریت‌ها مخصوص OS است و در اصل OS این کار را انجام می‌دهد ولی چون باید پشتیبانی پردازنده را داشته باشد می‌گوییم CPU این اشکال را دارد.

← صفحه بندی از دیدگاه برنامه کاربر مخفی است ولی بر عکس قطعه بندی با دیدگاه برنامه‌نویس انجام می‌شود
همین دیدگاه را می‌توان در مدیریت حافظه به کار برد. می‌توان این برنامه را به چند قطعه تقسیم کرد



نیازی نیست مکان‌های پشت سر هم پیدا کنیم که هر کدام مستقل نگه می‌کنیم ولی هر قطعه باید پیوسته باشد

این نوع مدیریت هم‌چنان مشکل fragmentation را دارد ولی کمتر شده چون در این هم فضاهای کوچک نیز داریم

جدول قطعه
Segment table

1	base1	L_1
2	base2	L_2
3		
4		
5		
6		

OS از روی Seg Table می‌فهمد این قطعه‌ها کجا است.
هر قطعه یک base و یک طول دارد و در این جا می‌توانیم
از شماره گذاری قطعه‌ها استفاده کنیم چون طول قطعه‌ها ثابت نیست

این مدیریت نسبت به paging سخت‌تر است. باید حساب کتاب فضاهای آزاد را داشته باشیم و باید برای پیدا کردن فضای خالی بیشتر بگردیم.

مدیریت حافظه 8086 ← پردازنده Intel

شکل مدیریت Segmentation است ولی خیلی ساده‌تر. باید مشخص کنیم که کجاش code seg، stack seg، data seg ... است و این را ما تمرین می‌کنیم ← این خیلی نوع ساده‌ای است چون Seg های محدودی دارد.
که نوع کامل‌تر این است که بتوانیم هر چند تا قطعه که می‌خواهیم داشته باشیم.

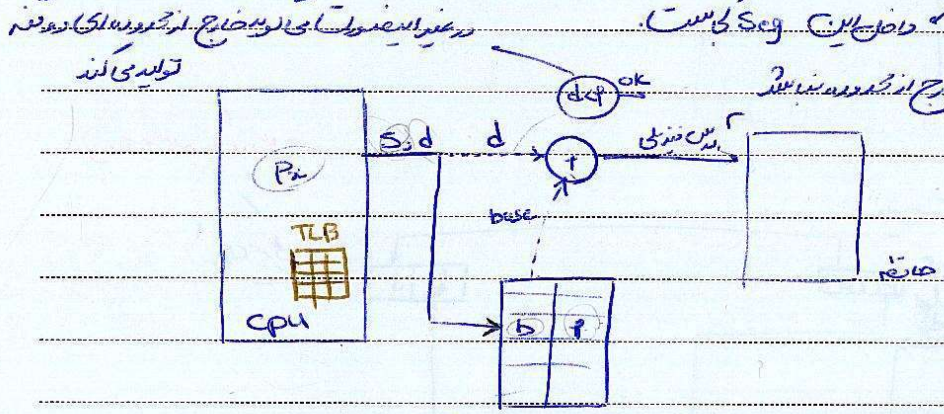
Subject:
Year. Month. Date. ( )

تبدیل آدرس در Segmentation

آدرس به دو بخش هستند که کدام seg ← این seg را برنامه کاربردی می‌گوید

داخل این seg کجاست.

قبل از جمع d را ببینیم اگر بیشتر از محدوده نباشد



تقریباً شبیه paging است ولی در این جا یک adder یک مقایسه‌کننده اضافه کنیم.

این روش تمام مشکلات را ندارد. Intel برای حل این ساختار را استفاده کرد paging این دو را با هم ترکیب

کرده یعنی به جای page کردن page در segment ها seg ها page می‌کند هر seg را به صورت page در حافظه

ذخیره می‌کند

TLB سه field دارد ← شماره seg

base

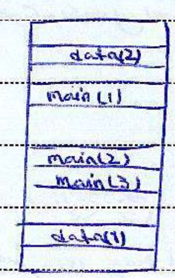
limit

صفحه‌بندی paged Segmentation (ترکیب مدیریت قطعه‌بندی، صفحه‌بندی)

← مدیریت حافظه قطعه‌بندی صفحه‌بندی شده

هر seg را به طور مستقل صفحه‌بندی می‌کنیم

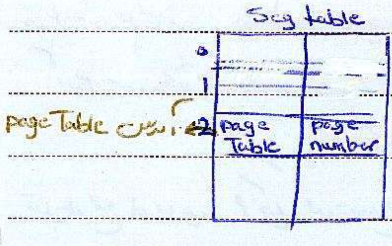
seg توسط برنامه‌نویس انجام می‌شود ولی paging بر عهده سیستم‌عامل مخفی است



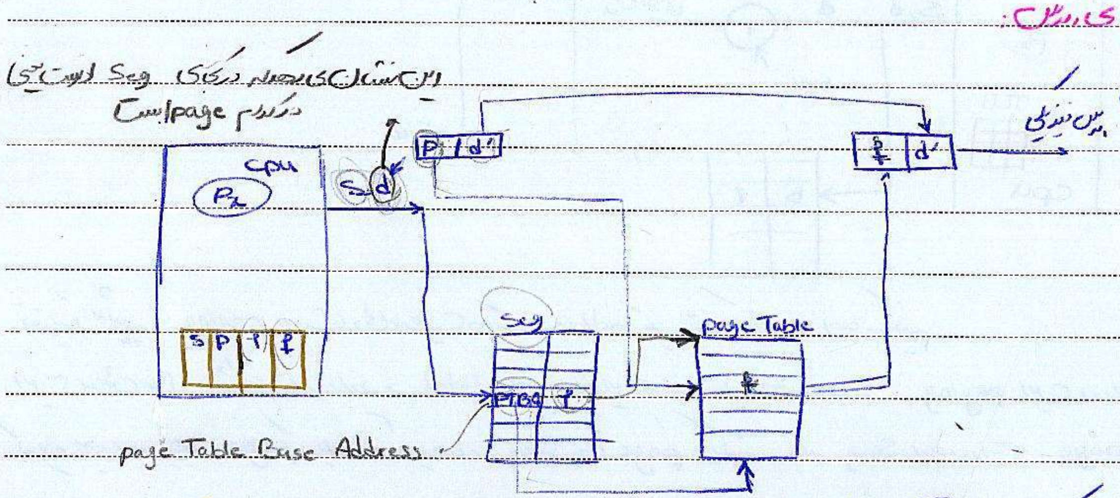
دیگه seg ها لازم نیست که به صورت پیوسته در حافظه قرار بگیرد

Subject:
Year. Month. Date. ( )

برای هر Seg در Seg Table باید نشان دهیم که page Table آن کجاست



ترکیب دو روش:



S ← مشخص می‌کند به کدام Page Table برویم

باید حواسمان باشد که limit را رد نکند

در این حالت برای یک فرایند چندین page Table داریم به تعداد Seg ها ولی اگر paging محض داشتیم برای کل فرایند یک page Table داشتیم

در این حالت مسئله دسترسی به حافظه بیشتر شد ← Seg table

page "

حافظه

برای حل از روش‌های قبلی استفاده می‌کنیم ← استفاده از TLB

Seg page d f

که کدام page مربوط به این Seg — برای دسترسی سریع ذخیره می‌کنیم

در TLB باید دنبال زوج (seg, page) برویم. ← قبلاً فقط دنبال page جستجو می‌کردیم

L در واقع تعداد صفحه‌های Seg است.

که خاصیتش متغیر بودن است و هر Seg می‌تواند تغییر کند

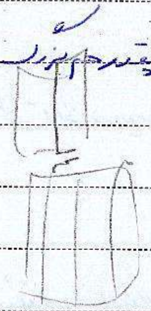
پردازنده Intel علاوه بر این این اجازه را می‌دهد که فقط از paging استفاده کنیم به این ترتیب که کل برنامه را یک قطعه تعریف کنیم ولی در داخل باز هم از قطعه‌بندی استفاده می‌کنیم.

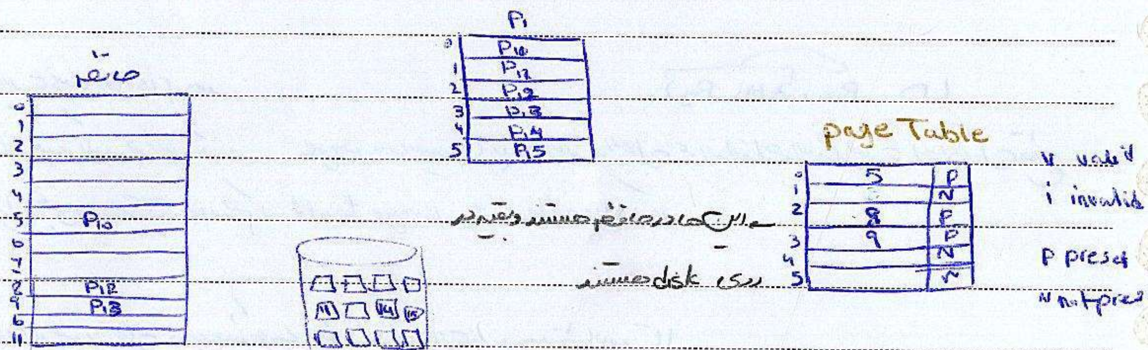
← آخرین جواب همیشه هر جا که می‌گیریم paging است و خیلی از مشکلات را حل می‌کند

حافظه مجازی: Virtual Memory

تا حالا فرض کردیم برای اجرای هر فرآیند باید تمام فرآیند داخل حافظه باشد (فرض محدودکننده‌ای است، هر چقدر هم بزرگ باشد ۲ فرآیند بزرگ می‌توان آنرا پر کنند در نتیجه حداکثر ۲ تا فرآیند با هم اجرا می‌شوند)

حافظه مجازی این فرض را نقض می‌کند و می‌گوید لازم نیست کل فرآیند در حافظه باشد





اگر با توجه به اصل locality تقسیم قسمت برنامه اجرا می‌شود، هر لحظه هر جنبه page به لازم می‌شود داخل حافظه می‌آوریم و یا وقتی به بقیه احتیاج نبود آن‌ها را وارد حافظه می‌کنیم یعنی در اصل بخش‌هایی از فرآیند در حافظه باشند که به آنها لازم داریم و با این منطق خیلی فرآیند می‌توانیم با هم اجرا کنیم

← page Table یک field جدید برای این به page Table اضافه می‌کنیم تا ببینیم محتوای کدام خانه page valid است یعنی در روی disk نیست چون حافظه می‌تواند خالی باشد.

به هر حال جدول صفحه ساخته می‌شود (توسط OS) و مشخص می‌کند که کدام قسمت‌های آن معتبر و کدام‌ها غیر معتبر هستند (یعنی در اصل disk را هم به حافظه اضافه کردیم)

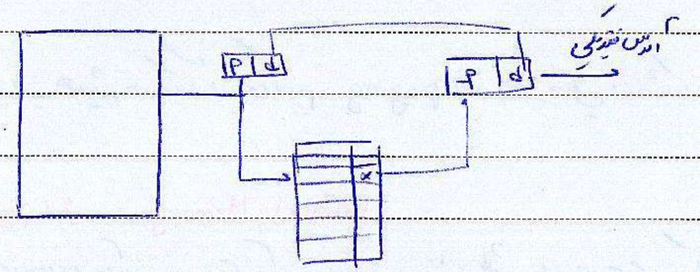
در واقع disk می‌شود Virtual Memory چون ما محدودیت نداریم دیگر بزرگ به نظر می‌رسد.

حافظه مجازی: حافظه بسیار بزرگی که واقعی نیست

از بخشی از disk به عنوان حافظه مجازی در نظر گرفتیم چون که در disk فقط اینها نیستند

حالا تبدیل آدرس چگونه انجام می‌شود؟ تبدیل آدرس توسط سخت افزار است

تبدیل آدرس تفاوتی با paging ندارد و فقط یک تفاوت کوچک دارد.



سخت افزار باید به بیت valid توجه کند، در صورت غیر valid بودن تولید وقفه کند

Page Fault (خطای صفحه) ←

LD Rx, 341(Ry)

مثلاً می‌خواهیم دستور را انجام دهد

آدرس را تولید می‌کند می‌بیند که در page حاضر نیست پس دستورالعمل نمی‌تواند اجرا شود پس اجرای آن قطع می‌شود و به سراغ بخشی از OS می‌رویم که page fault را handle می‌کند

1. اجرای دستورالعمل دچار مشکل می‌شود (page در حافظه نیست)
2. تولید وقفه page fault
3. اجرای برنامه قطع می‌شود و OS کار می‌کند ← قسمت page fault در OS کار می‌کند

از روی آخرین آدرسی که انجام شده می‌فهمد در کدام page است

توسط OS انجام می‌شود:
- page را پیدا می‌کند
- از disk به حافظه منتقل می‌شود
- باید page Table را اصلاح کند
- کنترل به برنامه کاربری باز می‌گردد

کنترل دوباره به برنامه کاربری برمی‌گردد و دوباره دستور انجام می‌شود

← پس تبدیل آدرس OS نیست بلکه سخت افزار است که در صورت عدم انجام تولید وقفه می‌کند

اگر حافظه جای خالی نداشته باشد تکلیف چیست؟ باید page از یک فرآیند را از یک frame بیرون کنیم و جایگزین کنیم

برای TLB، cache برای بیرون کردن روش‌های مختلفی مطرح است

این سوال مطرح است

روش عمده‌ای که استفاده می‌شود LRU است ولی در cache از روش های ساده‌تر مثل FIFO استفاده می‌شود

جلسه بیست و ششم ۸۷،۹،۶

می‌توان بطور نسبی واقعی نشان داد که حافظه بزرگ است.

حافظه فیزیکی 4GB = 2^{32} باشد ولی حافظه مجازی خیلی بزرگتر می‌تواند باشد در حدود 2^{45} = 32TB باشد

پس در نتیجه اگر یک OS خوبی داشته باشیم می‌توان تعداد زیادی Task را داشت

روش های جایگزینی: Replacement Policies

حافظه فیزیکی پر شود و ما برای دسترسی آن از disk استفاده می‌کنیم اینکه چه کسی را بیرون کنیم تا فرآیند جدید را جایگزین کنیم این روش هست.

ساده‌ترین راه: کاری داریم که چه قدر در حافظه بوده، به ترتیب بیرون می‌کنیم یعنی کسی که زودتر از همه آمده بیرون می‌رود و کاری نداریم که هنوز استفاده می‌شود یا نمی‌شود.

رشته مرجع Reference string ، برای توضیح مثال

ارجاع به صفحه چطور است. فرض کنیم سیستمی را monitore می‌کنیم تا آدرس های که تولید می‌کند را ببینیم

0112 0113 0114 0614 0615 0112 0113

(در 0112: P = 01 ، d = 12)

این آدرس های کاری

جاهایی که page عوض می‌شود مهم است که page fault داریم.

d ها را حذف می‌کنیم

01 01 01 06 06 01 01

جاهای تکراری که page fault نیست را دور می‌ریزیم

01 06 01 ← Reference string

page fault است ، خطی است