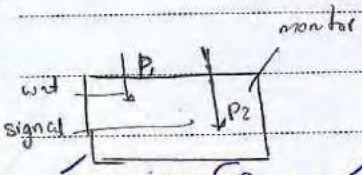
Subject:
Year. Month. Date. ( )

مثلا) فرض کنید یک monitor داریم که proc. P_1 وارد این می‌شود و وسط کار wait می‌کند از طرف دیگر یک P_2 وارد monitor می‌شود و پس از مدتی P_2 متغیری که P_1 روی آن wait شده signal می‌شود از این لحظه به بعد P_1 باید ادامه پیدا کند یا P_2



جواب صحیح مورد چهارم کدام است؟ باید P_2 از متغیر P_1 را سیگنال کرده است باید ادامه پیدا کند تا P_1 بعد از آن شروع کند

پس اینجا کدام یک از سه مورد بسته به مثال دارد اما بهترین راه حل اینست که signal کردن P_2 آخرین کاری باشد که P_2 می‌کند و پس از آن بیرون برود و کدی که است بتواند همچین کاری کرد

مثال 2 کدام راه حل را اجرا می‌کند؟

1- monitor که عملیات را کنترل می‌کند ولی نمی‌تواند این کار را بکند

2- نیاز به پشتیبانی compiler داریم تا بتوان این کار را انجام داد یعنی باید بفهمد چه زمانی کرد ولی نمی‌تواند منطقه پشتیبانی OS نیاز دارد ولی compiler آن را compile می‌کند سپس برای کنترل نمی‌تواند کاملا اضافی می‌شود نیاز به OS هم دارد

3- اینکه P_1 باید انجام شود یا P_2

چند بار برنامه‌نویسی را می‌شناسید که monitor را شناسده Java

کلمه monitor را ندارد ولی اگر یک synchronized را object تعریف کنیم این خاصیت را دارد در دیدیم چطور همچین compiler نبود خودمان باید این حساب را صدا می‌کردیم

باشگاه مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

Subject:
Year. Month. Date. ( )

جلسه نوزدهم 8, 28

OS های موجود بطور مستقیم از ساختار پشتیبانی نمی‌کنند بنابراین داشتن مشکل است.

monitor ← نیاز به پشتیبانی compiler

semaphore ← نیاز به پشتیبانی OS دارد

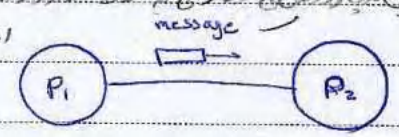
اما می‌توانیم برایش یک library routine تهیه کنیم که انجام شود.

استفاده از فراخوان‌های سیستمی send , recieve ← تبادل پیام یا message passing

این راه OS ها دارند و نیاز به پشتیبانی خاصی ندارد خیلی ساده است ولی کاربرد مشکل است

دو تا فرآیند داریم و می‌توان از هر فرآیندی به فرآیندی دیگر message ارسال کند

این کار در سطح پیشرفته‌تر OS ندارد و فقط ایجاد کننده این ارتباط است



message هر پیامی است و داخلش هر چیزی می‌تواند باشد مثلاً pointer یک ساختار

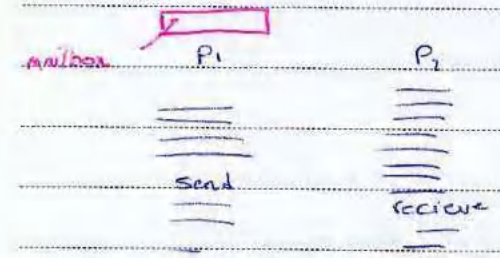
send (m, P2) → send می‌کند

message قضیه recieve (M) ← از سمتی header این کلمه پیغام برگشته

هر message دریافت کرد. ← هر کس حق دارد message را بفرستد

تا زمانی که این اجرا شود P2 message را می‌گیرد

در مقابل send ، فرآیند مقصد حتماً باید recieve را اجرا کند



وقتی send نشده چرا recieve می‌کنی

احتمال زیاد send باید زودتر از recieve اجرا شود

اما این شرط سختگیرانه است

راه حل: (در حالت منطقی send باید زودتر از recieve باشد)

برای هر فرآیند OS یک جعبه به نام mailbox برایش درست می‌کند هر پیامی که به فرآیندی می‌رسد داخل mailbox می‌رود

هر وقت بستر recieve را اجرا کرد یک پیام از صندوق mailbox برمی‌دارد

(چون message ها در mailbox می‌مانند پس گم نمی‌شوند)

← recieve از نوع سیستم call های blocking است یعنی اگر نتوانست کاری انجام دهد (هیچی نداشتیم) فرآیند را مسدود می‌کند و نگه می‌دارد در حالت مسدود تا زمانی چیزی داخل mailbox بیاید ← کاربر چیزی می‌خواهد

کجا برای نوع recieve مهم است چون جلوی دستی که دنبال از block خارج می‌شوی حتماً اون چیزی رو که می‌خواستی داری ←

دانشکده مهندسی کامپیوتر دانشگاه صنعتی امیرکبیر علاوری

Subject:
Year. Month. Date. ( )

حل مسئله producer consumer

```
void Producer ()
{
  int item ;
  message m;
  while (True) {
      produce (item);
      recieve (m);
      build_message (m, item) ;   ← item را کپی message قرار دهد و شاید چیزهایی
      Send (consumer, m) ;   → کاری نیم که Send شود   لیزال رهاش کند
  }
}


void Consumer ( )
{ int item, i;
  Message m;
** for (i=0; i<N, i++)  Send (producer, m)
  while (True) {
  → recieve (m) ;   Send که برای گرفت در این m recieve می‌شود
    extract_item (item);   از message، item را کپی می‌کند و به متغیر item ارسال کند
    Send (producer, m) ;  →
    consume (item) ;  →   مصرف کن
  }
}
```

انجمن علمی مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

Subject:
Year. Month. Date. ( )

فرض می‌کنیم mailbox نداریم یعنی ما send نکنیم می‌توانیم recieve کنیم.

← recieve(m) در producer یک مجوز است که consumer می‌گوید من جا دارم بفرست و وقتی جا نباشد producer در این جا معطل می‌شود

← برای اینکه این دو با هم جلو روند consumer دارای حرف دریافت مجوز صادر می‌کند و این طوری است که تولیدی شود مصرف می‌شود تولید می‌شود تا وقتی مصرف نشده جای تولید نمی‌شود

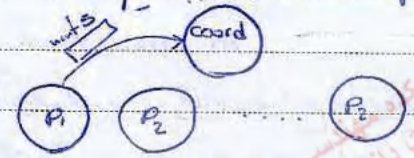
** در این خط قبل از حرکاری N تا مجوز می‌فرستد پس هیچ وقت روی recieve در producer گیر می‌کند چون که mailbox ها پر است و هیچ وقت block نمی‌شود اگر ۱ مجوز بفرستیم به این معنی است که این یکی تو است دیگر در این جا producer می‌تواند تولید کند و کاری نداشته باشد که consumer مصرف می‌کند یا نه

← ما می‌توانیم Semaphore را با استفاده از این بسازیم

اگر بتوانیم semaphore را شبیه‌سازی کنیم می‌توانیم تمام کارهایی که می‌کردیم الان هم انجام بدهیم

شبیه‌سازی Semaphore

تجاری می‌کنند هستیم که می‌خواهیم از یک Semaphore مشترک ایجاد کنیم



rec
open message
wait operation
if not negative send

یکی از این‌ها باید Semaphore را ایجاد کند ← ایجاد Semaphore: یک فرآیند کلی ایجاد می‌کنیم فرآیند کلی در حالت recieve قرار می‌گیرد (block)

P1 یک message (wait) به coord می‌فرستد (مستقیم باهم شاید چند تا صفر نداشته Signed کند) coord می‌فرستد بلند صدا چیست wait می‌کند P1 wait(S)

coordinate message را بازمی‌کند می‌بیند wait روی S است

messag | wait | S | ...

پس wait را اجرا می‌کند اگر پر نشد send می‌کند

send(coord,)
recieve => block شد

در حالی که از این کار نمی‌کند و اگر منفی شد چیزی جلوش نفرستاد دست (مثل wait است)

دانشگاه صنعتی امیرکبیر
اطلاعات دانشگاه صنعتی امیرکبیر

Subject:
Year. Month. Date. ( )

اگر ما خودمان را داشته باشیم، monitor را هم می‌توانیم پیاده‌سازی کنیم (اگر OS آن را نداشته باشد)

فصل 5 ← 2، 9، 11

جلسه بیستم

بن‌بست Deadlock

این بحث را زیاد کش می‌دهیم چون در سیستم‌های تک پردازنده که ما با آن‌ها کار داریم چندان مطرح نیست چون اکثراً هر فرآیندی که چیزی می‌خواهد از OS درخواست می‌کند البته خودش هم می‌تواند اقدام کند ولی اکثراً این جوری نیست ـ البته این بحث در جای خود بسیار است.

اگر 4 شرط زیر برقرار باشد بن‌بست پیش می‌آید

1. انحصار متقابل mutual exclusion ـ یعنی اگر منبع انحصاری نباشد بن‌بست پیش نمی‌آید ـ (منابعی از منبع غیرانحصاری رها شده است)

2. نگهداری و انتظار hold & wait ـ دو تا منبع داریم برای هر دو کاربر داریم اگر یکی را رها کنیم که نمی‌کنیم) ← (وقتی که یکی از کاربرهای مال یک منبع را از دست می‌دهیم آن را رها می‌کنیم تا فیلی را بدست بیاوریم) وقتی کار به پیش می‌رود که همه کاربرها را بدست بیاوریم و در نهایت بن‌بستی پیش می‌آید

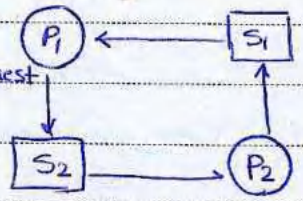
3. قبضه کردن نباشد no preemption
منظور پردازنده است که اگر یکی وجود دارد هیچ اتفاقی نمی‌افتد ولی هر منبع این جوری نیست (اگر یک منبع را که داریم تا خودش تمام شود کار کند است بن‌بست پیش آید ولی اگر بتوان آن را قبضه کرد دیگر بن‌بست پیش نمی‌آید)

4. انتظار چرخشی circular wait
شرط ساختاری نیست بلکه کار است و چون کار پیش می‌آید. هر آنچه را که بررسی کردیم این دیدگاه را داشت

دانشکده مهندسی کامپیوتر و فناوری
دانشگاه صنعتی امیرکبیر

Subject:
Year. Month. Date. ( )

allocation تخصیص منابع

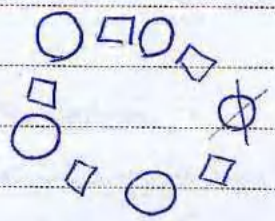


حالا P2 هم درخواست را دارد ولی در اختیار P1 است پس می رود در حالت انتظار

درخواست منابع خواهد ولی نمی تواند بگیرد چون در اختیار P2 است پس می رود در حالت انتظار

در مثال فلسفه ما انتظار چرخشی بود که در آن انتظار چرخشی بین 5 تا فیلسوف است و هیچ راهی هم ندارد مگر اینکه یکی را بلند کنی و بیرا هی های دیگر

ساده

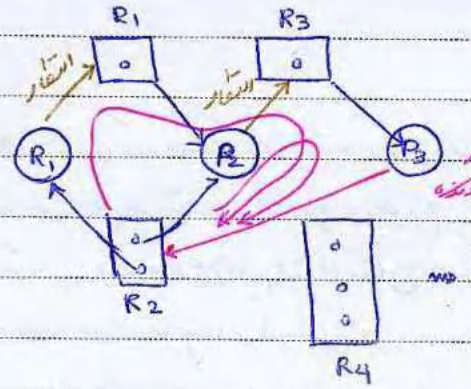


مثال:

R ← منبع resource

تخصیص منابع

request



R1 را می خواهد ولی در اختیار P2 است می رود در انتظار

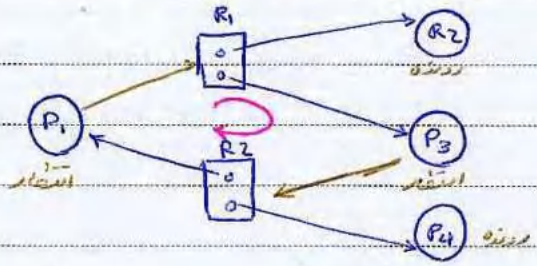
از منبع یک نوع بیشتر از یکی داریم (چند تا یک)

در این وضعیت سیستم در حالت بن بست نیست چون ما امید داریم (مطمئن نیستیم) که P3 تا آخر برود و منبع را آزاد کند و بعد P2 تا آخر رفته و P1 تا آخر خلاص می شود.

* حالا که هر 3 تا در انتظار رفتند و منتظر هم اند بن بست داریم. در تا حلقه به وجود آمده

دانشکده مهندسی کامپیوتر دانشگاه صنعتی امیرکبیر

Subject:
Year. Month. Date. ( )



مثال) حلقه دیده می‌شود

* لزومی ندارد همه فرآیندها از بن بست باشند

در حلقه - هر تعداد افرادی که در حلقه اند کافیست

در بن بست باشند

با وجود حلقه در بن بست نیستیم می‌توانیم امید وار باشیم که P_2 تا R_1 را آزاد کند پس P_1 می‌تواند R_1 را بگیرد تا

انجام شود

راه برای خلاصی است.

← وجود حلقه شرط لازم برای بن بست است ولی کافی نیست

(اگر بن بست باشد حتماً حلقه هست)

راه حل های بن بست

1. در OS های یک پردازنده ای به کاری بکند که جلوی کم اتفاق می‌افتد ← بعد راه حل. در این حالت چون راه حل

نداریم یعنی تشخیص بن بست هم نداریم پس اگر پیش آمد چی می‌شود؟ کاربر می‌بیند hang کرده پس یا process

را از بین می‌برد یا reset می‌کند. ولی با اصلاحی مجیم بن بست رخ داده فکر می‌کنیم یک مشکلی بوده و چون

هم نیست راه حل ارائه دادن نداریم!!!)

در تک پردازنده چون یک پیش داریم خود این بست پیش می‌آید (ولی اگر چند تا پردازنده بود ... OS)

2. الگوریتم بانکدار ← اجتناب از بن بست Dead lock Avoidance

مسیر ها را طوری تعیین می‌کند که به بحث بن بست نزدیم

3. تشخیص بن بست (متضمن مخدوری راه حل می‌باشد چون همین تشخیص قدم اول است که قطعاً ساده هم نیست)

4. جلوگیری از بن بست Dead lock Prevention ← اصلاح سیستم های یک پردازنده ای که برا ندارد

در سیستم های توزیع شده برد دارد چون به شدت اتلاف منابع دارد

طراحی اولیه سیستم به گونه ای که یکی از این 4 شرط اصلا وجود نداشته باشد ← نقض همیشگی یکی از این 4 شرط

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

Subject:
Year. Month. Date. ( )

- الگوریتم بانکدار که روی آن مانور زیادی داده شده در هیچ جا کاربرد ندارد ولی برای سوال طرح کردن خوب است! (می‌شود بچپانید)

منابع را یک جوری تخصیص می‌دهیم که همه تحت امن بسته نروند. از ترس اینکه بن بست نشود منبع را دارد توهم داری نه که می‌زنی ولی بعد نمی‌رسد!!

مثال ربات‌ها (در نقش فرآیندها) همان مسیری را که امید برویم!!

- تشخیص بن بست در سیستم‌های توزیع شده خیلی صرف دارد (با ساختن گراف تخصیص منابع) و بعد از تشخیص با برگشت به عقب از بن بست خلاص می‌شود (حالتی که هنوز بن بست رخ نداده ...)

(جزو روش تخصیص نیست)

جلوگیری از بن بست در سیستم‌های توزیع شده دیده شده ولی زیاد رایج نیست

مثال برای الگوریتم بانکدار

سه فرآیند P_1 و P_2 و P_3 یک منبع R_1 که 12 مورد از آن موجود است.

از قبل این اطلاعات اولیه به ما داده شده که حداکثر تقاضاها از منبع R_1 به صورت زیر است:

(هم نیست از کجا می‌دانستند)

P_1	10
P_2	4
P_3	18

15

حالت فعلی در زمان T

P_1	5	در این لحظه
P_2	2	
P_3	12	

این حالت بد است یا خوب؟ یعنی آیا احتمال دارد ما را به سمت بن بست ببرد یا نه؟

(safe) امن

93 مصرفی

3 موجودی

دانشکده مهندسی کامپیوتر
اطلاعات دانشگاه صنعتی امیرکبیر

Subject:
Year. Month. Date. ( )

باید ببینیم با این 3 تا می‌توانیم هیچ یک از این 3 فرآیند ها را به پایان برسانیم

- با انتخاب P_2 به آخر می‌رسد.

حالا که تمام شد موجودی 5

P_1 را با 5 تا می‌توان به آخر رساند انتخاب P_1 به آخر می‌رسد

موجودی 10

P_3 هم به آخر می‌رسد

این جوری ما یک مسیر پیدا کردیم، یعنی اگر به این ترتیب اجرا کنیم به آخر می‌رسند

ترتیب امن <Safe Sequence> $\langle P_2, P_1, P_3 \rangle$

پس بله این حالت امن است.

ادامه مسئله: P_3 تقاضای یک عدد اضافه می‌کند، آیا حالت جدید امن است؟

P_1	5
P_2	2
P_3	3

موجودی 2

انتخاب P_2 به آخر می‌رسد

موجودی 4 ← با این موجودی P_1 یا P_3 را نمی‌توان به آخر رساند ترتیب امن وجود ندارد

پس حالت جدید امن نیست

اگر بیتم بانکدار ان می‌گوید اگر P_3 تقاضای یکی اضافه کرد چون حالت جدید امن نیست یعنی احتمال بن بست هست، این یکی اضافه را به او نمی‌دهیم و می‌گذاریم در حالت انتظار بماند

(خلاف ترتیب P_2، P_1، P_3 تقاضا کرده)

برای همین منبع به درد نخوری است چون منبع را داشت ولی از روی ترس ندارد ①

از کجا می‌دانیم فرآیند چه اگر چند تا از یک منبع می‌خواهد حتی اگر 100 تا تکرار

مجبور نیستیم تازه دست بالا هم ⇐ شده باشد اگر دوری فرق کند، این هم فرق دارد ②

نکنیم

دانشکده مهندسی کامپیوتر علاوری
اطلاعات دانشگاه صنعتی امیرکبیر

PAPCO

Subject:
Year. Month. Date. ( )

هر بار بیست یکم

الگوریتم بانکدار

n فرآیند

n منبع

available [m] تعداد منابع j ام k عدد دارد — Available [j] = k

ماتریس max [n,m] ← هر سطرش max یک فرآیند

ماتریس allocation [n,m] ← سطر i ام مربوط به فرآیند i ام ← چه مقدار از منابع استفاده کرد

← ماتریس تخصیص

ماتریس claim ← هر فرآیند چه مقدار نیاز دارد تا به max برسد — claim = max - allocation

این سه تا ماده هم در اندازه و هم در مقدار در تغییر است. فرآیند ممکن است ایجاد شوند یا از بین بروند

می‌گوید منبع می‌خواهیم چه مقدار می‌خواهیم چه شکلی خواهیم

فرض می‌کنیم request i رسیده می‌خواهیم چکار کنیم؟

← یک بردار است — از max بیشتر نباشد

1- این چیزی است که چک می‌کنیم اینست که غیرطبیعی نباشد — Request i ≤ claim i

اگر برقرار نباشد به چه درد غیر این حالت خطا است

2- اگر Request i ≤ Available — اما به دلیل خطا — منابع را هم بخش بدهیم یا نه (منابع موجود نیاز دارد)

3- به حالت جدید شبیه کنیم ببینیم امن هست یا نه

حالت جدید ← Available = Available - Request i

Allocation i = Allocation i + Request i

claim i = claim i - Request i

اگر حالت جدید امن باشد تخصیص انجام می‌شود وگرنه انتظار

چه جوری بفهمیم امن هست؟

دانشگاه صنعتی امیرکبیر
دانشکده مهندسی کامپیوتر و فناوری اطلاعات

Subject:
Year. Month. Date. ( )

آزمایش حالت امن:

1. work = Available
   Finish[i] = False برای i=1,...,n

2. ردیفی که به حد جدید عمل می‌توانیم آن را به انتها برسانیم پیدا کنیم
   Finish[i] = False
   claim_i < work

اگر پیدا نشد به مرحله 4 برو

3. work = work + allocation; جمع منابع فرآیندی که به انتها رسیده، زیاد می‌شود
   پس حد جدید زیاد می‌شود
   اگر به این مرحله رسیدیم یعنی فرآیند
   بین مراحل 2 و 3 انقدر گشتیم تا یکی را پیدا نکنیم

4. اگر برای i=1,...,n finish[i] = True باشد ← امن است

اگر حالت امن نباشد به معنای این نیست که در حالت انتظار است جمع کل است باید بررسی
یعنی در حالت کل منابع را دارد ولی اجازه استفاده نمی‌دهد ← کاری که به این سمت نرسد

مثال) P_1 ... P_5 فرایندها

	A	B	C
منابع	10	5	7

← کل منابع ماست

مقدار max هم جزء داده‌های مسئله است

	A	B	C
P_1	7	3	2
P_2	3	2	2
P_3	9	0	2
P_4	2	2	2
P_5	4	3	3

ملحقه + سیستم را نگه می‌داریم و allocation را به این صورت بدیم

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

Subject:
Year. Month. Date. ( )

	A	B	C
P_1	0	1	0
P_2	2	0	0
P_3	3	0	2
P_4	2	1	1
P_5	0	0	2
	7	2	5

← T مرحله allocation

چیزی که مصرف شده

=> Available = 3 3 2

2 1 0

	A	B	C
P_1	7	2	5
P_2	1	2	2
P_3	6	0	0
P_4	0	1	1 ✓
P_5	4	3	1

max-allocation ← need (claim) ترسیم

available 3 2 1

آیا این حالت امن است؟ احتمال اینکه بن بست برسد هست یا نه

need را نگاه می‌کنیم ← P_2 و P_4 می‌توانند به این مقدار به آخر برسند

یک ترتیب امن باید پیدا کنیم لزومی ندارد یکتا (unique) باشد در این جا مثلاً P_2 را انتخاب می‌کنیم و انجام می‌شود

و در نتیجه P_2 موجودی خود را آزاد خواهد کرد پس داریم

Available = 5 3 2 ✓ 5 2 1

P_4 انتخاب	7 4 3	1 4 3
P_3 انتخاب	10 4 5	10 4 5
P_1		
P_5		

$\langle P_2, P_4, P_3, P_1, P_5 \rangle$ یک ترتیب امن پیدا شد پس حالت امن است

دانشکده مهندسی کامپیوتر و فناوری
اطلاعات دانشگاه صنعتی امیرکبیر

Subject:
Year. Month. Date. ( )

Request$_1$ = 0 2 0 2 می‌شود آیا حالت جدید امن هست یا نه؟

allocation — need (Claim)

	A	B	C	A	B	C
P_1	0	3	0	7	0	5

available => 3 1 2

باید ببینیم این حالت امن است یا نه.

در این جا P_2 را می‌توانیم انتخاب کنیم پس باید P_4 را انتخاب کنیم

available	5 2 3	P_4	انتخاب
"	7 2 3	P_2	"
"	10 2 5	P_3	"
	⋮		

این حالت هم امن هست. اگر پیدا نمی‌کردیم امن نبود.

اگر بخواهیم ترتیبی انتخاب کنیم باید در Safe Sequence باشد. در غیر این صورت به حالت امن نیست را اشتباه می‌دهیم.

تشخیص بن بست Deadlock Detection

الگوریتم قبلی باید برای هر request باید اجرا شود. ولی یک request باید اجرا شود که این کل وقت پردازنده را می‌گیرد.
این الگوریتم هزینه‌ی قابل اجراست و هر وقت اجرا کنی می‌گوید بن بست داری یا نه. مثلاً هر 50s که بینیم کسی در بن بست نیست. از اطلاعات موجود استفاده می‌کند.
خیلی شبیه الگوریتم قبلی است.

1. work = available
   اگر allocation ≠ 0 ⇐ Finish[i] = false
   وگرنه Finish[i] = True

از n متغیر باینری داریم، یعنی که allocation صفر دارند به حساب نمی‌آوریم
از زمانی که شرط اعضا و حفاظ دارند استفاده می‌کند
یعنی هیچی لازم ندارد

انجمن مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

Subject:
Year. Month. Date. ( )

سایتهایی که برای شرط اعضاء متقابل دارند در این بسته وضعیت دارند ← در اصل ما تجارت را محاسبه می‌کنیم که شرط اعضاء متقابل نداشته باشند

2. i را به گره‌ای پیدا کن که Finish[i] = False ← یک چیزی پیدا کن که سایتی که می‌خواهد از وجودی کمتر باشد

request_i ≤ work

اگر نه به مرحله 4

3. work = work + allocation_i ← فرآیند را انجام می‌کنیم و به جلو می‌رویم

Finish[i] = True

به مرحله 2 برو

4. به همه Finish ها True هست ← این بسته نیست

اگر ... ... شدند ← این بسته هست

اگر Finish[i] = True برای i = 1 ... n ← این بسته نداریم

این الگوریتم می‌گوید که کی‌ها در این بسته هست

← کمکش است request را به این بسته بیرون می‌آن موقع می‌فهمیم

مثال: فرآیندها P_1 ... P_5

منابع ← A B C

7 2 6

می‌توان request داشته باشیم که به آن منابع نداریم

	allocation			request		
	A	B	C	A	B	C
P_1	0	1	0	0	0	0
P_2	2	0	0	2	0	2
P_3	3	0	3	0	0	0
P_4	2	1	1	1	0	0
P_5	0	0	2	0	0	2
	7	2	6			

available 0 0 0 ← هر چی داشتیم تحویل داریم

← فرآیند request ندارد

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

Subject:
Year. Month. Date. ( )

در این سبد هستیم یا نیستیم؟

انتخاب P_1 یا P_3 چون هر دو صفر هستند

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

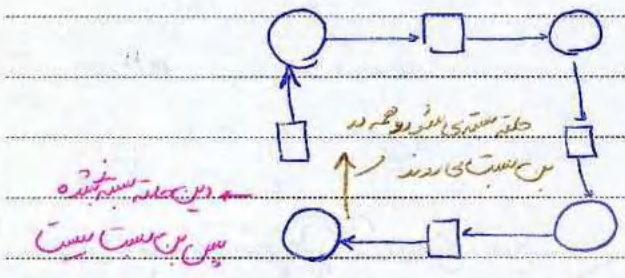
پس System در این سبد نیست، چون همگی True شدند

Request $_3$ = 0 0 1 2 می‌رسد ← از آیدی سوم تا صدای ! جدول را پر کن

ماتریس request در P_3 0 0 1

انتخاب P_1 0 1 0 ← با این چیزها کار را می‌توانیم به آخر برسانیم پس سبد می‌رسیم

← P_2 و P_3 و P_4 و P_5 ← در این سبد هستند



می‌دانیم در این سبد وجود دارد که یکی در این سبد است
است می‌توانیم در این اطلاعات برای شکستن حلقه
استفاده کنیم

← سیستم خلوت ← ready Queue ← خلوت‌تر است

" شلوغ ← " ← دارای request زیادتر است

الگوریتم این سبد را کدام باید بیشتر اجرا کنیم؟

در سیستم خلوت، در جایی که شلوغ است اصلاً وقت ندارد سرش را بخاراند

← شاید دلیل خلوت شدن این سبد به شدت منطقی است که این جا بیشتر اجرا شود

دانشکده مهندسی کامپیوتر و فناوری
اطلاعیه دانشگاه صنعتی امیرکبیر

Subject:
Year. Month. Date. ( )

جلسه بیست و یکم 87/9/12

تمرین OS برای هفته آینده

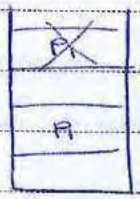
مدیریت حافظه: memory management

← همه‌ی فرآیندها باید بیایند ← آزمون مرحله دوم ← پنج‌شنبه 9/28 ← ساعت 8

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

← وظایف مدیریت حافظه

جابجایی relocation



هر فرآیند را می‌توانیم در هر جای حافظه به آن بدهیم

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

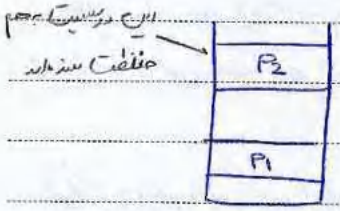
این چهار حافظه قابل جابه‌جایی هستند چون در هر جای حافظه‌ای توانیم قرار دهیم

وقتی همه چیز را می‌شود جابه‌جا کرد چون ممکن است در کد آدرس دقیق به کار برده باشیم و در برنامه مشکل می‌خوریم بجای از

مقادیر نسبی استفاده کنیم (نسبت به Base address)

تا بعد جای اول ← از مقادیر مطلق آدرس استفاده نشود

حفاظت protection



فرآیندها نباید به فضاهای یکدیگر دسترسی داشته باشند

این محل‌ها محافظت کنند

اگر این‌ها بخواهند از یک مشترک استفاده کنند ← مشکل هست

می‌توانند از بعضی چیزها که همه چیز را به طور مشترک استفاده کنند محافظت شده

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

← حافظه دارای دو سازمان ← logical organization

physical "

دانشگاه صنعتی امیرکبیر

Subject:
Year. Month. Date. ( )

باید این دو تفاوت قائل شویم

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

ولی رسانه منطقی به نظر می‌رسد که پشت سر هم است

physic ← چیزی که واقعاً هست

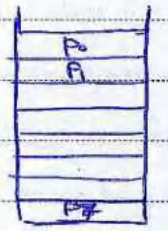
logical ← چیزی که به نظر می‌آید که هست

روش‌های مدیریت حافظه

۱. روش ایستا (static)

partitioning

در این روش حافظه از قبل تقسیم‌بندی شده ← بخش‌بندی ثابت



→ هر بخش حجم یکسانی هستند و هر کدام می‌توانیم ۴ فرایند داشته باشیم که فضاهایش هم محدود است

این روش خیلی محدودکننده ولی خیلی ساده است

در سیستم‌هایی که کاربر محدودی دارند می‌توان از این استفاده کرد چون ممکن برنامه‌ها زیاد باشند یا اینکه یک برنامه خیلی بزرگ باشد (در سیستم‌های general purpos)

این روش کاربردهای خاص دارد که تعدادشان هم زیاد هست ← روش خیلی ساده مدیریتی دارد ← ۸ تا حافظه دارد

← این فضا مدل کی هست

← embeded (سیستم‌هایی که کارکردی دارند)

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

بخش‌بندی ثابت

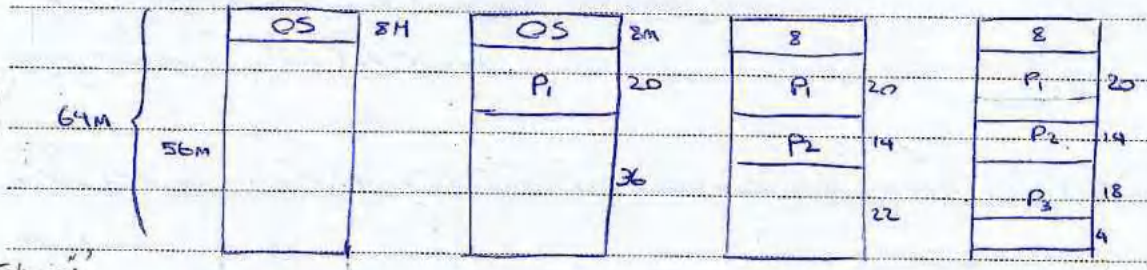
اندازه‌های متفاوت

دانشکده مهندسی کامپیوتر و فناوری
دانشگاه صنعتی امیرکبیر

Subject:
Year. Month. Date. ( )

2. روش پویا Dynamic

بخش بندی های حافظه متغیرند ← فضای حافظه یک فضای یکپارچه است که به هر صورت می‌توان آن را بخش بندی کرد.



مثلاً فرض کنیم خلق می‌کنیم

افزایش زمان →

در این جا فرایند P_4 می‌آید و فضای 8M می‌خواهد ← نداریم پس P_4 منتظر می‌شود ← پس از مدتی فرایند P_2 را خلق می‌کنیم. فضای P_2 خالی می‌شود، حالا می‌خواهد P_2 دوباره وارد شود ولی می‌بینیم که P_1 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 می‌توانیم به آن برسیم

این پدیده که هر چی جلو برویم باعث می‌شود حافظه بیشتر تکه تکه شود fragmentation می‌آیند

fragmentation →

فضای حافظه داریم ولی به درد ما نمی‌خورد

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

compaction

این کار در سیستم انجام می‌شود ولی این جا ارزش ندارد ولی این جا در سیستم ارزش دارد

← عدم پشتیبانی و مشکلات زیادی که اگر بیشتر جابه‌جا شوند زمان زیادی می‌گیرد

فضای خالی یکپارچه می‌شود

internal fragmentation

هزینه زیاد است ← مقدار زیادی از اطلاعات که جای به جای یکدیگر منتقل کنیم

زمان زیاد (زمان اشغال)

باشگاه مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی اصفهان

Subject:
Year. Month. Date. ( )

← Fragmentation ← در سرعت دسترسی تاثیر دارد چون فضاهای پشت سر هم تر می‌رسد هر چی فاصله فضاها در دسترسی زیاد باشد سرعت کندتر می‌شود ولی در حافظه تاثیری ندارد چون عضوی الکترونیکی است

↳ پس در حافظه کسی از این روش استفاده نمی‌کند

* در این بحث که کردیم سازمان منطقی و فیزیکی بر هم منطبق است

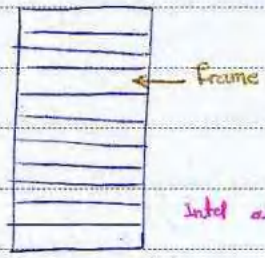
تا این حالت گفته شد مشکلات روش پویا بود.

روشی که در سیستم‌های امروزی برای مدیریت حافظه وجود دارد و مشکل روش پویا را حل می‌کند مدیریت حافظه paging است.

← بهترین روش مدیریت حافظه

مدیریت حافظه صفحه‌بندی Paging

↳ از نوع روش‌های پویا است.



حافظه از نظر OS:

تعداد جای‌های زیاد ولی خیلی ریز و کوچک (حافظه به این صورت تقسیم‌بندی شده)

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

مربوط به پردازنده Intel

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

اندازه frame ها از پیش تعیین شده است.

↳ نسبت به کل حافظه خیلی ریز است ← طوری که در کل هیچ فرآیندی جا نمی‌شود

فرض کنیم فرآیندی داریم که فضای بی‌خواصه اگر آن را در frame تقسیم‌بندی کنیم ← مثلاً 4 تا frame فضای خواصه

پیوسته (جای پیوسته)

پس فرآیند ما 4 تا page است ← می‌خواهیم این را بیاریم در حافظه ← لزومی ندارد حتماً frame پیوسته پیدا کنیم

بیزنیلی

یک صفحه frame که جای بدی یک page می‌تواند قرار گیرد

هر page هر frame می‌تواند قرار گیرد

پس دیگر نیازی به fragmentation نداریم

PAPCO

Subject:
Year. Month. Date. ( )

در این جا مسئله پیوستگی فضا از بین رفته پس دچار مشکل روش پویا می‌شود هر جا frame های بیکار را برای آزاد استفاده کنی

← پیوستگی را از بین بردیم پس در حین اجرا چی می‌شود از Page به هم Jump می‌کند؟

سوال های پیش می‌آید

← اختصاص دادن frame

← آزاد کردن frame

← مدیریت بر روی حافظه؟ اگر کل حافظه اشغال شد یک فرآیند آمد چی کار باید بکنیم

می‌توانیم بخش های از فرآیندی که در اجرای شده که الان به درد نمی‌خورد از حافظه خارج کنیم

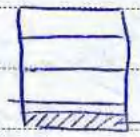
← ایرادش است کل فرآیند در حافظه باشد ← الان فرض می‌کنیم که کل فرآیند در حافظه است اما بعداً این فرض را بهم می‌زنیم

Fragmentation

خود Paging به تنهایی Fragmentation ایجاد می‌کند

هر فرآیند برای اتصال به Page تقسیم کردن می‌لزوماً Page آخر همش استفاده شده نیست، دارای بخش خالی باشد یعنی Page آخر خالی داشته باشد

این فضاهای خالی به فرم Fragmentation است که قابل استفاده نیست به هیچ وجه برای کاربر مطرح مقدارش خیلی کم است پس می‌گوییم اشکال ندارد



مراحل فرآیند اتمام یافته

internal fragmentation

← OS از کجا بدونه هر frame مال کی هست ← یکجایی باید این را ثبت کند ← OS برای هر فرآیند یک جدول صفحه درست می‌کند و در این می‌نویسد که page مال کجاست؟

جدول صفحه P_2

Page	frame
0	3
1	6
2	8
3	9

page 0 من کجاست

	P_1
0	6
1	4
2	5

پس مثلاً page 0 کجاست

Subject:
Year. Month. Date. ( )

حالا سیستم عامل از کجا می‌فهمد این جدول کجاست ← pointer جدول را در PCB نگه می‌دارد
آدرس شروع
برای حافظه آدرس شروع مهم است و کافی است کل جدول در حافظه نباشد

OS همیشه می‌داند که کدام frame ها در حافظه خالی هست.

جلسه بیست و هفتم 87.9.17

امتحان یکشنبه از ۱ آذر 87 ساعت 12:30 تا 13:30

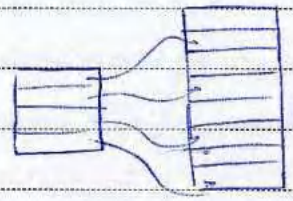
مدیریت حافظه صفحه‌بندی Paging

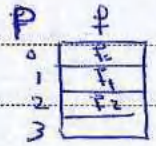
حافظه به frame ها تقسیم شده

فرایندها را به page ها تقسیم می‌کنیم

که اندازه page = frame و در حافظه قرار می‌دهیم

برای اینکه بدانیم page ها کجاست از یک جدول استفاده می‌کنیم





جدول صفحه

ترتیب frame ها مهم نیست ← مشکل فضای پیوسته وجود ندارد

اگر frame ها تمام شدند با یک سری راه‌حل کنیم

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

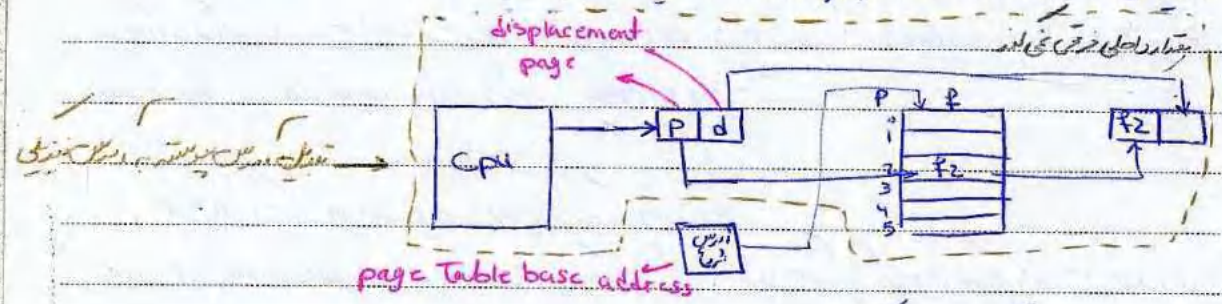
پردازنده این ها را به پیوسته تبدیل می‌کند ← کاملاً اختیاری

دید منطقی به منطقی تبدیل شود
یعنی آدرس منطقی به آدرس فیزیکی

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

Subject:
Year. Month. Date. ( )

این تکنیک عمل سخت افزاری است و اگر نرم افزاری باشد خیلی سرعت برنامه کم می‌شود.



← CPU در اجرای برنامه‌ها آدرس تولید می‌کند به حافظه می‌دهد

Instruction ← مربوط به

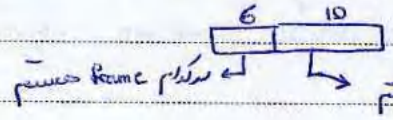
Data ← مربوط به

فرض می‌کنیم آدرسی که از پردازنده می‌آید منطقی است ← پیوسته

در عمل CPU آدرس خطی تولید می‌کند

برای حافظه 64k ← ما باید آدرس داریم.

16 frame => 1k ... frame تا 64 داریم



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

به آدرس ← به طور کلی نگاه می‌کنیم یعنی ما باید ببینیم

← دو قسمتی نگاه می‌کنیم یک ما باید یک 10 بیت ← برای حدس نگاه می‌کنیم

page جای کدام ← کدام page

باید page جای حافظه است ← از جدول صفحه مربوط به فرآیند استفاده می‌کنیم

حیطه پیوسته است ولی در حافظه پیوسته نیستند.

آدرس جایی که می‌بینیم پیوسته است ولی چیزی که در حافظه داده می‌شود

ناپیوسته است.

OS کمک افزار را به بدنه‌ی کاربر خودش گذاشته است انجام همین کار را کرده و دیگر اجرا حضور ندارد ولی کار خود را انجام داده و تا پایان خود را

← این کار کاملاً مخفی است و هیچ کس نمی‌بیند ولی OS می‌داند جدول صفحه خودش جدول صفحه را درست کرده ولی دیگر

اجرای فرآیند OS وجود ندارد.

دانشکده مهندسی کامپیوتر و مشاوره
اطلاعات دانشگاه صنعتی امیرکبیر

Subject:
Year. Month. Date. ( )

(point) هر فرآیند جدول مخصوص خود را دارد و فرآیندها به محدوده هم دسترسی ندارند
موقعی که فرآیندها عوض می‌شوند OS با توجه به PCB آن فرآیند Page Table base address را
تغییر می‌دهد OS قبل از اجرای فرآیند PTBA را ست می‌کند

← سخت‌افزاری که این تبدیل را انجام می‌دهد کی است؟
سخت‌افزار داخل CPU است و چیزی که از آن بیرون می‌آید آدرس فیزیکی است. ولی سخت‌افزارها
داخل CPU هست ولی PTBA در CPU نیست چون در حافظه است.

اندازه هر Page را طراح CPU تعیین کرده و اختیاری نیست و ما اختیاری نداریم
مشکلات حل شده:
- میدیم Fragmentation نداریم (داریم ولی بسیار مختصر و در آخرین Page هر فرآیند)
- مشکل پیدا کردن فضای حافظه را نداریم
مزایا: قابلیت انتخاب هر Frame ایجاد شده
جایگیری Frame های آزاد آسان شده ← به وسیله صفر و یک کردن خانه‌ها مثلاً اگر حافظه 64K باشد یک
رشته 64 تایی می‌تواند این کار را انجام دهد.

اندازه Page از کی 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: مثلاً مشکل مدیریت حافظه داشتیم 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 مشکل خیلی سرعت را پایین می‌آورد ← مشکل حافظه خط‌ها CPU است و سرعت را کم می‌کند
این سیستم برای سیستم پایین دارد
performance

PAPCO

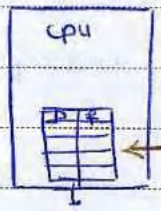
Subject:
Year. Month. Date. ( )

سیستم ها بدلیل حافظه خیلی مهم نیست، سرعت است که تعیین کننده است
بدین 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 بجای cache استفاده می کنیم
← بخشی که خیلی از حافظه را دارد درصد $\frac{1}{50}$ یا $\frac{1}{100}$ ولی همیشه اول cache را می گذاریم

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



اگر به جای اینکه سراغ حافظه برویم سراغ cache یا جدول TLB برویم سرعت ما خوب می شود سرعت ما درست مثل reg
مشکلش اینست که اندازه اش کوچک است

TLB: Translation Lookaside buffer (بافر دم دستی ترجمه)

← برای cpu دم دستی است.

در پردازنده Intel ← 32 تا مدل دارد ولی یک کنید براحتی می تواند 1000 page table داشته باشد
در TBL ← 98% آدرس ها جواب می دهد و تنها در 2% مجبور است سراغ page table برود
← اکثر دستورها با سرعت بالایی انجام می شود

TLB به OS هیچ ربطی ندارد و OS اصلاً نمی فهمد که TBL اصلاً وجود دارد و با جدول صفحه کار دارد
همگی 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 update کردن TLB ← یعنی اینجا page fault پیدا کردی یعنی در TLB

مشکل مهندسی کامپیوتر و فناوری
دانشگاه صنعتی امیرکبیر

Subject:
Year. Month. Date. ( )

← TLB از دید OS مخفی است ← سخت افزاری هستند

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

یک نکته: فرض کنیم تا الان P_2 کار می کرده الان P_1 به کد بیفتد داخل TLB چی هست؟ (هم زمانیش شروع شود)

هیچی چون هیچی داخلش بود مربوط به P_2 است دفعه پاک می شود و اولین بار به هر آدرسی که ارجاع می کنیم در TLB نیست ری هیت نیست چون این آدرس داخل PTB قرار می گیرد

← با شروع هر فرآیند (شروع همزمانی) TLB خالی است ← TLB به تدریج پر می شود.

↳ مجبوریم کار خود را از طریق جدول صفحه انجام دهیم

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

خود OS هم برای خود page table دارد ولی مجزا اولین چیزی که مرحله قرار گیرد پیوسته است.

d) همزمانی 1ms

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

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

↳ تعداد مراجعات از این بیشتر است چون ممکن است بیش از یک بار دستوری داشته باشد

TLB به کامپیوترهای بدون قبضه چقدر است؟

این نوع ذخیره سازی به جای آدرس به data چه هست

content Addressable Memory → cache مثل

2	
4	
5	
1	
8	

اطلاعات در TLB را چطور پیدا کنیم؟

به ترتیب نیست چون تنها page هایی که به آن دسترسی پیدا کردیم در TLB است

هیچ راهی جز این نیست که در تمام TLB را چک کنیم موازی

هر دسترسی باید کل TLB را چک کنیم ← 32 تا مقایسه کننده داشته باشیم (سخت افزاری)

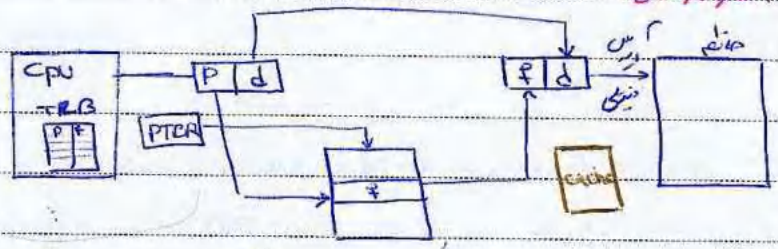
باهم موازی کار کنند ↳ به تعداد Lookup

↳ یکی از دلایل کم بودن خانه ها همین است.

اطلاعات دانشکده مهندسی کامپیوتر دانشگاه صنعتی امیرکبیر

Subject:
Year. Month. Date. ( )

جلسه بیست و ششم 87.9.24



یکی از reg های اصلی پردازنده که از PCB، load می‌شود که عنصر اصلی صفحه‌بندی است. چون شکل بهره‌وری پایین داشت کنیم از TLB استفاده کردیم.

مثال) حافظه: زمان دسترسی 50ns

TLB " " 2ns

hit ratio (نسبت اصابت): 98%

میانگین زمان دسترسی را حساب کنید (به حافظه) Teff

جدول همیشه دسترسی به حافظه نیاز دارد ← بعضی وقت‌ها از TLB بعضی وقت‌ها از طریق جدول صفحه

$T_{eff} = 0.98(2ns + 50ns) + 0.02(2ns + \underbrace{50ns}_{\text{جدول صفحه}} + \underbrace{50ns}_{\text{خود حافظه}})$

اول برای TLB جست‌وجو می‌شود

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

اگر بخواهیم یک دسترسی به حافظه داشتیم 50ns وقت می‌خواستیم ولی الان فقط 3ns اضافه شده که کمتر از 10% است و این خوب است تا 10% هم خوب است چون 10% را خطا در نظر می‌گیرد.

← معمولاً سیستم‌های واقعی این طور نیست. حافظه خیلی کند است و باید راهی پیدا کنیم این را تندتر کنیم (50ns در مثال 2ns خیلی زیاد است. سرعت حافظه بسیار کند است)

برای حل این مشکل از حافظه کمکی یا cache استفاده می‌کنیم. cache سرعت بالایی دارد به قیمت خرج کنیم که اصلاً به حافظه نیاز نباشد نیست. بخشی از حافظه را در cache قرار می‌دهیم (به صورت بخش حافظه نداریم که خاصه کد، data، Table ... باشد) مبنای کار ما حافظه اصلی است اما می‌خواهیم بیشتر وقت‌ها آن چیزی که می‌خواهیم در cache بیاوریم.

دانشکده مهندسی کامپیوتر و فناوری اطلاعات علم و صنعت ایران

Subject:
Year. Month. Date. ( )

← ساختار cache ، بحث افزاری است و OS از وجودش بی اطلاع است و OS با حافظه کار می کند و تمام مدیریت ها روی حافظه انجام می شود

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

نسبت اصابت هم داریم که کمتر از TLB هست : 90% ـ 90% مواقع در cache مشکل حل می شود.

برای کمک حافظه دیگر به مدیریت حافظه ندارد ← مراحل بهتر از این است.

cache با حافظه درهم گره می شود و در اصل می خواهیم بگوییم 50ns به چی تبدیل می شود.

مثال قبل درست شد 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 نگرفتیم

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

تا شد cache در جایی است که به حافظه دسترسی داریم عدد واقعی از 17.3ns بهتر است چون بعضی چیزها را در نظر نگرفتیم:

ممکن است در cache چیزی را پیدا نکنیم می رویم سراغ حافظه این طور نیست بلکه اگر در cache پیدا نکردیم از حافظه به cache می آید 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 با حافظه ارتباط دارد.

cache سطوح دو ← L1 cache داخل CPU ← سرعت خیلی بالاتر و ظرفیت خیلی کوچک

L2 cache روی mother board

این زمان دسترسی به حافظه هنوز هم خوب نیست یعنی تبدیل نسبت به CPU سرعت پایین تری دارد و هنوز هم گلوگاه سیستم است

مدیریت حافظه به روش paging بحث مشکلات Fragmentation و مدیریت را حل کنیم

← همه فضاهای خالی را OS باید حساب داشته باشد

S_1 b_1, l_1 — S_2 b_2, l_2

base — طول فضای خالی — آدرس شروع

S_3 b_3, l_3

P_4 وارد می شود و فضای خواهد و بیشتر از یک choice داریم

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

Subject:
Year. Month. Date. ( )

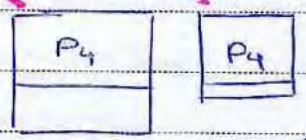
می‌توانیم جدول تا از این فضاهای خالی را به آن بدهیم، فضاهای خالی را باید search کنیم و ببینیم کدام فضای خالی از همه به P_4 نزدیک‌تر است که برای این نشان بدهد کل فضاهای خالی را search کنیم که این search خیلی طول می‌کشد ← روش best fit (بهترین برازش) search طولانی‌ترین انتخاب

روش دیگر اینست که به جای search کردن به اولین جایی که رسیدیم که P_4 می‌تواند در آن قرار گیرد آن را انتخاب می‌کنیم بدون این که search را کامل کنیم first fit (اولین برازش)

← صبحی کوتاه بدون بهترین انتخاب

روش دیگری هم هست ← گشاد‌ترین را پیدا می‌کند search می‌خواهد جدول بزرگ‌ترین را پیدا می‌کنیم و یعنی اینکه در list sort کردیم ← روش worst fit بدون جستجو

worst fit best fit



این منطق worst fit اینست: اگر بزرگ‌ترین قطعه 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 تئوری‌ای تایخ کننده است.

← مجموعه‌ای از برنامه‌های واقعی که برای استفاده می‌شود.

بدون این برنامه‌ها چیزی که می‌خواهیم آپ کنیم اجرا کنیم و اندازه‌گیری می‌کنیم و ببینیم کدام یک از حافظه بهتری می‌تواند استفاده کند ← مقدار جستجو

مقدار حدود نیمه حافظه ← بیش از $\frac{1}{3}$ حافظه هدر می‌رود.

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

$\frac{1}{3}$ حافظه تلف می‌شود

روش که بیشتر از همه استفاده می‌شود روش first fit است.

← paging تنها مسئله fragmentation را حل می‌کند بلکه مدیریت پیچیده فضاهای پیوسته را حل می‌کند و یک مدیریت ساده می‌آورد. یعنی ما دیگر فضای پیوسته نمی‌خواهیم

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

مدیریتی که الان گفته می‌شود که در برازنده Intel از آن استفاده شده مدیریت حافظه به برازنده حسابیده می‌شود چون نام تبدیلات را CPU باید انجام دهد که باید پشتیبانی CPU را مدیریت حافظه داشته باشد ← segmentation

PAPCO

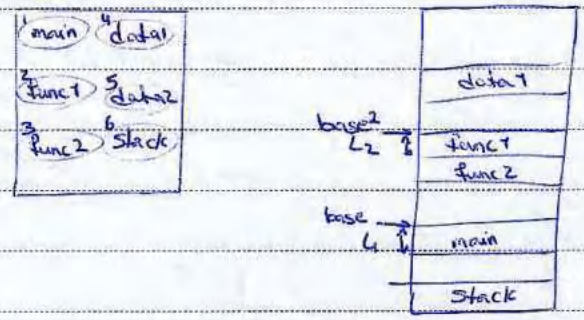
Subject:
Year. Month. Date. ( )

جلسه بیست و پنجم 9,26

مدیریت حافظه قطعه بندی Segmentation

این مدیریت مخصوص OS است و در جدول OS این کار را انجام می‌دهد. وقتی جدول باید پشتیبانی پردازنده را داشته باشد می‌گوییم CPU این اشکال را دارد.

← قطعه بندی از دیدگاه برنامه کاربری مخفی است ولی برعکس قطعه بندی با دیدگاه برنامه کاربری انجام می‌شود همین دیدگاه را می‌توان در مدیریت حافظه به کار برد می‌توان این برنامه را چند قطعه تقسیم کرد



یک سری شکل‌های پشت سر هم پیدا می‌کنیم که به هر کدام مستقل نگاه می‌کنیم ولی هر قطعه باید پیوسته باشد

این نوع مدیریت همچنان شکل fragmentation را دارد ولی کمتر شده چون در این جا فضاهای کوچک نیاز 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 هستند هر قطعه یک آدرس base و یک طول دارد در این جا می‌توانیم از شماره‌گذاری قطعه‌ها استفاده کنیم چون طول قطعه‌ها ثابت نیست

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

دانشکده مهندسی کامپیوتر و فناوری
دانشگاه صنعتی امیرکبیر

در پردازنده 8086 ← پدر بزرگ پردازنده Intel

نقطه مدیریت Segmentation است ولی خیلی ساده‌تر باید مشخص کنیم که کجاش code seg، stack seg، data seg ... است و این را ما تعریف می‌کنیم ← این خیلی نوع ساده‌ای است چون Seg های زیادی دارد.

این نوع قابل‌تر این است که بتوانیم هر چند تا قطعه که می‌خواهیم داشته باشیم.

Subject:
Year. Month. Date. ( )

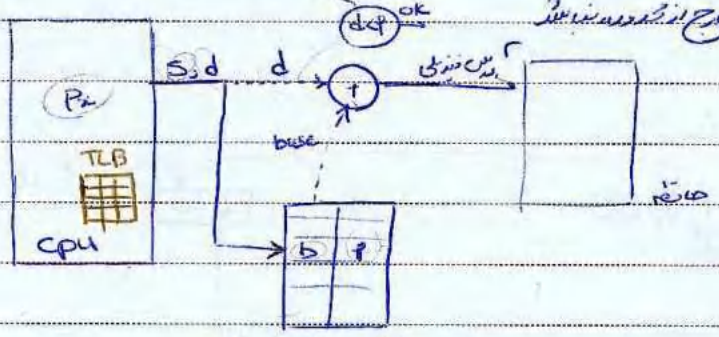
تبدیل آدرس در Segmentation

آدرس به دو بخش هستند ← کدام Seg ← این Seg را برنامه‌نویس می‌گوید

داخل این Seg کجاست.

در غیر این صورت اگر بیرون از محدوده‌ای بود وقفه تولید می‌کند

قبل از جمع d را باید چک کنند تا خارج از محدوده نباشد



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

این روش تقریباً مشکلات را دارد. Intel برای حفظ این ساختار و استفاده از paging این دو را با هم ترکیب کرده یعنی به جای page کردن آدرس‌ها، Seg ها page می‌کند. هر Seg را به صورت page در حافظه ذخیره می‌کند

TLB سه field دارد ← شماره Seg

base

limit

مدیریت paged Segmentation (ترکیب مدیریت قطعه‌بندی و صفحه‌بندی)

← مدیریت حافظه قطعه‌بندی صفحه‌بندی شده

هر Seg را به طور مستقل صفحه‌بندی می‌کنیم

Seg توسط برنامه‌نویس انجام می‌شود ولی paging به هم از دیدگاه آن مخفی است

دیگه Seg نداریم و تک تک به صورت page در حافظه قرار می‌گیرد

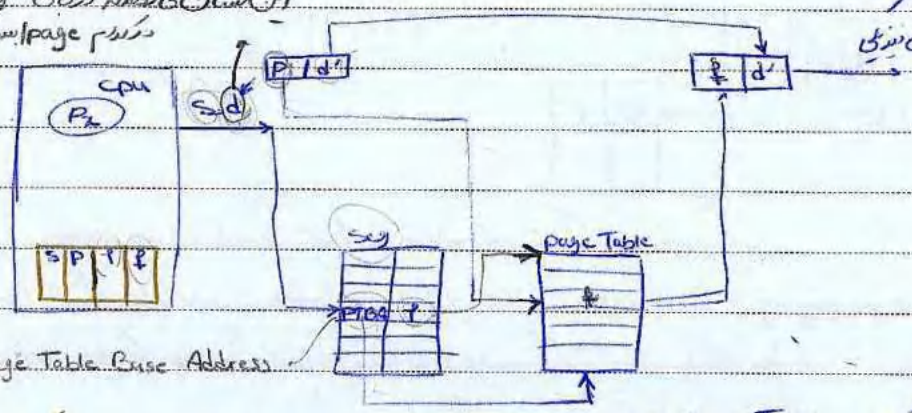
باشگاه مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

Subject:
Year. Month. Date. ( )

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

خروجی سه بخشی داریم.

این نشان می‌دهد در کدام Seg است و در کدام page است



S ← کدوم page Table بریم

باید طول رو هم چک کنیم که limit را رد نکند

در این جا برای یک فرآیند چندین page Table داریم که هر کدام برای یک Seg است ولی اگر paging خطی داشتیم برای کل فرآیند page Table داشتیم

در این جا مشکل سرعت که با حافظه بیشتر میشه ← Seg table

page " "

حافظه

برای حل از مدل‌های قبل استفاده می‌کنیم ← استفاده از TLB

Seg page f

برای متناظر نگه می‌داریم ← کدام page مربوط به این Seg

در TLB باید دنبال زوج (seg, page) بگردیم. ← قبلاً تشخیص page جستجو می‌کردیم

f در واقع تعداد صفحه‌های Seg است.

← خصوصیتش متغیر بودن است و می‌تواند تغییر کند

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

Subject:
Year. Month. Date. ( )

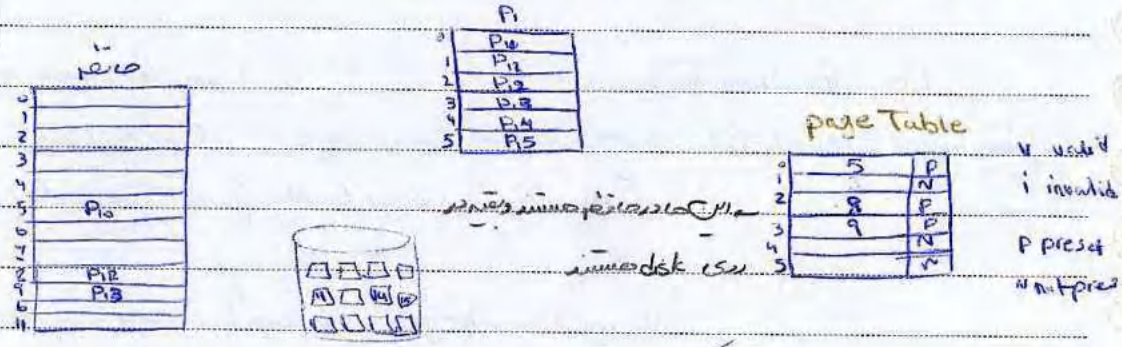
پردازنده Intel علاوه بر این این اجازه را می‌دهد که فقط از paging استفاده کنیم به این ترتیب که کل برنامه را یک قطعه تعریف کنیم ولی در اصل ما هم از قطعه‌بندی استفاده می‌کنیم.

← آخرین جواب همیشه چیزی که پیاده می‌کنیم paging است و خیلی از مشکلات را حل می‌کند

حافظه مجازی: Virtual Memory

تا الان فرض کردیم برای اجرای فرآیند باید تمام فرآیند داخل حافظه باشد (فرض محدودکننده‌ای است و چقدر هم بزرگ باشد چه فرآیند بزرگ می‌تواند آن را پر کند در نتیجه حداکثر چند فرآیند با هم اجرا می‌شدند!)

حافظه مجازی این فرض را نقض می‌کند ولی اگر باید لازم نیست کل فرآیند در حافظه باشد



← این‌ها در حافظه هستند و بقیه در روی disk هستند

اگر با توجه به اصل locality قطعه‌ای قسمت برنامه اجرا می‌شود، هر لحظه هر چند page که لازم می‌شود داخل حافظه می‌آوریم تا وقتی به بقیه احتیاج نکرده آن‌ها را وارد حافظه نمی‌کنیم یعنی در اصل بخش‌هایی از فرآیند در حافظه باشند که به آن‌ها لازم داریم به این منطق خیلی فرآیند می‌توانیم با هم اجرا کنیم

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

هر جدول جدول صفحه ساخته می‌شود (توسط OS) تا 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 واقعیت ندارد ولی بزرگ به نظر می‌رسد.

حافظه مجازی: حافظه بسیار بزرگی که واقعی نیست

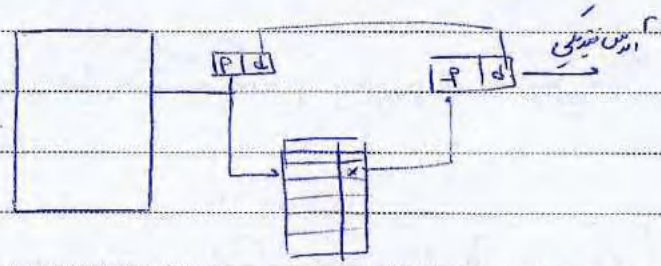
از بخشی از disk به عنوان حافظه مجازی دست برداشتیم چون در disk قطعه‌ها نیستند

دانشکده مهندسی کامپیوتر دانشگاه صنعتی امیرکبیر
اطلاعات دانشگاه صنعتی امیرکبیر

Subject:
Year. Month. Date. ( )

حالا تبدیل آدرس چطور انجام می‌شود؟ تبدیل آدرس توسط سخت‌افزار است

تبدیل آدرس تمامی با paging ندارد و مشکل یکسان دارد که چیک دارد



سخت‌افزار بیت valid توجه کن تا در صورت غیر valid بودن تولید وقفه کند

Page Fault (خطای صفحه)

مثلاً می‌خواهیم این دستور را اجرا کند

LD Rx, 34(Ry)

آدرس را تولید می‌کند می‌بیند که در page حافظه نیست پس دستور العمل می‌تواند اجرا شود پس اجرای آن قطع می‌شود و به سراغ بخشی از OS می‌رویم که page fault را handle می‌کند

1. اجرای دستور العمل دچار مشکل می‌شود / page در حافظه نیست!!
2. تولید وقفه page fault
3. اجرای برنامه قطع می‌شود و OS کار می‌کند ← سخت page fault در OS کار می‌کند

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

توسط OS انجام می‌شود:
- page را مشخص می‌کند و
- از disk به حافظه منتقل می‌شود
- باید page Table را اصلاح کند
- کنترل به برنامه کاربر برمی‌گردد

کنترل دوباره به برنامه کاربر برمی‌گردد و دوباره دستور انجام می‌شود

← موقع تبدیل آدرس OS نیست اما یک سخت‌افزار است که در صورت عدم انجام تولید وقفه می‌کند

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

برای بیرون کردن روش‌های مختلف مطرح است

برای TLB , cache

این سوال مطرح است

دانشکده مهندسی کامپیوتر و فناوری
اطلاعات دانشگاه صنعتی امیر کبیر

Subject:
Year. Month. Date. ( )

B

روش بهینه‌ای که استفاده می‌شود LRU است ولی در cache از روش‌های ساده‌تر مثل FIFO استفاده می‌شود

جلسه بیست و ششم ۸۷/۱۲/۱

می‌توان به طور غیرواقعی نشان داد که حافظه بزرگ است.

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

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

روش‌های جایگزینی: Replacement Policies

حافظه فیزیکی پر شود و برای گسترش آن از disk استفاده می‌کنیم اینکه چه کسی را بیرون کنیم تا فرآیند جدید را جایگزین کنیم این روش‌ها هستند.

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

رشته مرجع Reference string ، برای توضیح مثال

ارجاع به صفحه چاپ است. فرض کنیم سیستمی را manitore می‌کنیم تا مثال‌هایی که توضیح می‌دهیم را ببینیم

0 1 1 2 0 1 1 3 0 1 1 4 0 6 1 4 0 6 1 5 0 1 1 2 0 1 1 3

p d

این روش‌های جایگزینی

جاهایی که page عوض می‌شود می‌خواهیم که page fault نداریم.

d را حذف می‌کنیم

0 1 0 1 0 1 0 6 0 6 0 1 0 1

جاهایی که اشکال page fault هست را مشخص می‌کنیم

0 1 0 6 0 1 ← Reference string

این page fault است این نیست

اطلاعات دانشگاه صنعتی شریف

PAPCO

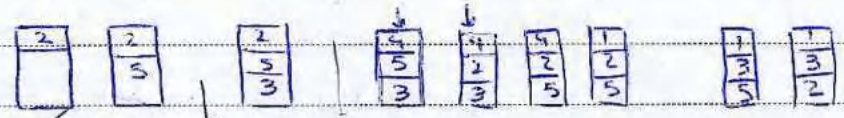
Subject:
Year. Month. Date. ( )

از این جا به بعد باید سیاست خاصی را به کار ببریم چون حافظه پر شده

مثال) یک رشته مرجع برای مثال از قبل بوده

2 5 2 3 5 4 2 5 1 2 3 2

سیاست FIFO



از قبل بوده مشکلی ایجاد نمی‌کند

اینجا جا نداریم و این خیلی بد است

که می‌خواهیم خارج شود در صورتی هنوز به آن احتیاج داریم

این روش خوب نیست چون در خیلی جاها تصمیم غلط می‌گیرد. پس به درد نمی‌خورد (در حافظه cache از این روش استفاده می‌کنند و اجرای این خیلی سریع است)

روش optimal (بهینه) ← نظر آینده

بر اساس چیزی که در آینده می‌بینیم عمل می‌کنیم خیلی است ولی برای بدست آوردن تقریب می‌سازیم

2 5 2 3 5 4 2 5 1 2 3 2

آینده را نگاه می‌کنیم اونکه از همه دورتر است را انتخاب می‌کنیم.

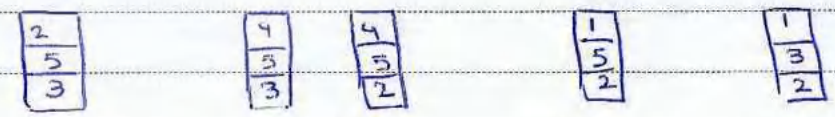
در صورتی که دو تا انتخاب داشته باشیم از روش FIFO بین آن دو استفاده می‌کنیم

برای تخمین این را می‌توان گفت که رفتار آینده مثل گذشته است و چیزی که در گذشته نزدیک اتفاق افتاده در آینده نزدیک هم اتفاق می‌افتد ← LRU

least Recently use (کمترین استفاده اخیر)

روش LRU ← با توجه به گذشته آینده را تخمین می‌زند

2 3 2 3 5 4 2 5 1 2 3 2



در این جا اگر تصمیم درست بگیریم زمانی ممکن است تصمیم غلط هم بگیرد ولی در کل بهتر از FIFO عمل می‌کند

PAPCO

دانشگاه مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

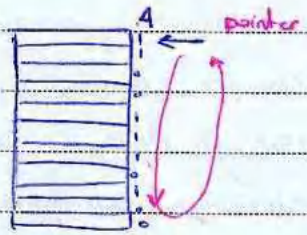
Subject:
Year. Month. Date. ( )

پیاده سازی این روش آسان نیست ولی خیلی شبیه stack عمل می‌کند ولی پیاده سازی سخت است و وقت گیر است این روش به همین دلیل کم از این روش استفاده می‌کنند

FIFO داشتیم FIFO در سیستم‌ها چون هیچ است استفاده می‌شود

optimal بهینه است و سعی می‌کنیم چیزی باشیم این بدست آوریم

LRU وقت گیر است ولی خوب عمل می‌کند



روش جدید استفاده: ساعت clocked

مختصات page Table ← frame

valid bit

Accessed یا A بیت ← این را اضافه می‌کنیم

هر page که جدید دسترسی باشد این set می‌شود

یک pointer می‌گذاریم که چرخی بزند اگر دسترسی شده باشد چرخی بزند و رد می‌کند اما قبل از اینکه جلو برود بیت accessed آن را صفر می‌کند اگر یک بود برگشت و صفر بود می‌خواهد که کسی از این استفاده نکرده پس این را انتخاب می‌کند و اگر کسی را انتخاب کرد بیت accessed آن را یک می‌کند اگر همه یک باشد یک دور بزند همه آن‌ها صفر می‌شوند

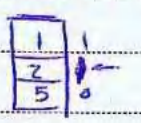
این LRU نیست ولی مثل آن است.

با یک بیت می‌توان LRU را استفاده کرد

2 5 2 3 5 4 2 5 1 2 3 2

در این که مثل LRU عمل می‌کند در این با صفحه مثل FIFO عمل کرده

در این مرحله page fault نیست چون رجوع به 2 داریم بیت accessed آن یک می‌شود



دانشگاه صنعتی شاهرود
دانشکده مهندسی کامپیوتر
کارگروه دانشگاه صنعتی شاهرود

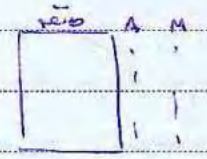
PAPCO

Subject:
Year. Month. Date. ( )

در این مثال عملکرد clock مثل LRU نیست و این طبیعی است چون جزء LRU نیست
در بعضی زمانها هم LRU هم clock خیلی بهتر عمل می‌کند

← clock پشتیبانی کمتا افزار برای خواندن چون بیت A باید به طور اتوماتیک set شود. هر چیزی که page Table اضافه می‌کنیم کمتا افزار باید پشتیبانی شود به همین دلیل این پشتیبانی برای خواندن
و reset شدن A توسط OS است به HW ربطی ندارد.

یک بیت دیگر می‌خواهیم به page Table اضافه کنیم:
بیت M: (Modify) اسم‌های مختلفی دارد در برگرداننده Dirty Intel ← D
← آیا محتویات page را تغییر داده‌ایم یا نه (خواندن و یا اینکه هم خواندن و نوشتن)
این هم HW این کار می‌کند و هر وقت نوشتن این خونه set می‌شود. OS می‌تواند این بیت را بخواند
(کارهایی که HW انجام می‌دهد ساده‌ها است و OS باید از این استفاده کند و تصمیم بگیرد)
← این خیلی کارهای مهمی انجام می‌دهد.



برای جابجایی یک صفحه ① قدیمی را روی disk بنویسیم
② جدید را روی صفحه بیاوریم
انتقال بین صفحه و disk خیلی زمانبر است
اگر بتوانیم اول frame را انتخاب کنیم که تغییر نکرده انتقال ① حذف می‌شود چون اول صفحه‌ای که
روی disk است با اول صفحه‌ای که روی صفحه است یکی است ← زمان خیلی کم می‌شود
HW این احتمال را به ما داده که تصمیم حالا می‌توانیم استفاده کنیم یا نکنیم
پس بهتر است اونی را انتخاب کنیم که access نشده و تغییر هم نکرده A=0 M=0
انتخاب دوم A=0 M=1

PAPCO

Subject:
Year. Month. Date. ( )

جلسه بیست و هفتم 87, 10, 3

مثال جلسه قبل.

احتمال page fault در حافظه مشترک‌ها

$$T_{eff} = 0.98(2ns + \underbrace{50ns}_{15ns}) + 0.02(1 - \frac{p}{0.02})(2ns + \underbrace{50ns}_{15ns} + \underbrace{50ns}_{15ns}) + \frac{p}{0.02}(2ns + \underbrace{50ns}_{15ns} + \underbrace{10ms}_{\text{دیسک}} + \underbrace{50ns}_{15ns} + \underbrace{50ns}_{15ns})$$

cache باعث تغییر این عددها می‌شود.

اگر حافظه cache داشته باشیم تاثیرش چی است؟ $0.9(10ns) + 0.1(15 + 50ns)$

تحت تاثیر cache سرعت سرعت به حافظه اصلی است. هر چه سرعت به حافظه رسیدیم مقدارش را کاهش می‌دهد

درصورتی که نبود cache. Teff قابل مقایسه بود با 50 ولی الان قابل مقایسه است با 15

حالا می‌خواهیم ببینیم که می‌توانیم 10ms را کاهش دهیم

اگر سرعت حافظه برای خواندن حافظه و نوشتن را نسبت صدی ¼ دارند (نوشتن، خواندن)

20% نوشتن و 80% خواندن

اگر بتوانیم از این استفاده کنیم می‌توانیم از بیت M استفاده کنیم و بجای حافظه‌ی استفاده کنیم که در آن نوشته نشده چون در این صورت قبل باید از روی disk بنویسیم

20% نوشتن ⟶ 10ms

قبل باید از روی disk بخوانیم و بعد بنویسیم 5ms ⟶ 80% خواندن

اگر صفحه انتخاب شد برای جای تغییر کرده باشد انتقال از روی حافظه به disk انجام می‌شود چون لازم نیست بیشتر وقت جای توان این صفحه را انتخاب کرد ولی همیشه هم می‌توان این کار را کرد اگر بتوان همیشه

این کار را کرد بجای از سیستم عمل نوشتن انتقال در طرفه انتقال یک طرفه انجام می‌شود

$0.8(5ms) + 0.2(10ms) = 6ms$ => زمان بیشتر انتقال

با این کار می‌توانیم 10 را به 6 کاهش دهیم پس این باعث پیچیدگی سیستم می‌شود.

حافظه cache چون پیچیده می‌شود پیچیدگی ایجاد می‌کند

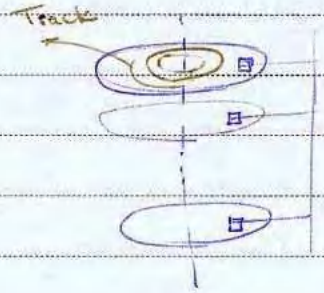
اگر $p = 10^{-6}$ قرار دهیم Teff را حساب کنید (نسبت قبل خیلی بهبود پیدا می‌کند.)

دانشکده مهندسی کامپیوتر و فناوری اطلاعات صنعتی امیر کبیر

Subject:
Year. Month. Date. ( )

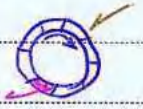
مدیریت دیسک

Disk ساختاری تودرتو دارد که بهبود پیدا می‌کند. بهبود برای ظرفیت، سرعت و ... است



هر کدام یک هد دارند که به بالای هم یا پایین دیسک تماس دارد
این هدها هم همگی در حال حرکت هستند
قاعده این است که برای این Track ها حرکت کنند

یک Track (شیار)



block یا sector (قطاع)

همین طور که این صفحه‌های چرخنده هد block های مختلف را می‌بیند واحد انتقال disk همین sector یا block است یعنی اگر بخواهیم اطلاعات را جابه‌جا کنیم یک block در block ... را جابه‌جا یا کند (واحد انتقال disk بیت نیست)

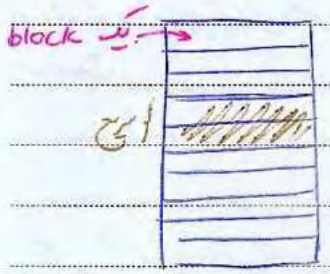
برای رسیدن به block به دو حرکت نیاز داریم ← انتقال: هد
چرخشی: block مورد نظر به هد برسد

Track هایی که شعاع یکسان دارند ← یک استوانه یا Cylinder

آدرس دهی اولین یک سیلندر تمام می‌شود و جهت سراغ جای دیگر رود چون می‌خواهند حرکت انتقالی را کم کنند

disk مکانیکی است و حرکت باید در آن انجام شود ← سرعتش خیلی کم می‌شود

جدول و کتاب block های OS است



می‌توانیم disk را این طوری ببینیم
در این صورت جایشان در حافظه ثانویه دیسک دوتا block مختلف
باهم فرقی نداشت اما در disk اگر block ها پشت سر هم باشند 4
دیسک خیلی سریع می‌شود اما اگر پشت سر هم نباشد باید کل حرکت مکانیکی انجام شود تا به block بعدی
برسیم که این خیلی بد است. یعنی اگر فایل‌های دیسک به محل block مشکلی دارد.

نشریه مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

Subject:
Year. Month. Date. ( )

اگر blocks های که نیاز دارند پشت سرهم اجرا شوند در جاهای مختلف disk قرار گیرند خیلی سرعت را پایین می آید. و اگر بخواهیم این نوع blocks را در disk پشت سرهم قرار دهیم دچار Fragmentation می شویم.

← مشکل disk اینست که شکل به شکل فرق دارد
از طرفی ترنسفر که به disk بالا دستی داشته باشیم بلکه اون حالتی است که همیشه به آن سنتری داریم

مثال) زمان سیک (seek) ← حرکت هد (بستگی دارد disk به ساختار)
" چرخشی circular ← زمانی که طول می کشد تا به سر block برسد
" انتقال transfer ← زمانی که انتقال یک blocks بین disk و حافظه صرف می شود

زمان سیک (میانگین) = 10ms ← مشخصه هر disk است و محاسبه نمی کنیم
چرخشی = 10000 دور در دقیقه (به سرعت بستگی دارد (چند دور در دقیقه می زند))

10000/60 rps ← $\frac{60}{10000} = 6ms$ (زمان یک دور)

میانگین چرخش نصف این است چون بعضی وقت ها بیشتر از نصف و بعضی از وقت ها کمتر از نصف
پس چرخش میانگین آن می شود 3ms

زمان انتقال = می توان از روی زمان چرخش بدست آورد ← $\frac{3ms}{n}$
n sector / track

هر sector یک block، 512 byte است و 320 $\frac{sector}{track}$

2560 sector

فرض کنید فایلی که اندازه اش 1.3MB است را می خواهیم بخوانیم کل زمان انتقال را تخمین بزنید

بدون ← همه جا پشت سر هم
random ← همه

بین این دو خیلی حالت ها ممکن است به سه معنی یک سری جا پشت سر هم
یک سری random

1 Sector های پشت سر هم
2 " " تصادفی روی دیسک

خواندن یک شیار (راول)

$\frac{2560}{320} = 8$ شیار

10ms
3ms
6ms
19ms

$19^{ms} + 9^{ms} * 7 = 82ms$

باشگاه مهندسی کامپیوتر و فناوری اطلاعات مسعود مصطفی امیر کبیر

Subject:
Year. Month. Date. ( )

بدترین حالت اینست که همه Sectorها پخش شوند

19ms ← زمان خواندن اولین شیار

زمان خواندن اولین sect، چسبت random

10ms

3ms

6ms/320 ← زمان لازم برای اسکن یک سکتور

$(10ms + 3ms + \frac{6ms}{320}) \times 240 = 33.32 s$ زمان خواندن یک فایل

در حالت اول و در حالت دوم را خواهیم داشت بلکه بیشتر نزدیک به حالت اول خواهیم بود

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

Subject:
Year. Month. Date. ( )

جلسه بیست و هشتم 87,10,7

زمان بندی دیسک : Disk Scheduling

چندین درخواست به OS برای دیسک رسیده. اینجا چه طوری جواب داده می‌شوند؟ به ترتیب جواب می‌دهند؟

مثال) درخواست‌های رسیده برای disk :

55 58 39 18 90 160 150 38 184 : شماره شیارها

راه‌هایی هست که این شیارها را به ترتیب آسان جواب دهند

می‌توان راه‌هایی پیدا کرد که این ترتیب را بهم زده و کمی میانگین زمان را کم کرد.

راه اول) head روی شیار 100

درخواست برای شیار 55 ← Seek time

حرکت چرخشی برای همه یکسان است

میزان حرکت به این شیار برسد

FIFO

(45 + 3 + 19 + 21 + 72 + 70 + 10 + 112 + 146)/9 = 55.3

که به طور میانگین این مقدار حرکت کرده تا به شیار مورد نظر برسد

FIFO هیچ تلاشی برای بهتر کردن میانگین ندارد و دنبال این است که به کسی که زودتر آمده زودتر جواب دهد

راه دوم) Shortest Seek Time First

این درخواست‌ها رسیده روی تراکم ببینیم که کدام شیار به جایی که ما الان هستیم نزدیکتر است مثلا

الان روی 100 هستیم باید ببینیم کدام به ما نزدیکتر است. در این جا نیازی به تخمین نداریم

ترتیب انتخاب شیارها

SSTF 90 58 55 39 38 18 150 160 184

روی هم می‌کنند → (10 + 32 + 3 + 16 + 1 + 20 + 132 + 10 + 24)/9 = 27.5

disk مکانیکی است و با حرکت سروکار دارد. در حد روی این روش جهت حرکت جدی می‌تواند باشد و جهت حرکت

می‌تواند عوض شود. در وسیله مکانیکی اگر دائما جهتش عوض شود دیتا مشکل پیش می‌آید

1. در قرون جهت حرکت نیروی زیادی به آن وارد می‌شود و بعد از مدتی از بین می‌رود

2. در حد و طرف بیشتر فضایی در پیرامون شیار داشته باشد

اشکال این دو روش، جهت حرکت هر لحظه می‌تواند تغییر کند

دانشگاه صنعتی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی شریف

PAPCO

Subject:
Year. Month. Date. ( )

SCAN

برای حل این مشکل راهی پیش نهاد داد که جهت حرکت تغییر کند

اگر در یک جهت حرکت می‌کنی به همان جهت حرکت کن تا آخر بروی و سپس به جهت مخالف برگردی و بقیه را انجام بده ← اگر به بالای دیسک تا آخر بروی سپس برگردی

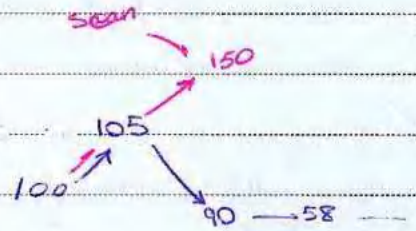
برگشتیم پایین بلافاصله کنی می‌خواهیم

بالای دیسک →

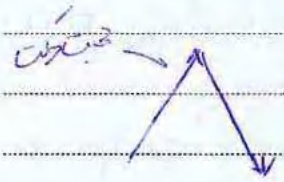
SCAN → 150 160 184 90 58 55 39 38 18

50 10 24 94 32 3 19=278

در SSTF اگر جهت‌ها ثابت باشد مثل حرکت SCAN است ولی اگر این جهت ثابت نباشد ممکن است دچار مشکلی بیاید که مجبور به تغییر جهت باشد.



• حالا اگر به این مثال بیاید 105 را اضافه کنیم



این جا قابل اثبات نیست که SSTF همیشه بهتر جواب می‌دهد.

با یکبار تغییر جهت این کار انجام می‌دهد

اشکال این روش سرعت از دست می‌رود.

در جهت بسیار زیادی دقیق حرکت کند که برسیاری بالا نیست

که ساختمان داده‌های دیگری برای حل این مشکل گفتند

دقت را در یک جهت می‌گذاریم

clock ← چون ساعت هم در یک جهت می‌چرخد

C-SCAN: این روش هزینه کمتری دارد

فرض می‌کنیم در جهت بالا دنده دقت دارد

C-Scan: 150 160 184 0 18 38 39 55 58 90

برگشتیم دور دقتی ندارد و به تر تیب می‌گردد و با حرکت زیاد هم این کار می‌کند

(50 + 10 + 24 + 184 + 18 + 20 + 1 + 16 + 3 + 32)

عدد خوبی بدست نمی‌آید چون برای 184 به 0 زمان زیادی صرف کردیم که خیلی زیاد است. در اصل در این مرحله سرعت خیلی زیاد است.

کانون آموزشی مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

Subject:
Year. Month. Date. ( )

Redundant array of Inexpensive (independant) disk : RAID

اطلاعات n تا داریم ← بیشتر از یک disk

bottle neck

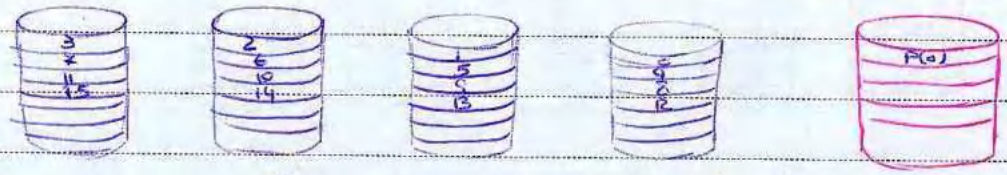
سرعت disk نسبت به سایر قطعات خیلی پرسرعت پایین است (گلوگاه سیستم است)

بیشتر در server ها کاربرد دارد ← چون دائماً باید سراغ disk بروند و درخواست اطلاعات پاسخ دهد.

مثل web server ←

جسم RAID اینست که بتواند سرعت را افزایش دهد.

$P_b(0) = b_0 \oplus b_1 \oplus b_2 \oplus b_3$



block های داده‌ی disk ها توزیع کرد که موقعی درخواست برای 5 و 6 آمد اینجا می‌توانند موازی انجام دهند اگر روی یک disk نباشند

← اطلاعات را روی چند disk پخش کرد تا بتواند خیلی از درخواست‌ها را به صورت موازی انجام دهد.

مشکل پیش می‌آید:

فرض کنیم احتمال خرابی یک disk ، P_0 باشد هر کدام از این disk ها خراب شود کار خراب می‌شود

احتمال اینکه یکی از چند تا disk خراب شود ← P

$P > P_0$ به راحتی می‌توان نشان داد.

قبلاً احتمال خرابی خیلی پایین داشتیم که الان این احتمال را افزایش داریم

برای حل این مسئله ← Redundancy در این جا وارد شد یک disk دیگر اضافه می‌کنیم که روی آن parity آن را ذخیره می‌کنیم)

با اضافه کردن این دیسک هر کدام از دیسک‌ها که از کار بیفتد قابل بازیابی است

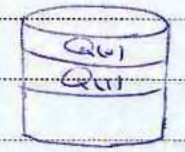
فرض کنیم b_1 از کار افتاده $b_1 = P_b(0) \oplus b_0 \oplus b_2 \oplus b_3$

اگر در یک disk از بین رفت اطلاعاتشان قابل بازیابی نیست و برای حل این باید اطلاعات بیشتر ذخیره کنیم

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

Subject:
Year. Month. Date. ( )

یک redundancy اضافه هم کار می‌کنند



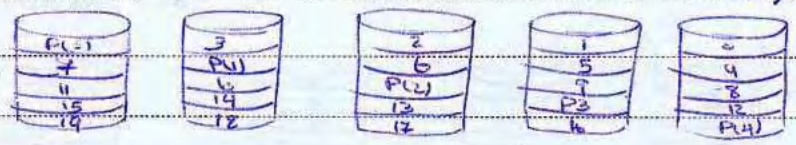
Q از راه دیگری محاسبه می‌شود

اگر بخواهیم قابلیت اطمینان بالا رو به همان اندازه با اطلاعات اضافی ذخیره کنیم

اشکال دیگری که داریم: تا write می‌شود یعنی علاوه بر block باید parity هم بنویسیم

RAID 5

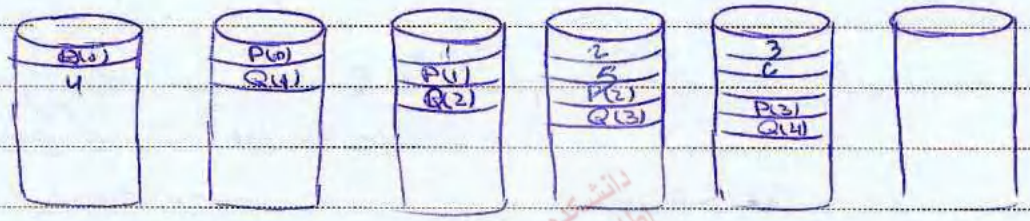
برای حل مشکل به ترتیب



مسئله سازی ساختن را در صورتی که به یک دیسک مختلف نگه کنند حل می‌شود

RAID6 : همان Q را داخلش جست‌وجو برای Server های مهم‌تر)

هم P و هم Q را چک می‌کنیم



برای Server های که در خیلی است زیادی را مثل سیستم‌های Transaction

Raid را می‌توان برای سیستم‌های بانکی که بر روی تعریف disk را افزایش داد برای این کار

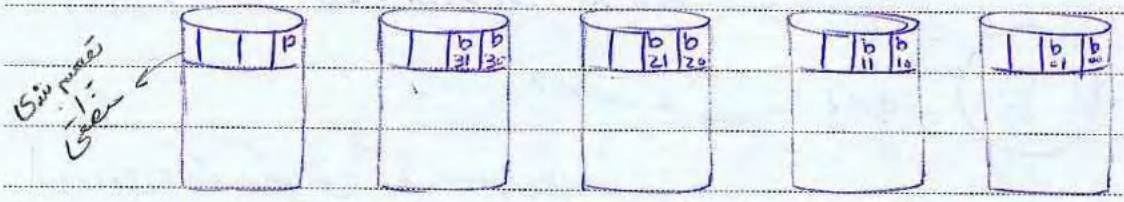
اطلاعات را باید خیلی زیاد پخش کرد

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

Subject:
Year. Month. Date. ( )

یک درخواست را سریع تر انجام می دهد

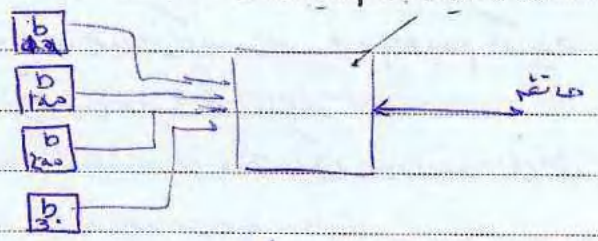
RAID 3 : سخت افزار پیچیده تر دارد که نحوه پخش شدن اطلاعات فرق دارد ← برای انجام یک درخواست



block ها را پخش کردیم و هر قسمت block در یک disk است

همیشه جمع یک block را می خواهیم به صورت موازی انجام می شود مانند هر چهار disk خوانده می شود

به صورت موازی انجام می شود

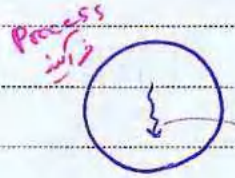


برای کارهای علمی و سیستمی ← محاسبات زیاد و دسترسی به block کم

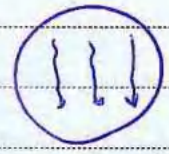
دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

Subject:
Year. Month. Date. ( )

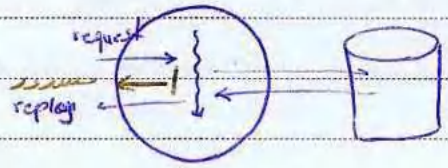
یک مسیر اجرایی که کدی را دنبال می‌کند را اجرا می‌کند، همان فرآیند است



آیا می‌توان در یک فرآیند بیش از یک مسیر اجرایی داشت؟



مثال: یک web server یک process است که درخواست‌های زیادی از آن می‌شود
یک request می‌گیرد و پردازش می‌کند و بعد request دوم را می‌گیرد
درخواست‌ها به ترتیب و به صورت سری اجرا می‌شود
این web server اطلاعات خیلی زیادی دارد که در حافظه جا نمی‌شود پس در disk است
درصدی که به سراغ disk می‌رود سرعت دری می‌شود و زمان
نسبت زیادی را باید صبر کند تا درخواست (اطلاعات)
برگردد



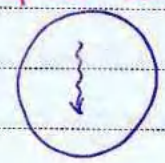
حالا به بحث اجرایی در نظر می‌گیریم. disk همان یکی است
یک Thread باید باشد که به بقیه فرق ندارد و درخواست‌ها به آن
می‌آید پس درخواست‌ها را پشت سر هم پشت‌سر هم می‌کند
فایده‌اش اینست که ببینید که اینجا آزاد است تا درخواست را
جلو ببرد
اجرایی اینکه یک کد را دنبال کنیم
وقتی request 2 می‌آید چون Thread آزاد داریم می‌تواند اجرا شود پس یکی از Thread های آزاد
می‌دهیم در اصل سرعت request
دستگاه disk سری است مگر اینکه چند تا disk داریم و اگر یک request سراغ disk رفت بقیه‌ی جا
می‌توانند قسمت‌های مختلف را که به disk ربطی ندارد انجام دهند

دانشکده مهندسی کامپیوتر و فناوری
دانشگاه صنعتی امیرکبیر

Subject:
Year. Month. Date. ( )

همه web server ها چنین Thread کار می‌کنند و سرعت بالایی دارند

مثال یک spread sheet را مشاهده کنید



محاسبات: اطلاعات را از من دریافت می‌کند و به جداول مربوطه می‌برد و آنها را update می‌کند

1. input / output
2. محاسبه و کمک به پردازش

← شامل محاسبه که بعد از عملیات ورودی خروجی انجام می‌شود

در حال گرفتن input کار دیگری انجام نمی‌دهد بعد از اتمام گرفتن input، محاسبات را انجام می‌دهد محاسبات تمام شده می‌تواند دوباره input بگیرد ← عملیات به صورت تکراری انجام می‌شوند

در حین محاسبه اطلاعات جدید آمده می‌تواند بگیرد و این دو عملیات می‌توانند به صورت موازی انجام شوند.

موازی { input / output
computation }

دیگر لازم نیست برای گرفتن input صبر کنند تا عملیات محاسبه تمام شود

به همین دلیل سیستم‌های امروزی این امکان را دارند که به صورت multiple thread کار کنند

برای اجرای process، مسیر را به آن می‌دهیم و PCB را set می‌کنیم و pc (program counter) را روی نقطه شروع قرار می‌دهیم تا برنامه را از آنجا اجرا کند پس در هر زمان تنها یک Thread اجرا داریم

یک کامپیوتر به شکل زیر داریم

در یک سیستم کامپیوتری می‌توانیم تعداد زیادی فرآیند را اجرا کنیم و کار کنند تا تمام شود

platform ← OS ، HW

← بر این اساس ما process تولید کنیم

دانشکده مهندسی کامپیوتر و فناوری
دانشگاه صنعتی امیرکبیر

PAPCO

Subject:
Year. Month. Date. ( )

← در یک platform (یک چیز خیلی) می‌توانیم چندین process تولید کنیم

← در یک process ... Thread ... "

آیا می‌توان گفت نسبت platform (سکوی پخش) به process مثل نسبت process است به Thread ؟ بله

← این جمله تقریباً درست است، دقیق نیست

process ها هر کدام مناخ خود را دارند یا هم مجزا هستند حتی حافظه آنها مشترک نیست پس هر کدام مناخ اختصاصی خود دارد ← به مجموعه مناخ از هم دیگر مجزا هستند بخصوص حافظه

← یک حافظه مشترک داریم که واقعاً مشترک نیست و هر کس بخش مجزای خود را دارد.

داخل PCB ← رجیسترها، stack، reg برای کنترل حافظه، PC، حسابرسی IO

accounting

برای اجرای یک فرآیند یک Thread داخلش هست جلوی می‌توانیم Thread اضافه کنیم ؛ یک Thread ها در مناخ process مشترک هستند

ایجاد یک Thread چطور:

یک سرویس OS است، همان طور که یک process می‌تواند است یک process دیگر تولید کند

← یعنی یک system call است

مناخ برای یک Thread:

هر process مناخی دارند که Thread ها نمی‌توانند از آنها استفاده کنند و در محدوده process کار کنند

یک زیرمجموعه‌ای از مناخ فرآیند را به کار می‌گیرد همان طور که process زیرمجموعه‌ای از مناخ سیستم را می‌گیرد

یکی از مزایای مهم نخ‌ها این است که خیلی کم‌هزینه‌تر از process است.

برای ایجاد process ← اختصاص دادن فضای حافظه جدید و مستقل

← هر چی داشته باشد process شبیه بوده

TLB در HW باید invalid شود (خالی شدن TLB)

بعد از (تا) فرآیند باید فضای حافظه آزاد شود

PAPCO