فصل ۱ - سیستم عامل

سیستم عامل یک واسط بین کاربر و سخت افزار کامپیوتر است. نرم افزار و اجزای برنامه های کاربردی با مدیریت منابع سیستم را فراهم می کند.

منابع ← CPU, RAM و ...

APP ← Kernel ← RAM
Device ← CPU

کرنل بخشی از سیستم عامل است که با منابع سخت افزاری و منابع نرم افزاری ارتباط برقرار می کند.

انواع سیستم عامل:
- تک پردازنده ای
- چند پردازنده ای
- بلادرنگ (Real Time)
- توزیع شده
- مجتمع کردن (کلستر)
- تعبیه ای

اشتراک منابع
نحوه استفاده از منابع

چند کاربره ← اشتراک زمانی (Time Sharing)

RR ← عنصر کلیدی از اجزای یک شبکه کامپیوتری

CLK/CPU ← تعداد واحد

بلادرنگ: در محدودیت زمانی مشخص نتیجه مورد نظر را ایجاد می کند / تعویض برای پرتگاهی / سخت زمان هستند / کنترل لوازم خانگی

deadline ← تا چه زمانی باید کار خود را به پایان برساند

سخت ← سخت زمان هستند

operating System Components

۱- مدیریت فرایند
۲- " حافظه اصلی
۳- " ذخیره ثانویه
۴- " فایل
۵- " I/O
۶-
۷-

اجزای تشکیل دهنده کامپیوتر

سخت افزار
- ۱- پردازنده
- ۲- حافظه اصلی
- ۳- وقفه های ورودی و خروجی
- ۴- اتصالات داخلی سیستم

سیستم عامل
- ۱- کاربر
- ۲- برنامه کاربردی

عمومی ↑ خصوصی

۱- پردازنده ← ALU, Control, Register

Program Status Word (PSW)	Fetch/decod/exe
Mode/Intrupt/Carry	

Kernel ← دسترسی به دستورات و حافظه
User ← محدودیت دسترسی

Tos در RAM-Reg ← کنترل کامل پردازنده

۲- تنظیم / Process order/ CLK
disable/Enable interrupts

Interrupt ← شناسایی کردن و اجرا برنامه ای دیگر و متوقف شدن CPU کار

Into..Rout ← روال اجرای وقفه

System call ← نرم افزاری / وقفه

Signal ← سخت افزاری

اجرا شود و بعد از اتمام وقفه CPU کار قبلی خود را ادامه دهد.

در هنگام رویداد وقفه PC بر روی stack ذخیره شود.

انواع وقفه ها:
- وقفه برنامه (Prog. check) ← سعت اجرای یک دستورالعمل
- درون زمان سیغ ← CPU بروی CLK
- ورودی و خروجی ← IO ← Toayout / Pointed
- نقص سخت افزار

overflow
devid by zero
- تلاش برای دستور خارج از اختیار کاربر
- مراجعه به آدرس خارج از فضای مجاز کاربر

Context Switch ← ذخیره و بازیابی وضعیت یک process تا بتواند بعداً ادامه اجرای process را بدهد.

این قابلیت از ارکان اصلی چندبرنامه گی است / قابلیت استفاده اشتراکی از CPU را می دهد.

System call ← واسط بین سرآیند و سیستم عامل که در C به صورت استاندارد شده و کاربر آن را می بیند.

طرق انجام IO:

- Program IO: چون CPU از IO به process نمی‌رسد و process را در حالت waiting قرار می‌دهد، CPU باید مرتب چک کند IO کارش تمام شده یا نه (Polling). این کار استفاده بهینه از CPU را از بین می‌برد چون CPU دائماً Polling می‌کند که کار تمام شد یا نه.
- Interrupt IO: CPU فرمان را صادر می‌کند و به فرایند دیگری می‌پردازد؛ IO موقعی که کارش تمام شد یک وقفه صادر می‌کند که این کار باعث می‌شود تا فرایند منتظر اتمام IO است CPU را بگیرد که کارش را انجام بدهد.
- DMA IO: مثال سال.

حفاظت:
- IO: همه دستورات ممتاز است و کاربر نتواند مستقیم اجرا کند و باید از طریق OS باشد.
- CPU: برنامه در حلقه گیر نکند (توسط تایمر).
- Memory: برنامه کاربر نتواند به حوزه سیستم عامل مراجعه کند و یا به برنامه دیگر (Reg Base/limit).

درس دوم: Process - Thread

فرایند (process): برنامه در حال اجرا
- محدود به CPU: بیشتر زمان صرف پردازش می‌شود.
- محدود به IO

PCB: هر فرایند یک PCB دارد که شامل تمام اطلاعات مورد نیاز CPU در مورد process است.
Process Control Block

1- حالت فرایند: اجرا / آماده / انتظار
2- PC
3- CPU Schedul
4- Memory Mang..: ثبات پایه، حد / صفحه / قطعه
5- Accounting: شماره فرایند / میزان مصرف CPU / محدودیت زمانی
6- IO status
7- CPU Regs

حالت‌های فرایند:

این dispatch شدن تحت حاکمیت برنامه‌ای که dispatch شود. CPU Scheduling

علل پایان فرایند:
1- پایان طبیعی
2- سقف زمان Session timeouts
3- گذشت زمان: انتظار زیاد فرایند برای یک اتفاق خاص (محدوده زمانی دارد)
4- نبود حافظه: mem تا نفس بگیرد
5- تجاوز از حدود: می‌خواهیم برویم تو حافظه OS
6- خطای حفاظت: می‌خواهیم منبع غیرمجاز بگیریم
7- " محاسباتی: تقسیم بر صفر / ذخیره عدد بزرگتر از حدود تحت اختیاری
8- " ورودی خروجی
9- دستور نامعتبر
10- " ممتاز
11- استفاده از نوع نامناسب از داده
12- دخالت OS به خاطر بن‌بست
13- پایان یافتن پدر
14- درخواست پدر

وابستگی: فرایندی که فرزند آن نمی‌تواند آنرا اجرا شود.

زمانبندی: کلیه اعمال چند برنامگی است / روی کارایی سیستم اثر می گذارد / فرآیند منتظر و اجرای را مشخص می کند.

زمان بند کار: ... به بلندمدت: که می باید توصیف آماده New-ready

به میان مدت: مدیریت کارایی که بین آماده و معلق و ایستاده (Swapping) Blocked-Ready

زمان بند پردازنده: ... به کوتاه مدت: که می باید تو CPU Ready-Running

به IOها

انتقال از RAM به HDD و برعکس را گویند.

Dispatcher (توزیع کننده): کنترل را به proc ای می دهد که Short Term sch انتخاب کرده است.

۱- Context Switch

۲- Change to User mode

۳- jump to pc

نخ thread: تعمیم فرآیند به کارهای کوچکتر و اجرای هم زمان به صورتی که هر کدام وظیفه خاصی را انجام دهد.

Code data files	Code data files		
register stack	Reg Stack	Reg Stack	Reg stack
Thread	Thread 1	thread 2	thread 3

نخ و فرآیند:

۱- یک فرآیند و یک نخ

۲- " " و چند نخ

۳- چند فرآیند و یک نخ

۴- " " و چند نخ

زندگی نخ به فرآیند وابسته است / تمام نخ های یک فرآیند شامل نخ ها، فایل ها مشترک اند.

تغییر متن میان ۲ نخ مربوط به فرآیندهای متفاوت باعث Context Switch بین ۲ فرآیند می شود.

نخ سطح کاربر:

os متوجه این نخ ها نیست / سریع در اجرا - ظرف و تغییر و cs در هنگام سازی است.

هزینه ایجاد نخ با هزینه به پایین زیر شته تعیین می شود.

cs با کمترین هزینه انجام می شود.

Context Switch = cs

ضعف:

۱- اگر یک pageFault رخ دهد تمام نخ های یک فرآیند block می شود.

۲- اگر یک system call blocking صادر شود تمام نخ ها block می شود.

۳- چون os از نخ های سطح کاربر آگاه نیست نمی تواند بین چند cpu پخش کند و زمانبندی کند.

نخ سطح کاربر: مشکلات قبلی را ندارد ولی هزینه cs بالاتر است. cs نخ ها ترکیبی = cs فرآیند باشد.

فصل سوم: زمانبندی

زمانبندی: تخصیص cpu به proc در طول زمان به صورتی که هدف های سیستم برآورده شود

معیارها:
1- بهره وری بالای cpu
2- توان عملیاتی بالا
3- کاهش زمان برگشت: از لحظه ورود تا لحظه خاتمه
4- کاهش زمان انتظار: مجموع زمان های ready

سیاست ها:
- Preemtive: فرایند می تواند توسط os متوقف یا به حالت ready برود
- non-Preemtive: نمی تواند توسط os متوقف یا ... تا proc تمام شود یا برای I/O خودش (نه توسط OS) block شود.

الگوریتم های زمانبندی:

1- FCFS: اونی که اول میاد سرویس می گیره.
2- RR: مثل FCFS است ولی برش زمانی هم تعلق می گیره.
3- SPN: اونی cpu می گیره که زودتر کارش تموم میشه. — shortest process Next
SJF — Shortest Job First
4- SRT: شبیه SJF (SPN) است ولی قبضه شدنی است. — Shortest Remaining Time
5- HRRN: اونی که نسبت پاسخ اش از همه بیشتر است cpu می گیره که به فرمول نیاز داره. — Hiest Response Ratio Next
6- MLFQ: بی خیال. — Multi Level Feedback Queue.

A/ FCFS: فرایندی که زودتر وارد ready شده یعنی بیشتر منتظر بوده cpu رو می گیره و توسط یه صف (FIFO) cpu رو می گیره.

Features:
1- انحصاری است (قبضه نشدنی است)
2- گرسنگی ندارد (Starvation) نهایتاً به همه cpu می رسه دیر یا زود داره موقت.
3- برای فرایند بزرگ کمتر از کوتاه است.
4- سربار کمتره چون یه بار cs میشه و تا انتها proc انجام می شود.

AWT (Averag wait-Time): زمان ورود تا زمانی که cpu می گیره
ready → Runing

$$\frac{\sum(\text{ورود} - \text{زمان اجرا} - \text{شروع})}{N}$$

OR

$$\frac{\sum(\text{زمان ورود} - \text{زمان شروع اجرا})}{N}$$

B/ LCFS: فرایندی که دیرتر اومده سرویس می گیره — قبضه شدنی / قبضه نشدنی

C/ RR: به اولین یا زودترین فرایند آمده سرویس می دهیم ولی یک برش زمانی خاص، اگر در آن برش کار proc تموم شد که هیچ در غیر این صورت برای دریافت دوباره cpu به انتهای صف ready می رود.

Features:
1- قبضه شدنی است
2- گرسنگی ندارد
3- در سیستم اشتراک زمانی خیلی موثره
4- برش زمانی کوتاه توان عملیاتی کم (همه زمان صرف cs می شود) ...
5- اگر برش زمانی از زمان اجرای proc بزرگتر صف ready بیشتر باشه چون FCFS رو داریم اجرا می کنیم.

مثلاً توی صف هم 20 تا نون داریم ... نون می گیرم 5 تا نون یعنی تیم کمتر می شه کمتر ملت نون گرفتنشون کامل می شود.

D/ SJF : فرآیند کوتاه‌تر همیشه میاد جلوی صف می‌ایستد.

Features :
۱- انحصاری (قبضه نشدنی)
۲- Starvation برای فرآیند بزرگتر دارد.
۳- میانگین زمان انتظار کمینه می‌شود.
۴- برای محیط اشتراک زمانی مناسب نیست.

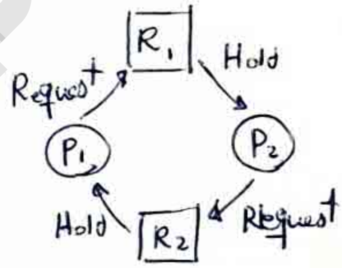
E/ SRT : SJF قبضه کردنی است.
فرآیند کوچک وارد شود فرآیند جاری قبضه می‌شود.

Features :
۱- انحصاری نیست (NP)
۲- Starvation دارد
۳- زمان کل کمتر از SJF است.

فصل چهارم: بن‌بست :

بن‌بست : مسدود بودن دائمی مجموعه‌ای از فرآیندها که برای دستیابی به منابع سیستم رقابت می‌کنند و این مسدود بودن تا زمانی که OS عملیات فوق العاده‌ای انجام دهد ادامه می‌یابد.

شروط بن‌بست :
باید حتماً هر ۴ تا باشد اگر نباشد بن‌بست رخ نخواهد داد



۱) نگهداری و انتظار (Hold & Wait) : یک سری منابع را گرفته و منتظر منبع ای است که آزاد نیست
۲) قبضه‌ای نبودن (Non-Preemtive)
۳) انحصار متقابل (Mutual Exclusion) : در هر لحظه تنها یک فرآیند می‌تواند منبعی را بگیرد. منبع اشتراکی نداریم
۴) انتظار چرخشی : منتظر منبعی است که توسط proc دیگری Hold شده و آن را رها نکند. حلقه می‌یابد و هر فرآیند Resource Hold کننده ای را می‌خواهد که در اختیار proc اول است

نکته : شرط ۴ رخ دهد شروط ۱ هم رخ داده است.

تشخیص بن‌بست از روی گراف تخصیص منبع :

۱- اگر چرخه نداشتیم بن‌بست نداریم / اگر چرخه داشتیم و از هر منبع چند تا بود احتمال وجود بن‌بست داریم ولی حتماً بن‌بست نیست.
۲- اگر n فرآیند و m منبع باشد اگر $\sum_{i=1}^{n} request(i) < n+m$ باشد بن‌بست نداریم.

روش‌های رفع بن‌بست

۱- prevention پیشگیری : کاری کنیم یکی از شروط بن‌بست رخ ندهد. (اجباری)
۲- Avoidance اجتناب : درخواست‌هایی که باعث بن‌بست می‌شود را انجام ندهیم. (از وقوع بن‌بست)
۳- Detection کشف بن‌بست : الگوریتمی توسط OS اجرا شود تا انتظار چرخشی را بیابد.

الف) Prevention :

چگونه موجب شد که فرآیندها ... : نگهداری و انتظار : فرآیند را مجبور کنیم منابع مورد نیازش را یکجا درخواست کند و تا همه‌ی منابع به او داده نشده فرآیند Block شود.

ورود کردن دسترسی‌ها باشه.

قبضه‌ای : اگر فرآیندی منبعی در اختیار دارد. درخواست منبع جدید داد اول همه‌ی منابع در اختیار رو بگیریم بعد منابع قبلی + جدید رو بهش می‌دیم (البته باید درخواست کنه).

انتظار چرخشی : به منابع یه order بده و اگه فرآیند منبع با شماره بالاتر را خواست بده اگر پایین‌تر را خواست ابتدا قبلی‌ها را بگیر بعد منبع جدید را بده.

باید آینده را بدانیم ...

ب) Avoidance :

رویکرد:
1- عدم شروع فرآیندی که ممکن درخواست‌هایش باعث بن‌بست شود.
2- عدم پاسخ به درخواست‌های منبع اضافی از طرف فرآیندی که با این تخصیص ممکن است منجر به بن‌بست شود (الگوریتم بانکداران)

محدودیت‌ها :
1- تعداد منابع باید ثابت باشد.
2- فرآیندی که منبعی در اختیار دارد نمی‌تواند خارج شود.
3- حداکثر منابع مورد نیاز باید از قبل معلوم باشد.
4- فرآیندهای مورد نظر باید مستقل باشند.

امتیاز : قبضه کردن لازم نیست بر خلاف کشف

حالت امن : حالتی از آن حداقل یک ترتیب از فرآیندها بدون ایجاد بن‌بست می‌تواند اجرا شود.
نا امن : وضعیت بن‌بست
XX

فصل ششم : همروندی :

جلسه نهم : همروندی :

همگام سازی (synchronize) : ترتیب انجام کارها باید رعایت شود. (درست)

شرایط رقابتی (race condition) : فرآیندها در فعالیت های بحرانی یک دیگر دخالت نکنند تا شرایط رقابت بوجود آید.

منبع بحرانی (Critical Resource) : منبعی که shared شود مثل چاپگر

ناحیه بحرانی (Critical Section) : قسمتی از کد برنامه که از منبع بحرانی استفاده می کند را گویند.

گرسنگی (Starvation) : فرآیندی اگر در دریافت منبع دچار تعویق شود یعنی منبع به فرآیندی دیرتر یا هرگز نرسد گویند

نکته : Inter Leave
یعنی امکان ایجاد وقفه یا
Switch در حین اجرای فرآیند

نکته : while (condition);
تا شرط false نشده اینجا گیر می کنیم (به)

شرایط برای همروندی درست :

۱- انحصار متقابل : Mutual Exclusion در هر لحظه تنها یک فرآیند وارد ناحیه بحرانی اش (جهت گرفتن منبع بحرانی) شود.

۲- پیشرفت progress : اگر فرآیندی در حالت اجرای دستورالعمل های عادی خود است نباید باعث جلوی ورود فرآیند دیگر به ناحیه بحرانی را بگیرد (سماجت نکنیم) (نون خودم نکنم تو هم نخوری) ❌

۳- انتظار محدود Bounded Waiting : مدت انتظار فرآیند برای ورودی به ناحیه بحرانی اش محدود باشد و دچار گرسنگی یا بن بست نشود.

به مدت نامعلومی منتظر ماندن ← (الف)
تا ابد منتظر ماندن

راه حل های ایجاد انحصار متقابل :

۱- نرم افزاری
۲- با حمایت سخت افزار (دستورالعمل خاص cpu)
۳- " " سیستم عامل (فراخوان سیستمی)
۴- " " زبان برنامه سازی (کامپایلر)

رویکرد نرم افزاری : الگوریتم دکر - مستقیماً توسط برنامه ها استفاده شده و وجود حافظه اشتراکی ضروری است.
از سمت ~~سخت افزار~~ یا سیستم عامل یا زبان های برنامه سازی حمایت نداریم.

Decker Algorithms

تلاش اول :

turn = 0

```
1  P0()
2  { while (true)
3    { while (turn != 0);
4      CS();
5      turn = 1;
6      nCS();
7    }
8  }
```

(line numbers as written: ۱ ، ۲ ، ۳ ، ۴ ، ۵ ، ۶ ، ۷ ، ۸ ، ۹ ، ۱۰)

```
1,  P0()
2,3 {while (true)
4,5  {while (turn != 0);
6,   CS();
7,   turn = 1;
8,   nCS();
9,   }
10, }
```

progress نداریم چون P1 اگر شروع کند در خط 5 گیر می کند تا P0 کپی را اجرا شد

Mutual نداریم اگر فرآیند P0 در خط 7 باشد و cpu را به P1 بدهیم هر دو در CS اند

تلاش دوم :

bool flag[2] = {False, False}

```
1,  P0 (Void)
2,3 {while (true)
4,5  {while (flag[1]);
6,   flag[0] = true;
7,   CS();
8,   flag[0] = false;
9,   nCS();
10 }
11, }
```

Mutual نداریم اگر فرآیند در خط 5 باشد وارد می شود و حالا فرآیند P1 اجرا شود و وارد ناحیه بحرانی شود P0 هم می تواند در ادامه وارد شود.

Bound نداریم P0 داخل می رود چون P1 flag[1] رو false است P1 داخل نمیره حالا نوبت P0 می شه ناحیه بحرانی رو انجام می ده قبل از انجام پرش دوباره می ره داخل

تلاش سوم :

```
1,  P0 (Void)
2,3 {while (true)
4,5  {flag[0] = true;
6,   while (Flag[1]);
7,   CS();
8,   flag[0] = False;
9,   nCS();
10, }
11, }
```

Mutual حل شده داریم.

Bounded نداریم بن بست رخ می دهد.
P0 بعد از خط 5 برش زمانی اش تمام می شود و P1 هم همین طور حالا دیگه هیچ کس نمی تونه وارد بشه.

تلاش چهارم :

```
1,  P0 (Void)
2,3 {while (true);
4,5  {flag[0] = true;
6,   while (Flag[1])
7,    {flag[0] = False;
8,    delay();
9,    flag[0] = true;
10,   }
11,  CS();
12,  flag[0] = False;
13,  nCS();
14, }
15, }
```

Mutual نداریم.

Bound نداریم ممکن است فرآیند بدون هیچ [illegible] یا مدت نامعلوم در delay گیر کند.
live lock : ← همان Live lock است.

تلاش پنجم :

turn = 0
flag[2] = false

```
P0 (void)
{ while (true)
  { flag[0] = true;
    while (flag[1])
    { if (turn == 1)
      { flag[0] = false;
        while (turn == 1)
          do;
        flag[0] = true;
      }
    }
    CS();
    turn = 1;
    flag[0] = false;
    nCS();
  }
}
```

Mutual داریم
progress داریم
Bounded داریم

Brife Decker Try.

تلاش	Mutual …	Progress	Bounded …
1	✓	✗	✓
2	✗	✓	✓
3	✗	✓	✓
4	✗	✓	✓
5	✓	✓	✓

الگوریتم Peterson :

```
P0(){
 while(true){
   flag[0]=true
   turn=0
   while(turn==0 && flag[1]);
   CS();
   flag[0]=false;
   RCS();
}}
```

عیوب الگوریتم های Decker و Peterson

۱- فرایندی در داخل ناحیه بحرانی از کار بیافتد فرایند دیگری تا ابد منتظر خواهد ماند.

۲- هر دو روش دچار انتظار مشغول است یعنی بیهوده در while گیر می کند تا نوبت اجرا شود.

انحصار متقابل ... روی کردهای سخت افزاری

نیازمند حمایت cpu است که ۳ روش داریم.

۱- وقفه را غیرفعال کنیم (با دستورالعمل) ← چون علا غیرفعال باشد برای زمانی نداریم که برم سراغ فرایند دیگه. یعنی در CS باشیم Critical Section ها Context Switch نداریم چون CS

۲- دستورالعمل TSL ← فقط در پردازنده جاری کاربرد دارد در Multi proc ها processor دیگر دسترسی ندارد.

۳- " " SWAP

struct Semaphore
{
  int Count;
  queue Queue;
}

جلسه ششم : سمافور : قدرت زیادی در انحصار متقابل دارد. ← یک ساختاری شامل ۲ فیلد

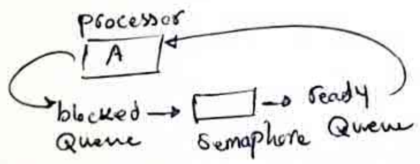
- شمارش تعداد wakeup های ذخیره شده برای استفاده بعدی
- صف نگهداری فرایندهای بلوکه شده

توابعی که روی سمافور اجرا می کنیم:

- wait : یک واحد از شمارنده کم می کند و اگر منفی شود فرایند را به صف مسدود می فرستد.
- Signal : افزایش یک واحد به شمارنده و اگر صفر یا منفی شود فرایندی را از صف مسدود بیرون می فرستد.

```
Wait(Semaphore S)
{
 S.Count=S.Count-1
 if(S.Count<0)
 {
  place this process in S.Queue
  block this process
 }
}
```

```
Signal(Semaphore S)
{
 S.Count=S.Count+1;
 if(S.Count<=0)
 {
  remove a process from S.Queue;
  place this process in ready queue;
 }
}
```



Wait ← اگر Count منفی شود فرایند جاری مسدود شود.

Signal ← اگر " " یا صفر شود یک فرایند از صف مسدود به آماده می رود.

انواع Semaphore
- ضعیف: اگر صف پشت نباشد. (گرسنگی داریم)
- قوی: اگر خروج از صف FIFO باشد.

انواع Semaphore
- عمومی: مقدار + و - و صفر می‌گیرد.
- دودویی: مقدار ۰ و ۱ می‌گیرد.

مثال: می‌خواهیم ابتدا S_1 بعد S_2 اجرا شود.

P_1	P_2
S1	Wait(s);
Signal(s);	S2;

مسئله تولید کننده و مصرف کننده

mutex ←--- ۱- یک Semaphore برای انحصار متقابل تولید کننده و مصرف کننده می‌خواهیم.

full ←--- ۲- " " برای شمارش پر بودن خانه‌ها (مقدار اولیه صفر)

empty ←--- ۳- " " " " " خانه‌های خالی (" " n) ← تعداد مکان‌ها

```
Producer(){
  int item;
  while(true){
    item = produce();
    Wait(empty);
    Wait(mutex);
      insert(item); // CS
    Signal(mutex);
    Signal(full);
  }
}
```

```
Consumer(){
  int item;
  while(true){
    Wait(full);
    Wait(mutex);
    item.remove();
    Signal(mutex);
    Signal(empty);
    Consume();
  }
}
```

مسئله غذا خوردن فیلسوف‌ها

مسئله انحصار متقابل فیلسوف‌ها با جواب‌گوی گرسنگی یا بن‌بست هم باشد.

```
Semaphore room = 4
Semaphore fork[5] = {1};
Void Semaphore (int i){
    while(true){
        think();
        Wait(room);
        wait(fork[i]);
        wait(fork[(i+1)%5]);
            eat(); // CS();
        Signal(fork[(i+1)%5]);
        Signal(fork[i]);
        Signal(room);
    }
}
```

← اجازه ورود بیشتر از ۴ نفر را نده در صورتی که ما ۵ فیلسوف داریم چون در این حالت که بیاد توی بخش یکی دو تا چنگال‌ها ... آزاد و تک... و ... گرسنگی یا بن بست رخ بده

در این راه حداکثر ۴ فیلسوف وارد می‌شود.

نکته: راه حل استفاده عدل راست ← به جای اینکه ۴ فیلسوف راه بده همه رو راه می‌دهد ولی برای فیلسوف پنجم میگه برعکس چنگال‌ها رو بردار یعنی همه راست چپ این یکی چپ راست هست.

راه حل نسا ... هم جای خود قابل تعمق است.

Visual studio
TeamWork ← این چطوری

مسئله خواننده‌ها و نویسنده‌ها

هرچقدر خواننده بخواد می‌تونه بخونه ولی برای نوشتن فقط یک نفر مجوز داره و اگر کسی شروع به نوشتن بکنه کسی نمی‌تونه بخونه

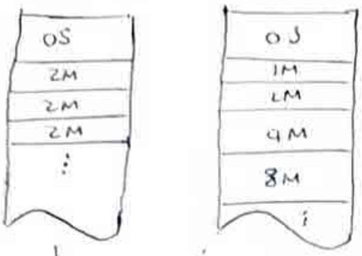
بنحیال شو... ☺

درس ششم : مدیریت حافظه :

یکی از وظایف سیستم عامل مدیریت حافظه است < RAM (اصلی) / Disk (جانبی)

تک برنامگی / چند برنامگی

چند برنامگی : با پارتیشن ثابت



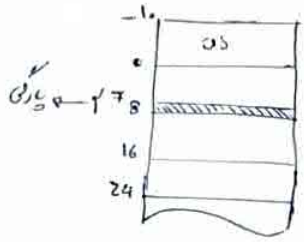
Static Partition معایب

۱- مشکل بودن تعیین تعداد و اندازه پارتیشن

۲- محدود بودن درجه چندبرنامگی به تعداد پارتیشن ها

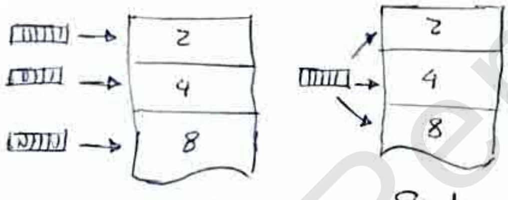
۳- تکه تکه شدن داخلی

تعداد برنامه هایی که در یک لحظه می‌تواند وارد سیستم شود.



پارگی داخلی

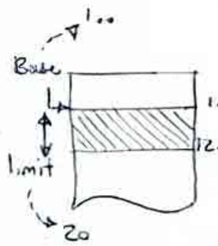
Internal Fragmentation



صف برای فرآیند جدید.

one queue per partition — Single queue

Base & limit Register : وقتی فرآیند cpu را گرفت آدرس شروع پارتیشن در Reg Base و طول پارتیشن در limit Reg ذخیره می‌شود



جلوگیری از جابه جایی (آدرس تولیدشده قبل از ارسال به حافظه با Base مقایسه می‌شود.

جلوگیری از تجاوز آدرس تولیدشده با limit مقایسه می‌شود تا به فضای خارج partition نرود.

Swapping (مبادله) : در سیستم های اشتراک زمانی نمی‌شود همه‌ی فرآیندهای فعال را در RAM جا داد. فرآیندهای اصلی تا زمانی که به آن ها نیاز داریم به حافظه‌ی اصلی منتقل می‌شوند و بعد به صورت Dynamic در RAM بارگذاری می‌شوند به این فرآیند Swapping می‌گویند. در Swapping وقتی فرآیند در RAM بارگذاری شود کی فرآیند خواهد آمد تا بعدی

مثلاً در RR وقتی Quantum تمام شود با فرآیند دیگر Swap می‌شود.

Increase utilization

Dynamic Partition : در این نوع پارتیشن بندی (تعداد / موقعیت و اندازه) پارتیشن ها متفاوت است. این انعطاف felexibility باعث افزایش بهره وری می‌شود

External Fragmentation : اگر فرآیند از حافظه بره جای خالی اون رو می‌گن.

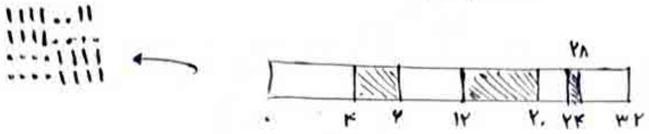
نکته : اگر یه proc از حافظه بیرون بره اگر Binding در زمان < Assembly/load : در جای قبلی / Run : در جای جدید

نکته: تکه تکه شدن خارجی به خاطر مبادله swap ها به وجود می آید.
" " قابل حل است ولی داخلی قابل حل نیست. برای از بین بردن تکه تکه شدن خارجی فشرده سازی انجام می دهیم. تمام حفره ها رو یک سمت می بریم تا یه حفره بزرگ درست بشه ولی این روش هزینه بر است.

تخصیص بخشهای آزاد حافظه در تخصیص پویا:

۱- با نگاشت بیتی
۲- " " لیست پیوندی

۱- نگاشت بیتی



پویا بیشتر — ایستا بیشتر

نکته: پارگی خارجی از داخلی — حفره

هر چه واحد تخصیص کوچکتر Bitmap ما بزرگ تر می شود. 1K→1M→1G
1G→1.24M→2^2KB

اگر بخواهیم n بیت وارد حافظه کنیم باید بگردیم و n بیت 0 متوالی پیدا کنیم.

۱- لیست پیوندی: بیخیال :)

الگوریتم های مکان یابی و تخصیص حافظه:

۱- First Fit: همواره از ابتدای حافظه شروع شده و در اولین مکان اندازه (حفره) قرار داده می شود.
۲- Next Fit: " از آخرین نقطه تخصیص شروع می شود.
۳- Best Fit: بهترین مکان را انتخاب می کند (کل لیست رو می گردد)
۴- Worst Fit: مثل ۳ — در بزرگترین حفره ای که جا می شود قرار می گیرد.

زمان بر چون کل لیست را باید بگردیم

نکته: در Next Fit حفره های بزرگتر ها سریع شکسته شده و برای ورود حفره های بعدی دچار مشکل می شویم

نکته: اگر لیست فضاهای خالی بر اساس اندازه فضاها مرتب شده باشد سرعت Best Fit بیشتر.

۵- Quick Fit: یه لیست داریم که در آن یه Pointer داریم به ابتدای خانه ها مثلاً ۴، ۸، ۱۲، ۱۶، ... کیلوبایتی حالا اگر فرآیند بیاد سریع می توانیم فضای مرتبط را اختصاص دهیم

Buddy System: طرحی برای حل مشکل تخصیص پویا و ایستا است که حافظه را همواره به توانی از ۲ تقسیم شود. ← اگر فرآیندی از ½ اندازه خالی بزرگتر باشد کل Block را بهش می دهیم ولی اگر کمتر بود بلوک خالی جاری نصف شده و ۲ تکه خالی از حافظه به صورت رفیق (Buddy) ایجاد می شود.

آدرس اصلی
↑
حافظه ی اصلی ← RAM
" منطقی ← که برنامه
↓
آدرس منطقی

Paging صفحه: حافظه اصلی را به بخشهایی به نام Frame تقسیم می کنیم.
" منطقی هم به بخشهایی به نام Page تقسیم می شود.

وقتی فرآیندی وارد حافظه شد تمام صفحات آن در قاب ها می نشیند و یک جدول به نام Page_Table ایجاد می شود که نگاشت یک به یک Page _ Frame را نشان دهد.

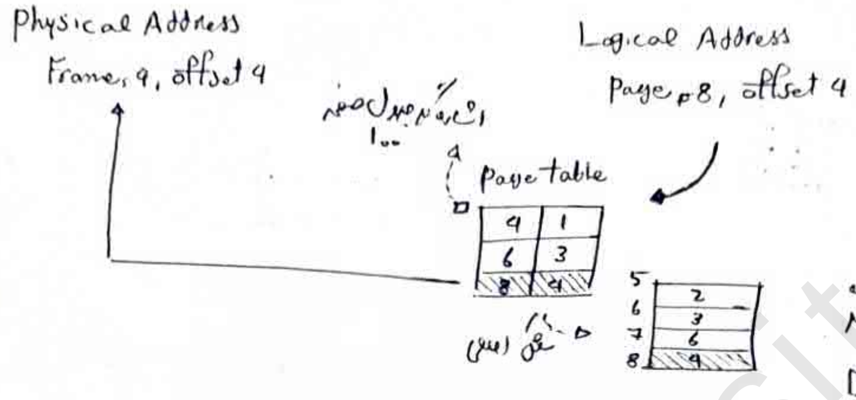
نکته: اندازه Frame، Page باید توانی از ۲ باشد.

نکته: تعویض صفحه صفحه‌بندی تکه‌تکه شدن داخلی است.

انواع آدرس:

آدرس منطقی: شماره صفحه و offset.

آدرس فیزیکی: شماره frame، offset



نکته: اندازه صفحه 1KB باشد

2^{10} مقدار می‌توانیم بدهیم

پس 10 B لازم داریم

قطعه‌بندی:

برنامه و داده‌ها به قطعه (Segment) تقسیم می‌شود و لزومی ندارد هم اندازه باشند. وقتی یک فرایند به حافظه وارد می‌شود کلیه قطعه‌های آن بار می‌شود. و جدول قطعه ساخته می‌شود.

هر سطر جدول قطعه ← شروع قطعه مورد تقاضا در حافظه اصلی و طول قطعه.

امتیاز قطعه‌بندی: حفاظت از قطعات و اشتراک داده‌ها و کد / حافظه مجازی / در الگوی صفحه‌بندی نمی‌تواند از دید کاربر نسبت به حافظه تطبیق شود / به خاطر قطعه‌های غیرهم اندازه شبیه به بخش‌بندی پویا است.

و قطعه‌بندی تمایل بیشتر کاربر برنامه‌ساز است و باعث تسهیل سازماندهی برنامه‌ها و داده‌ها می‌شود.

نکته: بخش‌بندی ایستا و صفحه‌بندی

پارگی داخلی

بخش‌بندی پویا و قطعه‌بندی

پارگی خارجی

حافظه مجازی: تکنیکی که موجب می‌شود فرایند بدون اینکه کامل در حافظه باشد اجرا شود.

با این روش می‌توان برنامه‌های بزرگتر از حافظه فیزیکی را اجرا شود.

چندبرنامگی را ممکن کرده و کاربر را از محدودیت‌های RAM رها می‌کند.

نکته: روش صفحه‌بندی یا قطعه‌بندی کل برنامه لازم نیست وارد حافظه شود فقط موارد مورد نیاز وارد می‌شود.

نکته: اگر آدرسی بخواهیم که در حافظه نیست به وقفه صادر می‌شود. فرایند مسدود می‌شود و تکه مورد نظر وارد حافظه می‌شود و OS ادامه می‌دهد.

روشهای پیاده سازی حافظه مجازی :
- صفحه بندی درخواستی
- قطعه بندی درخواستی
- قطعه بندی صفحه بندی شده.

صفحه بندی درخواستی : مثل صفحه بندی ساده است ولی تفاوت آن این است که نیازی به بارشدن کل page ها نیست.

Page fault : اگر MMU (memory management unit) به جدول صفحه ها مراجعه کند و بیت حضور و غیاب آن صفحه با ۰ یعنی صفحه در RAM نیست.

الگوریتم های جایگزینی :

۱- Belady optimal : صفحه ای که به مدت طولانی استفاده نمی شود بیرون می رود. واجرا کار نیست چون نیاز به پیش بینی آینده داریم.

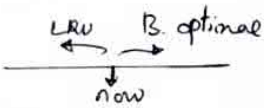
۲- NRU (Not Recently used) : بیخیال.

۳- FIFO : صفحه ای که بیشترین مدت در RAM بوده را بیرون می فرستیم

۴- الگوریتم دومین شانس : بیخیال

۵- ساعت : بیخیال

۶- الگوریتم LRU : برعکس Belady Optimal
(Least Recently used)



نقض Belady : هرچه تعداد قاب ها زیاد شود باید page fault کمتر شود ولی در FIFO همیشه این گونه نیست. ودر BO , LRU اینگونه نیست.

زمان ها در مدیریت دیسک :

Seek Time : زمانی که طول می کشد تا نوک Head به استوانه ای که داده در آن است برسد.

Rotational Latency : زمانی که طول می کشد تا نوک Head بالای ابتدای داده مورد نظر در اثر دوران دیسک برود