«بنام خالقم کار بالل پروانه‌هاست»

سیستم های عامل — استاد: فاطمه سپهری

تجدید عامل دکتر متصی ← اشتارات گسترش علوم پایه

کلام استالینگز ← ترجمه دکتر پیدام

@jozveotamrin

فصل ۱ مفاهیم سیستم عامل

① سیستم عامل چیست؟ واسط بین کاربر و سخت افزار:

GUI : Gratical Vser Inter face

② سیستم عامل به عنوان ماشین مجازی یا توسعه یافته:

vr.tuak or Extentde mackin

عملیات واقعی در سخت افزار انجام می شود و سیستم عامل آن را نمایش می دهد

کاربر با جزئیات عمل درگیر نمی شود. (توسعه یافته است)

③ سیستم عامل به عنوان مدیر منابع:

Ye source Manager

نظارت کردن و کنترل کردن کارتمایی نرم افزارها و سخت افزار صانعت نظیر سیستم عامل است

④ سیستم عامل به عنوان یک نرم افزار:

سیستم عامل بزرگترین نرم افزار یک سیستم کامپیوتری است و برنامه‌ای است که همیشه اول اجرا

Scanned by CamScanner

ی شود و آخر شام ی شود، سیستم عامل اجرای متوالی دارد (لایه لای برنامه ها اجرای شود) کسی سیستم عامل اجرای شود و کسی برنامه دیگر و معین رو به اجرا می یا به این علت نیس که مدام سیستم عامل یا مدام برنامه های دیگر اجرا شود.

مزایای به کارگیری سیستم عامل

۱) سهولت کار کاربر: چون ارتباط ما را با سخت افزار قطع می کند و کاربر شروع به کار با مدم افزار نخواهد کرد.

۲) سهولت پیاده سازی نرم افزار: به واسطه ی امکاناتی که سیستم عامل ارائه می کند می توان با برنامه هایی نظیر زبان عالی برنامه نویسی، نرم افزار و ... کار کرد.

۳) سهولت تولید و توسعه سخت افزار: اگر سخت افزار جدید تولید شود که با نرم افزارهای فعلی سازگاری نداشته باشد سیستم عامل این سازگاری را فراهم خواهد کرد.

۴) افزایش کارایی: سیستم عامل یک مدیریا برقرار کننده تعادل است در نتیجه کارایی افزایش می یابد.

کاربر سیستم عامل را به عنوان یک واسط ارتباطی بین نرم افزار و سخت افزاری بیند راز دید سخت افزار سیستم عامل مدیر منابع است.

چه اجزائی برای تولید یک سیستم عامل جدید دیده شده است؟ ۱) کار کاربر را راحت می کند.

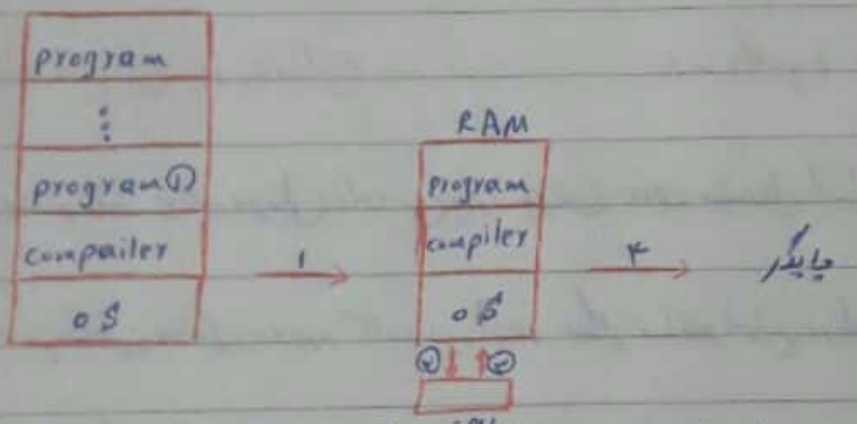
۲) کارایی سخت افزار را افزایش دهد.

Scanned by CamScanner

تاریخچه تولد سیستم عامل

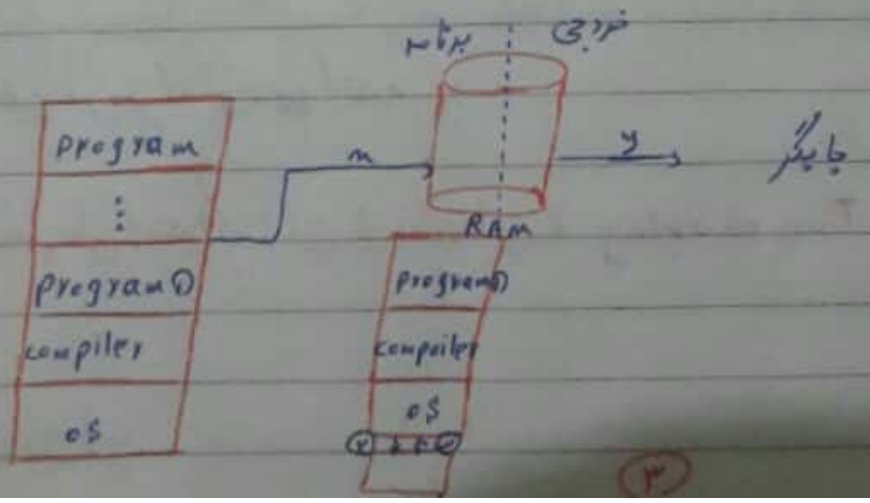
۱) سیستم های دسته ای (Batch systems) کارت پانچ

اولین چیزی که شد برای دسته سازی تولید کرد کارتنوان بود و هر ماه آن معادل یک بیت است سیستم عامل و کامپایلر این دستگاه نیز کارت بود که اول کارت سیستم عامل و بعد کارت کامپایلر را قراری دهد و سیستم عامل و کامپایلر به Ram منتقل می شود و سپس برنامه یکم وارد Ram می شود و بعد برنامه ها یک دهم وارد Ram نمی شوند. برنامه ها یکی یکی برای اجرا وارد می شوند و برنامه مورد نظر اجرایی شود و در نهایت خروجی به چاپگر فرستاده خواهد شد.



معایب: عملیات ورودی و خروجی همزمان انجام نمی شود و یعنی بیکاری در آن زیاد است و کارایی کاهش می یابد.

۲) تکنیک Spooling



Scanned by CamScanner

تکنیک spooling زمانی اجرا شد که دیسک مغناطیسی یا Hard disk ساخته شد تا بافت Hard disk

دستیابی تصادفی داشت یعنی به طور همزمان هم می تواند بنویسد و هم می تواند داده های مختلف.

برنامه را از کارخوان به Hard می برد بعد برنامه ها را یکی یکی از قسمت هارد وارد رم و CPU شده، اجرای شد و خروجی به Hard منتقل می شود و خروجی را به چاپگر منتقل می کند.

مزیت: ورود اطلاعات و اجرای برنامه ها و خروج اطلاعات همزمان اجرای شود در نتیجه بیکاری کاهش و کارایی افزایش می یابد.

سیستم های چند برنامه ای Multi program Batch systems

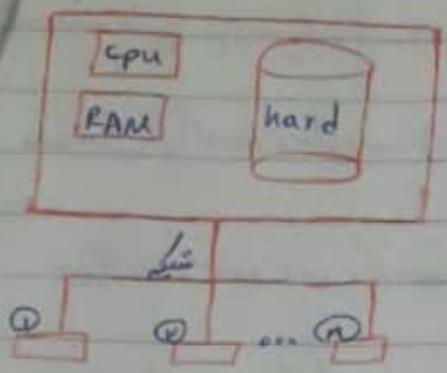
چند برنامه به طور همزمان وارد می شود تا وقتی CPU مشغول انجام وقفه است و یا اگر وسط برنامه به هر دلیلی مشکلی به وجود آمد پردازنده بیکار نماند سراغ برنامه بعدی برود در این سیستم ها بیکاری کم و کارایی افزایش می یابد و کاربر قطع و وصل شدن برنامه ها را متوجه نمی شود

به همین دلیل فکر می کند چند برنامه به طور هم زمان اجرای شوند.

با فقط یک CPU داریم پس چگونه چند برنامه اجرای شود؟ با قطع و وصل شدن برنامه ها که خیلی سریع اتفاق می افتد.

سیستم های اشتراک زمانی یا سازمانی: Time sharing

Scanned by CamScanner



وقتی هر کاربر اسم برنامه را وارد کند آن برنامه از hard وارد RAM می شود سپس سیستم عامل زمان را بین می دهد و در هر بین زمانی فقط یک برنامه حق اجرا شدن دارد سپس بین آن ها می چرخد ولی چون سریع اتفاق می افتد کاربر متوجه قطع و وصل شدن نمی شود. بدکاری کاهش و کارایی افزایش می یابد.

هم ترین دست آورد: استفاده چند کاربره از یک سیستم کامپیوتری

☑ زمان پاسخ در این سیستم ها باید کوتاه باشد.

☑ سریع کردن بین کاربران باید سریع اتفاق بیفتد.

سیستم های بلادرنگ → Real time
- بلادرنگ نرم
- بلادرنگ سخت

سیستم هایی هستند که برنامه ها برای اجرا ، deadline ، ضرب العجل دارند و باید سیستم عامل برنامه ها را دقیقاً در آن زمان اجرا کند.

بلادرنگ سخت: مانند پرتاب موشک

بلادرنگ نرم: اگر ضرب العجل رعایت نشود رضایتمندی کاربر کم می شود ولی مشکلی پیش نمی آید

Scanned by CamScanner

سیستم‌های توزیع شده: distributed systems

مجموعه‌ای از کامپیوترهای مستقل هستند که برای کاربران خود به صورت منسجم به نظر می‌آیند و تفاوت بین یک کامپیوترهای متصل و نحوه ارتباط آن‌ها با یکدیگر از دید کاربران پنهان است. اگر کامپیوتر شخصی جا نداشته باشد از طریق شبکه در کامپیوتر شخص دیگری ذخیره می‌شود ولی هیچ‌کدام متوجه این موضوع نمی‌شوند.

سیستم‌های چند وظیفه‌ای multitasking systems

برنامه به چندین قسمت شکسته می‌شود که به هریک از این قسمت‌ها وظیفه یا task می‌گویند.

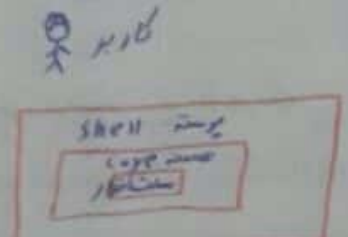
این قسمت‌ها نیاز است به سه حالت هم اجرا شوند:

۱. اگر یک cpu داشته باشیم هر task به یک قسمت از cpu واگذار می‌شود.

۲. اگر چند cpu داشته باشیم هر task به یک cpu اختصاص داده می‌شود.

☑ چون برنامه به چند بخش تقسیم شده زودتر اجرا می‌شود.

ساختار کلی هر سیستم عامل



* سیستم عامل * operating systems ← os

پوسته shell: دستورات ما توسط پوسته منتقل می‌شوند مثل دستور copy و past.

پوسته دستورات را به هسته منتقل می‌کند و هسته شروع به انجام عملیات copy و past می‌کند و در حال انجام عملیات، عملیات و نتیجه را نشان خواهد داد.

Scanned by CamScanner

هسته core : مدیریت حدی سخت افزار مثل رم، امکان دسترسی به همه منابع سخت افزاری و رم افزاری.

«پیام خدا»

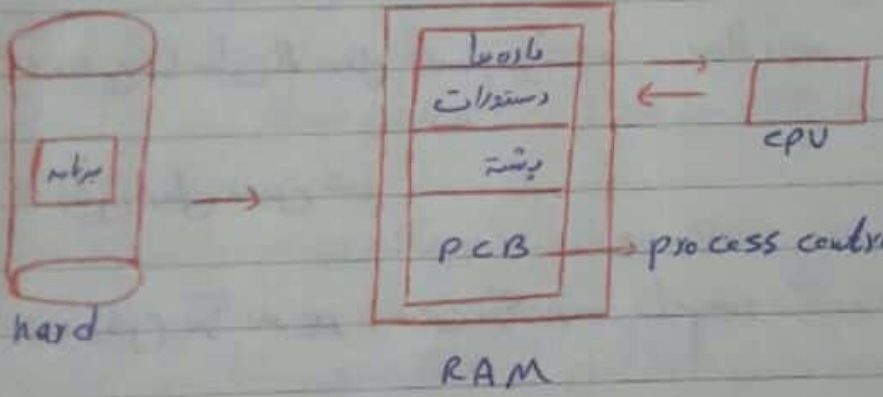
جلسه دوم

مدیریت فرآیندها processmanagment

تمام برنامه‌هایی داخل رم به عنوان برنامه‌ی در حال اجرا فرآیند نامیده می‌شود.

حالت برنامه و فرآیند ← ماهیت : برنامه غیرفعال (passive) و فرآیند ها فعال هستند Active

مکان : برنامه در هارد جانبی و فرآیند داخل حافظه اصلی RAM می‌باشد



پشته : متغیرهای محلی از آن گرفته می‌شود، آدرس برگشت در آن ذخیره می‌شود

pcb : تمام اطلاعات مربوط به یک فرآیند داخل PCB ذخیره می‌شود.

✓ دلایل ایجاد فرآیند

1. درخواست کاربر
2. ارائه یک سرویس توسط سیستم عامل

Scanned by CamScanner

۳) ایجاد یک فرآیند توسط فرآیند دیگر:

خیلی از فرآیندهای در حال اجرا فرآیندهای دیگری تولید می کنند که اصطلاحاً به آن زایش فرآیندی می گوییم. زاینده را فرآیند پدر و فرآیند ایجاد شده را فرزندی نامیم.

۴) برقراری ارتباط متقابله‌ای با سیستم:

وقتی کاربر جدیدی برای برقراری ارتباط با کامپیوتر اقدام می کند یک فرآیند ایجادی شود

دلایل خاتمه فرآیند

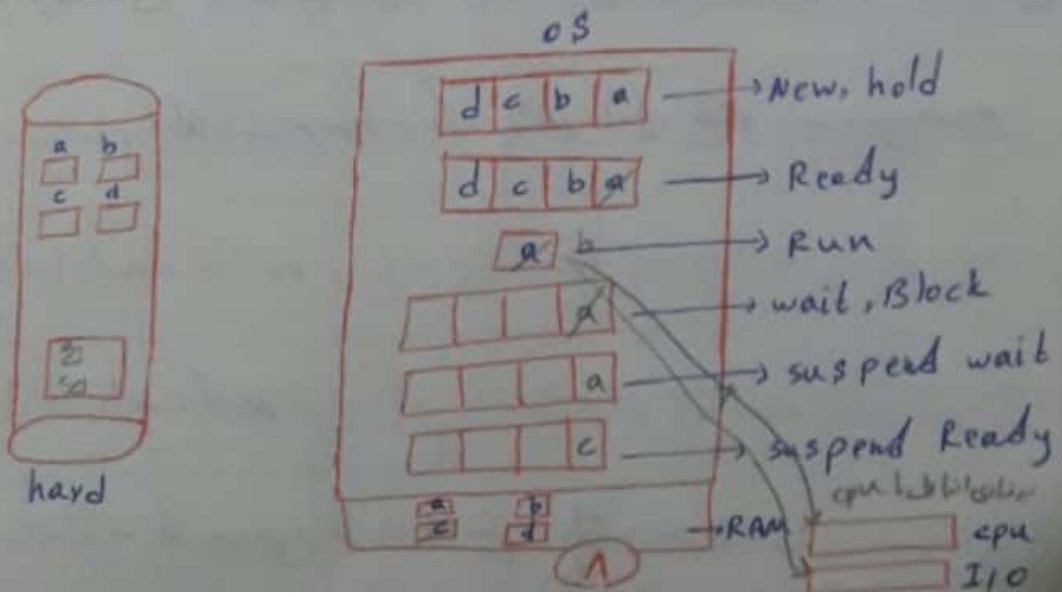
۱) درخواست کاربر

۲) پایان طبیعی: مثل فرآیند پخش کردن که پس از اتمام پایان می یابد.

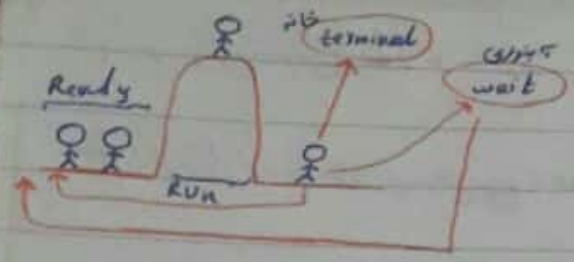
۳) خطای محاسباتی: مثل تقسیم بر صفر، overflow

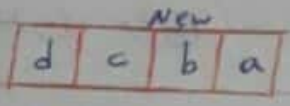
۴) پایان زمان تعیین شده

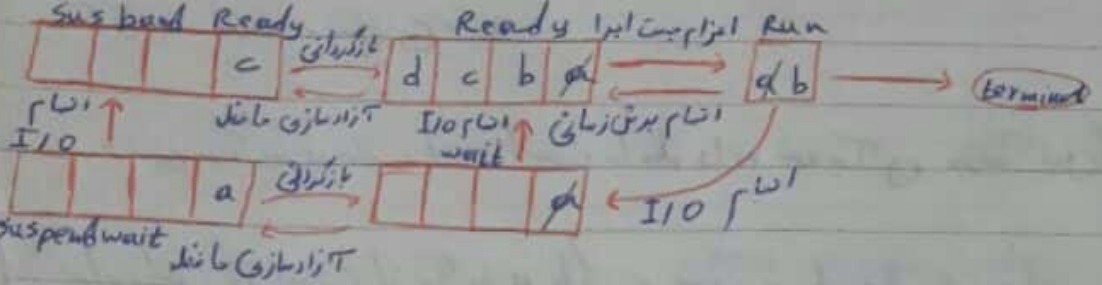
۵) پایان فرآیند پدر: وقتی فرآیند پدر تمام شود فرآیند فرزند نیز تمام خواهد شد که به آن سدل آبشاری می گوییم.



Scanned by CamScanner







* با صدا زدن کلیک کردن روی برنامه مسیر آن (اسم و آدرس) برای سیستم عامل فرستاده می شود

در سیستم عامل آرایه‌ای به نام Hold یا new ساخته می‌شود.

وقتی روی هر برنامه کلیک می کنیم اسم و آدرس آن برنامه داخل آرایه Hold قرار

می گیرد سپس برنامه ها وارد RAM می شود (به فرآیند تبدیل شده اند)

سیستم عامل یک آرایه دیگر به نام Ready می سازد که اسم و آدرس همه‌ی فرآیند در آن

قرار می گیرد. این فرآیند ها داخل حافظه اصلی یا RAM هستند و منتظرند که اجرا شوند

یکی از برنامه ها (a) برداشته می شود و اسم آن در Run قرار می گیرد سپس با

cpu ارتباط برقرار می کند در نهایت توسط cpu اجرا خواهد شد.

بعد از اجرای فرآیند ۳ حالت ممکن است رخ دهد:

۱) فرآیند بعد از اجرا خاتمه می یابد (وارد ترمینال می شود)

۲) برش زمانی اش تمام می شود که در این حالت به انتهای صف Ready برمی گر

Scanned by CamScanner

۳) دچار وقفه می شود.

☑ اگر دچار وقفه شود چه رخی می دهد؟ ارتباطی با CPU نخواهی شود چون کار آن با ورودی و خروجی است با I/O ارتباط برقرار می کند و فرآیند a از Run به آرایه wait منتقل می شود.

وقتی فرآیند در حالت wait است حافظه داده شده به آن موقتاً پس گرفته خواهد شد برای این کار قسمتی از فرآیند a پاک می شود در Hard قرار می گیرد و در عین حین فرآیند a از آرایه ی wait به آرایه ی suspend wait منتقل می شود.

suspend معلق: قسمتی از فرآیند را از RAM پاک می کند و به طور موقت در Hard قرار می دهد زمانی که RAM دوباره جا داشته باشد تا a را دوباره برمی گرداند.

حکمت: فرآیند در حال اجرا را نمی توان پاک کرد در نتیجه برای فرآیندهای داخل Ready می رود و یک قسمت از c را پاک می کند و آن را در Hard قرار می دهد در این حالت فرآیند c از Ready وارد suspend Ready می شود.

وقتی a دچار وقفه می شود برای بیکار نبودن پردازنده صف چک شده و b به اجرا درمی آید.

زمانبندی Schaduling

زمانبندی دارای چهار صف می باشد:

۱) فراهم کردن زمان مناسب برای کار

Scanned by CamScanner

۲) بالا بردن راندمان پردازنده

۳) استفاده از سایر منابع سیستم با یک راندمان خوب

۴) اجرای برنامه در موعد های مقرر تعیین شده توسط کاربر

✓ معیارهای امتیاز مثبت از دیدگاه کاربر

۱) کم کردن زمان پاسخ

۲) کم کردن زمان کل: مثل برنامه Nero

۳) قابلیت پیش بینی: یعنی یک برنامه در شرایط یکسان در زمان یکسان خروجی رو تولید کند.

۴) اجرای برنامه در موعدی که کاربر تعیین کند: مثل سیستم های بلادرنگ

✓ معیارهای امتیاز مثبت از دیدگاه طراح سیستم عامل

۱) توان عملیاتی: تعداد برداشتی هایی که در طول زمان تکمیل می شود $\frac{\text{تعداد کارهای انجام شده}}{\text{زمان انجام کار}}$ = توان عملیاتی

۲) استفاده از پردازنده: یعنی CPU در تمام زمان ها (تقریبا) مشغول باشد. هرچه استفاده از پردازنده بیشتر باشد طراح سیستم عامل به آن امتیاز مثبت خواهد داد.

۳) عدالت: یعنی هر فرآیند سهم مادلانه ای از CPU داشته باشد

۴) اعمال اولویت ها: گاهی اوقات خود کاربر می خواهد یک برنامه زودتر یا دیرتر انجام شود پس مکانیزم زمان بندی باید بتواند این موضوع را بپذیرد.

۵) تعادل نظام تحت بار سنگین: مکانیزم زمان بندی نباید طوری باشد که وقتی بار سنگین

Scanned by CamScanner

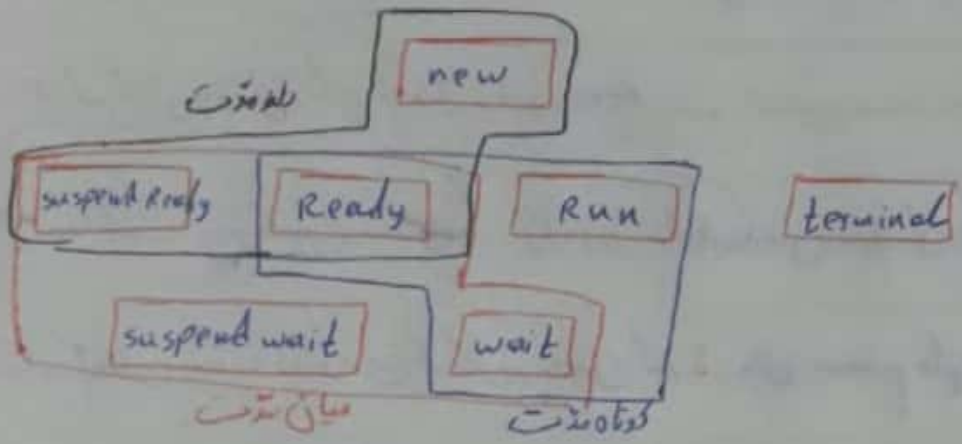
به آن وارد شود تا منتظم شود •

انواع زمانبندی

۱) بلندمدت یا زمانبند کار (long time scheduler)

۲) میان مدت (middle time scheduler)

۳) کوتاه مدت (sort time scheduler)



Scanned by CamScanner

«پیام خالق زیبایی ها»

جلسه ستوم

برنامه زمانبندی بلندمدت

برنامه ی زمانبندی بلندمدت پذیرش برنامه هاست و از بین کارهای درخواست شده تعدادی کار را انتخاب و تبدیل به فرآیند می کند.

چه زمانی برنامه ها را انتخاب می کند؟ به کارایی سیستم نگاه می کند اگر کارایی با این آمده به دلیل شلوغی باشد مدت برنامه جدید وارد نمی کند ولی اگر به دلیل بیکاری باشد چند برنامه را وارد می کند.

کدام برنامه ها را انتخاب می کند؟ ۱) FIFO (First In First out)

۲) SJF (shortest jobfirst)

۳) Mixed (ترکیبی)

۱) هر کدام که زودتر درخواست شده باشد را زودتر وارد می کند.

۲) اگر بیشتر زمان اجرای برنامه ها را نسبت به زمان اجرای کندترها جلوتر اجرا می کند.

۳) مرتب فرآیند cpulimit و Io limit را مشغول نگه می دارد.

Scanned by CamScanner

فرآیندها : ۱) cpu limit یا cpu bound سرعت‌ها سنگین دارد cpu را درگیر می‌کند.

۲) Io limit یا Io bound سنگینی دارد cpu کمتر درگیر است.

First in first out

این الگوریتم تابل پیاده سازی است عدالت هم دارد به شرطی که زمان اجرای تمام فرآیندها تقریباً یکسان باشد.

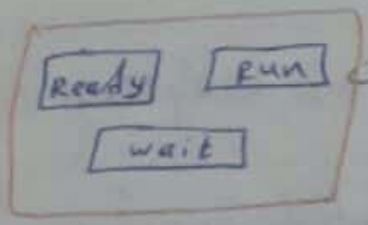
shortest job first

تعیین زمان اجرای برنامه‌ها امکان پذیر نیست در نتیجه این الگوریتم تابل پیاده سازی نخواهد بود. عدالت هم ندارد چون کارهای بزرگ دچار گرسنگی یا خطی زدگی یا starvation می شود.

Mixed

تابل پیاده سازی می باشد و عدالت هم دارد.

وظیفه زمانبند کوتاه مدت یا زمانبند cpu (cpu scheduler)



زمانبند کوتاه مدت وظیفه دارد یکی از فرآیندها را از آرایه Ready انتخاب کند (scheduler) و آن را به پردازنده برای اجرا بدهد.

چه زمانی باید یکی از فرآیندها را انتخاب کند و به پردازنده بدهد؟

Scanned by CamScanner

زمانی که پردازنده بیکار است.

چه زمانی پردازنده بیکار است؟ ۱) فرآیند درحال اجرا خاتمه یابد (دلخواه) Run → Terminal

۲) اجبار اتمام بودن زمانی Run → Ready

۳) فرآیند دچار وقفه (I/O) شود (دلخواه) Run → wait

۴) اجبار فرآیند دیگری وارد شود که اولویت آن نسبت به فرآیند در حال اجرا اولویت دارد.

انحصاری، بدون تبعه کردن، غیرقابل پس گرفتن، غیرانحصاری، تبعه کردن، قابل پس گرفتن

از بین ۴ حالت بالا حالت اول و سوم به دلخواه انجام می‌شود یعنی هرگاه پردازنده به دلخواه است ولی در حالت های دوم و چهارم هرگاه پردازنده به زور است.

اگر مکانیزم هرگاه به دلخواه فرآیند باشد به آن زمانبندی انحصاری یا غیرقابل پس گرفتن یا بدون تبعه کردن می‌گویند. اگر مکانیزم هرگاهی از هرگاه به دلخواه و هرگاه انحصاری نباشد به آن زمانبندی غیرانحصاری یا قابل پس گرفتن یا با تبعه کردن می‌گویند.

خصاریت مربوط به زمانبندی

۱) sleep: انتقال فرآیند از Run به wait

۲) wakeup: انتقال فرآیند از wait به Ready

۳) Dispatch: انتقال فرآیند از Ready به Run

۴) زمان بازگشت (Turn around Time): از لحظه ورود فرآیند تا لحظه خروج فرآیند.

Scanned by CamScanner

۵) زمان انتظار (waiting Time): مجموع زمان هایی که فرآیند در صف Ready منتظر بوده است.

۶) زمان پاسخ (Response time): از لحظه‌ی ورود فرآیند تا تولید اولین خروجی.

۷) تعویض متن (context swich): جابه‌جایی یا جایگزین شدن فرآیند در حال اجرا با فرآیند دیگر.

۸) اعزام کننده یا Dispacher: قسمتی از سیستم عامل است که در زمان تعویض متن اطلاعات مربوط به فرآیندی که باید متوقف شود را در PCB ذخیره و اطلاعات فرآیندی که در حال وارد شدن است را بارگذاری می کند.

الگوریتم زمانبد کوتاه مدت

۱) First come first service (FCFS): فرآیندها را به همان ترتیبی که وارد صف Ready شده‌اند انتخاب می کند برای اجرایی فرستد این الگوریتم اجرای است و نیازی به دانستن زمان اجرای فرآیندها ندارد و قابل پیاده سازی می باشد عادل است به شرطی که زمان اجرای فرآیند ها یکسان باشد در غیر این صورت به نفع بزرگترها می باشد.

۲) RoundRobin (گردش نوبت): زمان به قسمت های مساوی تقسیم می شود که به آن برش زمانی یا کوانتم می گویند (time slice).

فرآیند فقط به اندازه یک کوانتوم اجازه دارد در پردازنده بماند اگر بعد از این

Scanned by CamScanner

بدون زمانی اجرای فرآیند تمام شد به terminate می‌رود در غیر این صورت به انتهای صف Ready فرستاده می‌شود

نیازی به دانستن زمان اجرا ندارد و قابل پیاده سازی است. غیر انحاری است و عدالت دارد چون نوبت را بین فرآیندها رعایت می‌کند و گرسنگی و تعطلی زدگی ندارد. سربار و یا کار اضافه‌ای که برای سیستم می‌فرستد نسبت به FCFS کمی بیشتر است.

کم یا زیاد شدن مقدار کوانتوم چه تاثیری بر روی سیستم دارد؟

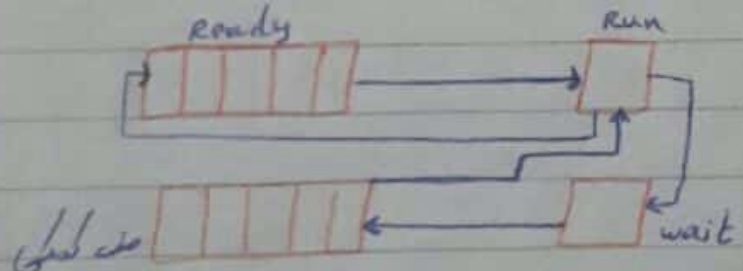
اگر مقدار کوانتوم کوچک باشد تعلق و حل شدن برنامه زیاد خواهد شد در نتیجه تا حدی دارد: ۱. پایین آمدن کارایی. ۲. بالا رفتن زمان انتظار کاربر.

اگر کوانتوم بزرگ باشد فرآیندهای بزرگ زمان بیشتری را به خود تعلق می‌دهند.

نکته ۱: کمترین کوانتوم، زمان تعویض متن است.

نکته ۲: ایده آل ترین مقدار کوانتوم، کمی بیشتر از کوچکترین زمان اجرا است.

گردش نوبت مجازی



گردش نوبت عدالت مجاز ندارد برنامه‌هایی cpu limit به ازای مقداری در پردازنده از کوانتوم استفاده می‌کنند اما برنامه‌ی IO limit به ازای مقداری کوانتوم کامل استفاده نمی‌کنند در نتیجه گردش نوبت مجازی مشکل گردش نوبت اصلی را

Scanned by CamScanner

حل کرد.

۱۳ SPN (Shortest process Next) شبیه SJF است با این تفاوت که شبیه SJF

SJF انتساب کردن از ست Hold بود ولی در SPN انتساب کردن از ست Ready بود

این روش انحصاری است یعنی اجرای خود آیدرا قطع نمی کند اجازه می دهد تا

تا تمام اجرا شود و عدالت ندارد و بابت کم بودن خود آیدهای طولانی می شود نیاز به دانستن

زمان اجرا دارد سربار آن از RunRobin و FCFS بیشتر است.

الگوریتم هایی که انحصاری هستند برای سیستم های اضطراری مناسب نیستند چون

این سیستم ها نیاز به قطع و وصل کردن دارند.

Scanned by CamScanner

«به نام خداوند بخشنده و مهربان»

جلسه چهارم

۴) SRTF (Shortest Remaining Time First)

Ready

C	B	A
6	5	8

$t=2$ → 0 وارد شد

B	D	C	A
3	2 ✓	6	8

D	B
2	3

✓

CPU

در SRTF وقتی پردازنده خالی است فرآیندی که کمترین را برخواهد داشت ولی اگر پردازنده مشغول باشد مقایسه می کند تا ببیند چه مقدار از فرآیند داخل CPU باقی مانده است اگر زمان فرآیند داخل Ready از آن کمتر باشد جایگزین فرآیند داخل CPU می شود و فرآیند داخل CPU (فرآیند قبلی) به انتهای صف Ready منتقل خواهد شد.

۵) HRRN (Highest Response Ration Next)

$$\text{Response ration} = \frac{\text{زمان انتظار} + \text{زمان اجرا}}{\text{زمان اجرا}} = \frac{S+w}{S} = 1+\frac{w}{S} = w/S \uparrow$$

سیستم عامل بر اساس کسر w/S فرآیندها را برای اجرا انتخاب می کند یعنی هرچه کسر بزرگتر باشد فرآیند برای اجرا الویت پیدا می کند فرآیندهایی که S کوچک یا زمان اجرای کوچکتری دارند زودتر اجرا می شوند اما فرآیندهای طولانی (S داخل صف منتظری مانند و به تدریج زمان انتظار آن ها (w) زیادی شود

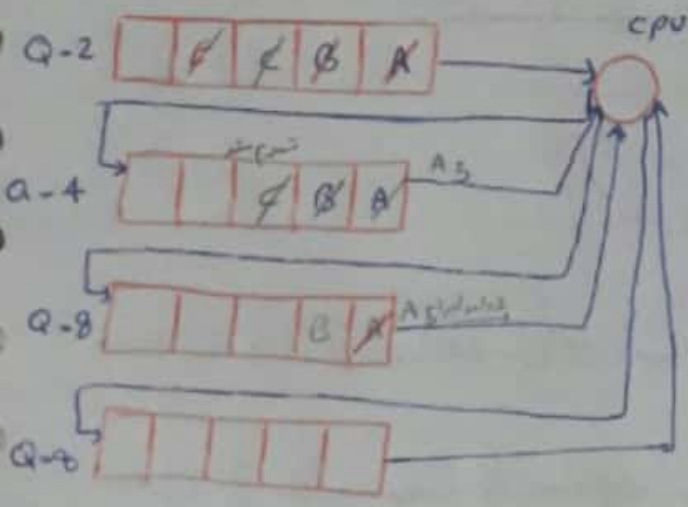
Scanned by CamScanner

و چون به صورت کسر اندی گذارد الویت آن ها را افزایش می دهد عین بالا می شود

نسبت به فرآیندهایی که تازه آمده اند الویت بیدا کنند.

نیاز به دانستن زمان اجرا دارد در نتیجه تابل پیاده سازی نیست و انتزاعی است.

۶) صف پسخورد چند سطحی (multi Level Feedback Queue)



برای هر صف یک کوانتوم می گذاریم که از بالا به پایین مقدار کوانتوم افزایش می یابد. صف Ready به چند صف شکسته می شود که هدی آن صف FirstInFirstOut می باشد. فرآیندهایی تازه وارد، وارد صف بالایی می شوند و هر فرآیندی که از صف انتخاب می کنیم به اندازه کوانتومی که برای آن تعیین شده است اجرا خواهد شد اگر کار آن تمام نشد وارد صف پایین می شود. نکته: ۱) تا زمانی که فرآیندی در صف های بالایی باشد اصلاً فرآیندی از صف های پایین اجرا نمی شود.

۲) اگر فرآیندی از صف پایین در حال اجرا باشد و فرآیند دیگری وارد صف اول شود فرآیند در حال اجرا قطع شده و به ابتدای صف خود برمی گردد و در دور بعد

Scanned by CamScanner

باقی مانده کوانتوم را دریافت می کند این روش نیاز است و نیازمند دانستن زمان اجرا ندارد در نتیجه قابل پیاده سازی است.

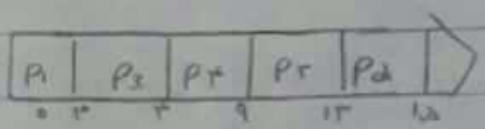
A و B در صف دوم بوده که C وارد شد در نتیجه A و B سریع کند تا به آن ها برسد.

A از صف سوم به پردازنده رفته و ۲ واحد اجرا شده است که در عین زمان E وارد صف اول شده است در نتیجه A متوقف شده به ابتدای صف خود برمی گردد تا در دوری بعد باقی مانده کوانتوم را دریافت کند.

FCFS

مثال ما: ترتیب اجرای فرآیندها را مشخص کنید.

پردازش	ورود	زمان اجرا
P_1	۰	۳
P_2	۲	۶
P_3	۰	۱
P_4	۱	۵
P_5	۴	۲



نمودار گانت

$\bar{T} = \frac{1}{n} \sum T_i$

زمان برگشت $T_i = O_i - a_i$

O_i زمان خروج فرآیند P_i ← a_i زمان ورود فرآیند P_i

$T_1 = 3 - 0 = 3$

$T_2 = 13 - 2 = 11$

$T_3 = 4 - 0 = 4$

$T_4 = 9 - 1 = 8$

$T_5 = 15 - 4 = 11$

$\bar{T} = \frac{1}{5}(3 + 11 + 4 + 8 + 11)$

$\bar{T} = \frac{37}{5}$

Scanned by CamScanner

$\bar{w} = \frac{1}{n} \sum w_i$

$w_i = O_i - a_i - S_i$: زمان انتظار

O_i زمان اجرای فرآیند P_i

$w_1 = 3 - 0 - 0 = 0$

$w_2 = 13 - 2 - 4 = 7$

$w_3 = 4 - 0 - 1 = 3$

$w_4 = 9 - 1 - 5 = 3$

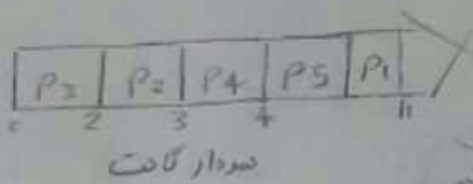
$w_5 = 15 - 4 - 2 = 9$

$\bar{w} = \frac{1}{5}(0 + 7 + 3 + 3 + 9) = \frac{22}{5}$

SPN

فرآیند	زمان ورود	زمان اجرا
P_1	0	4
P_2	1	1
P_3	0	2
P_4	1	1
P_5	4	3

مثال:



نمودار گانت

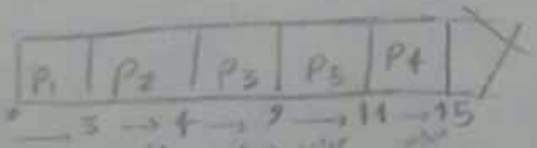
ابتدا زمان ورود را نگاه می‌کنیم و سپس زمان اجرا را. اون که کمتر بود رو اول داخل نمودار گانت قرار می‌دهیم.

HRRN

پردازش	زمان ورود	زمان اجرا
P_1	0	3
P_2	2	1
P_3	1	5
P_4	6	4
P_5	4	2

مثال:

P	زمان انتظار w	$\frac{w}{s}$
P_1	0	0
P_3	3-1=2	2/3
P_2	3-2=1	1
P_5	4-4=0	0
P_4	5-6=-1	



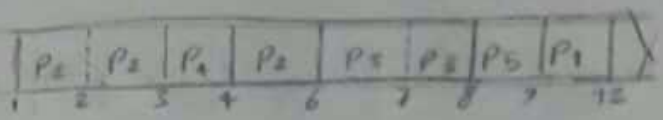
برای به دست آوردن زمان انتظار زمان ورود را از زمان وارد شدن کم می‌کنیم.

Scanned by CamScanner

PRTF

مثال:

پردازش	ورود	اجرا
✓ P_1	2	3
✓ P_2	1	~~4~~ ~~2~~ 0
P_3	4	~~2~~ 2 0
✓ P_4	3	1 0
P_5	7	1 0



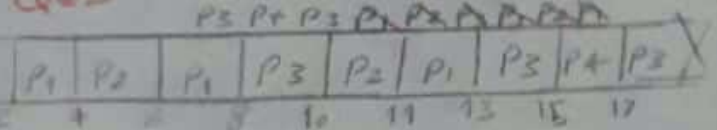
اجرا نشود

اولویت بالاتری کرد ظهر

مثال:

پردازش	ورود	اجرا
P_1	2	~~6~~ ~~4~~ 0
P_2	4	~~3~~ 1 0
P_3	6	5 5
P_4	12	2

FB
Q=2



«بنام عشق، به یاد عشق و در پناه عشق»

جلسه پنجم

MLFQ

نرم‌افزار	ورود	اجرا
A	0	6 ~~4~~ ~~2~~ 0
B	1	~~7~~ ~~6~~ 2
C	5	8
D	13	3
E	20	15
F	25	16

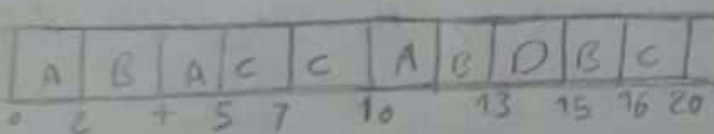
Q=2 Q=8 Q=∞

C ~~B~~ ~~A~~ ← Q_2

D C B A ← Q_1

C_2 B_1 ← Q_0

← Q_∞



مدیریت حافظه Memory management

۱) مدیریت حافظه: یک قسمت از سیستم عامل است که از بخش‌های پر و خالی RAM

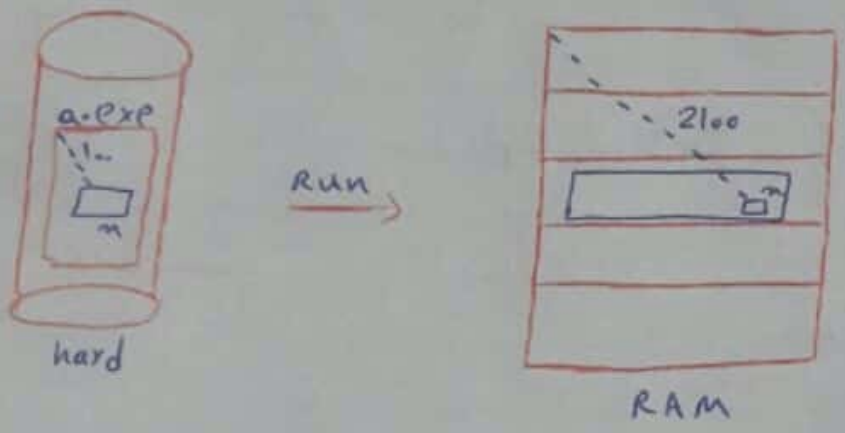
Scanned by CamScanner

خبر دارد و در زمان اجرای برنامه به برنامه حافظه‌ی دهد و زمانی که اجرا تمام شد حافظه را از برنامه می‌گیرد. کپی شدن برنامه داخل RAM یعنی دادن حافظه و بار کردن برنامه از داخل RAM یعنی آزاد کردن حافظه.

آدرس: محل متغیر توسط آدرس آن مشخص می‌شود.

۱) منطقی: مکان متغیر داخل برنامه (آدرس مجازی)

۲) فیزیکی: مکان متغیر در حافظه‌ی اصلی زمان اجرا (آدرس واقعی)



آدرس فیزیکی متغیر n = 2100

آدرس منطقی متغیر n = 100

تکنیک های مدیریت حافظه

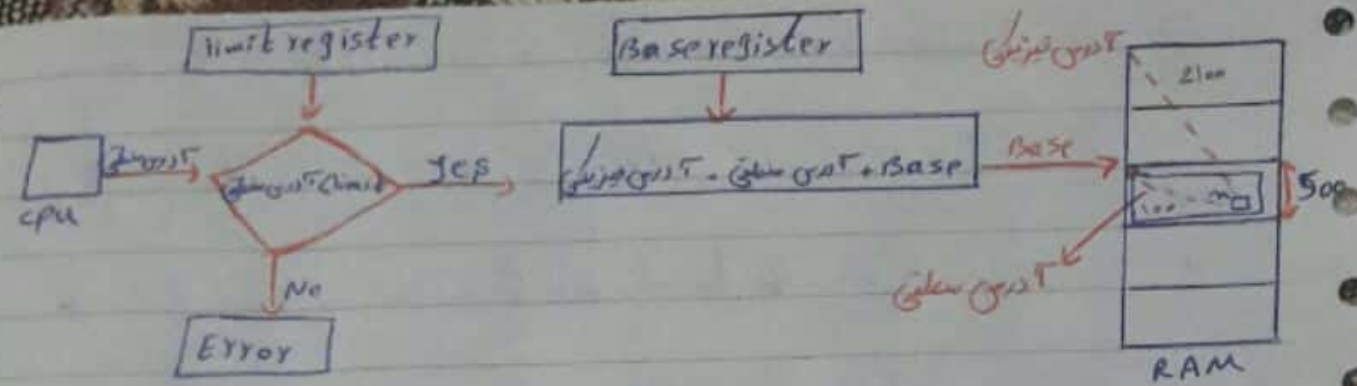
۱) تکنیک های مجاور (پیوسته) ← تکنیک های تک قسمتی
تکنیک های چند قسمتی

۲) تکنیک های غیر مجاور ← صفحه بندی
قطعه بندی

تکنیک های مجاور

اولین نگاشت:

Scanned by CamScanner



Base آدرس برنامه‌ای که متغیر مورد نظر داخل آن است.

limit : اندازه برنامه

Base = 2000

آدرس منطقی = 100 → آدرس فیزیکی = 2000 + 100 = 2100

limit = 500

تکنیک تک تسک

چند برنامگی وجود ندارد در نتیجه بیکاری پردازنده زیاد و کارایی کم می‌شود.

تکنیک چند تسکی ← با پارتیشن بندی ایستا
با پارتیشن بندی پویا

ایستا: سیستم عامل یک تسک از RAM را برای خودش می‌دارد و مابقی را به تسک‌های با اندازه‌های متفاوت تقسیم می‌کند که به هر تسک یک پارتیشن می‌گوییم هر برنامه‌ای که در حافظه می‌خواهد قرار بگیرد در پارتیشن بزرگتر یا مساوی خود قرار می‌گیرد.

* یک پارتیشن فقط متعلق به یک برنامه است.

* هر برنامه فقط داخل یک پارتیشن قرار می‌گیرد.



a.exe = 230 بایت

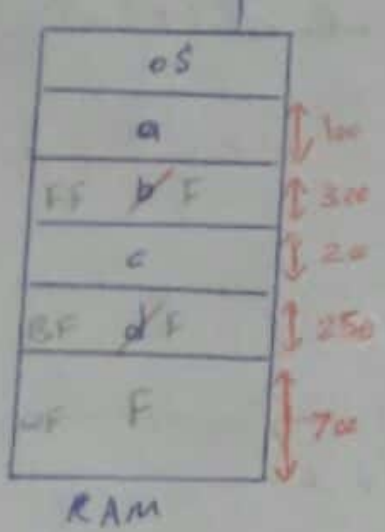
* تکه تکه شدن داخل حافظه قابل استفاده نیستند.

Scanned by CamScanner

مشکلات این روش: ۱) تکه تکه شدن داخلی حافظه ۲) تعداد برنامه‌ها محدود می‌شود به تعداد پارتیشن‌ها.

* تعداد و سایز پارتیشن‌ها قبل از ورود برنامه‌ها ساخته شده است.

پویا: تعداد و سایز پارتیشن‌ها از قبل ساخته نشده است بلکه به صورت پویا و بسته به برنامه، اندازه آن‌ها ساخته می‌شود در این روش نیاز به تعدادی الگوریتم برای یافتن محل خالی داریم.



Time
a = 100
b = 300
c = 200
d = 250
e = 180
b = خالی
d = خالی

F = 220

۱) First Fit (اولین مناسب): اولین جای خالی مناسب

۲) Best Fit (بهترین مناسب): کوچکترین جای خالی

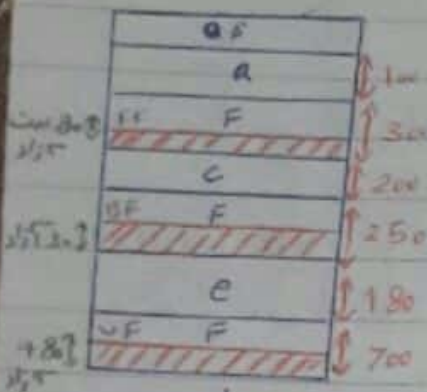
۳) worst Fit (بدترین مناسب): بزرگترین جای خالی

۴) Next Fit (مناسب بعدی): شبیه به FirstFit است با این تفاوت که برای اولین درخواست از اول حافظه شروع می‌کند به پیدا کردن مکان مناسب اما برای درخواست‌های بعدی از جایی شروع می‌کند که دفعه قبل متوقف شده است.

اگر فرآیندی وارد شود که RAM فضای آزاد برای آن نداشته باشد ولی مجموع حافظه‌های آزاد شده برای پذیرش آن آزاد باشد فرآیندها به یک سمت shift داده

Scanned by CamScanner

می شوند و ماخلی آزاد در یک ست باقی می ماند به این روش خارجی گفته می شود.



این روش زمان طلف می گذد. (از تکه تکه شدن نادیده نادیده استفاده است.)

تکنیک نیرهم جوار

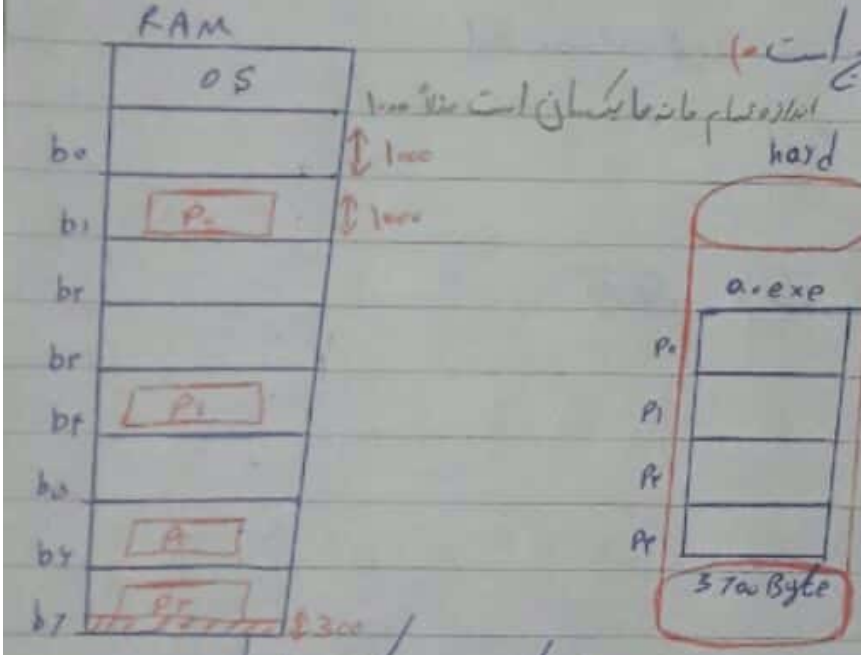
۱) صفحه بندی paging

سیستم عامل یک تست از RAM را برای خودش بری دارد و مابقی را به تست های مسلی

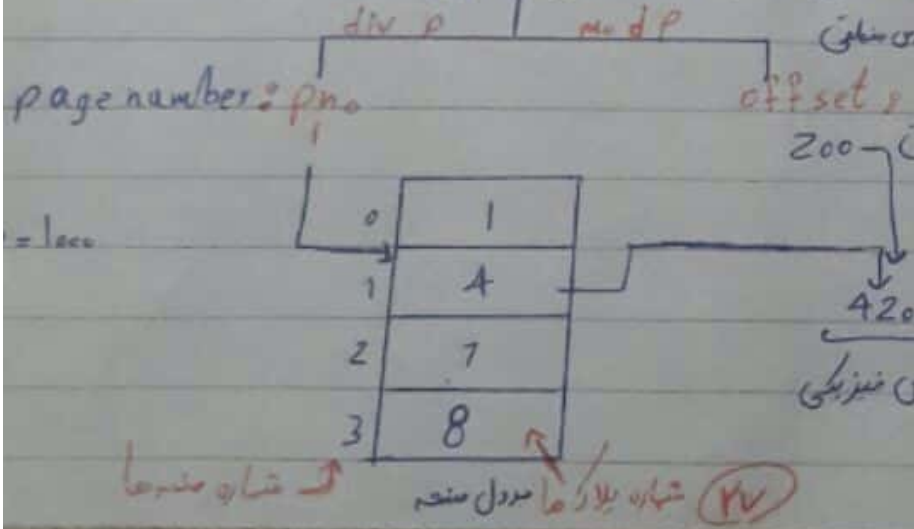
تقسیم می شود که به هر تست یک بلاک گفته می شود به بلاک تاب هم گفته می شود.

سیستم عامل تالبرا براساس اندازه بلاک مشخص می کند که به هر تالب یک پیج یا صفحه گفته

می شود. (منظور از تالب همان پیج است.)



* هر صفحه را در هر بلاکی می توان قرار داد به شرطی که آن بلاک آزاد باشد.



۲۷

معایب سنجش بلوی: ۱) اتلاف حافظه ۲) تکه تکه شدن داخلی
۲) حجم جدول صفحه

۳) کاهش سرعت کار با حافظه

اتلاف حافظه

- تکه تکه شدن داخلی: به طور متوسط به ازای هر برنامه: $P/2$

- حجم جدول صفحه: اندازه هر خانه: e

تعداد خانه های جدول: $S = P_0 + P_1 + \dots$ اندازه تمام صفحات S/P

حجم جدول صفحه: $\frac{S}{P} * e$

اتلاف در حالت کلی: $(\frac{S}{P} * e) + \frac{P}{2}$

$$f(P) = \frac{P}{2} + (\frac{S}{P} * e)$$

$f'(P) = 0 \longrightarrow$ با مشتق گیری $P = \sqrt{2Se}$

Scanned by CamScanner

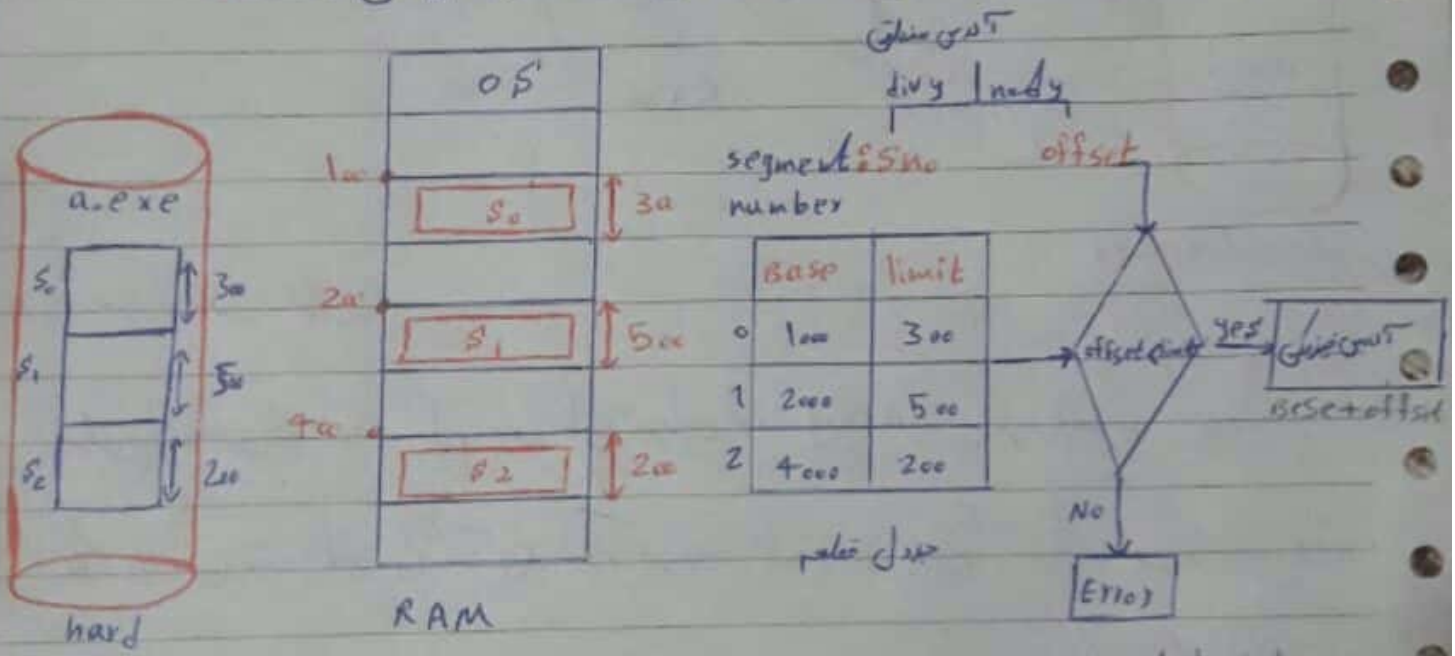
«به نام خالق عشق پیشه»

جلسه ششم

تقسیم بندی

برنامه به اندازه‌های متفاوت تقسیم شده که به هر قسمت یک segment یا قطعه می‌گویند. سیستم عامل برای هر segment یک پارتیشن ساخته و segment را داخل آن copy می‌کند.

offset: آدرس منطقی نسبت به قطعه‌ای که متعلق داخل آن است.



جدول قطعه

EX: اندازه اولین قطعه

مانند مجازی Virtual memory

حافظه‌ی مجازی استفاده از hard در کنار حافظه‌ی اصلی است به این منظور که زمانی که یک برنامه‌ی خواهد اجرا شود قسمتی که لازم است وارد RAM شود و آن قسمت که فعلاً لازم نیست وارد حافظه‌ی مجازی شود تا در صورت نیاز به RAM انتقال یابد.

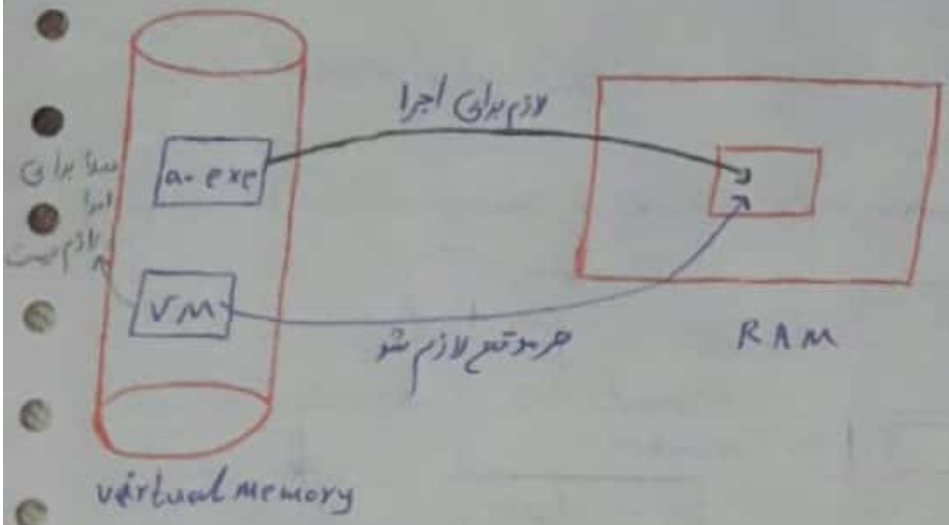
Scanned by CamScanner

مزایای حافظه‌ی مجازی

۱) عدم محدودیت سایز برنامه به اندازه‌ی حافظه‌ی اصلی

۲) امکان اجرای برنامه‌های بیشتر

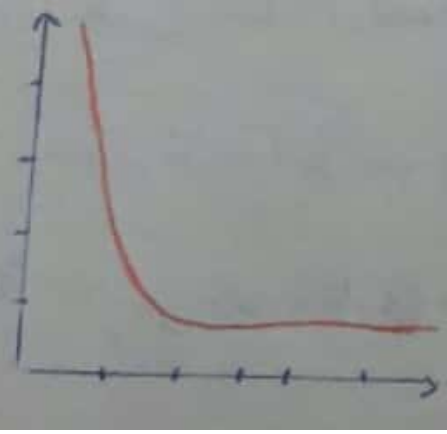
۳) جلوگیری از عمل‌های io بیهوده



صفحه‌بندی حافظه‌ی مجازی

برای جلوگیری از ورود تمام صفحه‌های برنامه به داخل RAM به هر برنامه یک محدوده معینی بلاک داده می‌شود تا زمانی که آن تعداد پر نشده باشد هر پیجی که برنامه درخواست دهد سیستم عامل برای آن می‌آورد آن پیجی را ولی آن تعداد بلاک را که پر کرد از آن به بعد هر صفحه‌ای که بخواهد وارد RAM شود یکی از صفحات قبلی باید پاک شود.

هر وقت برنامه صفحه‌ای از صفحات خود را بخواهد که در RAM نباشد به آن پیج فالت یا نقص صفحه می‌گویند.



Scanned by CamScanner

الگوریتم های جایگزینی صفحه

۱) FIFO (First In First Out)

صفحه ای که زودتر وارد شده زودتر حذف می شود و مشکل دارد: ۱) کوردکورانه عمل می کند

۲) ناسازگاری بلیدی دارد: گاهی اوقات از زیاد کردن تعداد فریم صفحه نیست نمی کند یعنی وقتی بلاک بیشتری به آن میدهم تعداد صفحه ای آن بیشتر می شود.

۲) Best option یا optimal :

صفحه ای را خارج می کند که تا اولین مراجعه به آن در آینده زمان بیشتری مانده باشد و همین باید از آینده اطلاع داشته باشد قابل پیاده سازی نیست. ناسازگاری بلیدی ندارد. کوردکورانه هم عمل نمی کند.

۳) LRU (last recently used)

صفحه ای را خارج می کند که از آخرین مراجعه به آن در گذشته زمان بیشتری گذشته باشد و مشکل ناسازگاری بلیدی و کوردکورانه را هم ندارد.

۴) LFU (last Frequently used)

سیستم عامل برای هر صفحه ای یک شمارنده یا counter در نظر می گیرد که در مرحله اول همه ی آن ها صفر است در هر بار مراجعه counter افزایش می یابد صفحاتی که counter آن ها صفر است یعنی در RAM نیستند و از بین آن هایی که در RAM هستند

Scanned by CamScanner

صفحه‌ای که counter آن کوچکترین است خارج می‌شود و باعث بوجود آوردن کوچکترین counter

زمان تلف می‌شود.

(Most Frequnlly Used) MFU (۵

صفحه‌ای که counter آن از همه بیشتر است خارج می‌شود.

(Not Frequnlly Used NFU (۶ counter ∈ ℕ

یک بیت یا حد آستانه در نظر گرفته می‌شود و نیازی به جست و جوی کل جدول نیست. اولین صفحه‌ای که counter آن کوچکتر مساوی به باشد خارج می‌شود.

(Not Recently used) NRU (۷

	pno	bno	⑦ counter	R (Reference برای)	M (Modify تغییر)
1) R=0 , M=0	0		14	0	0
2) R=0 , M=1	1		0	0	0
3) R=1 , M=0	2		0	0	0
4) R=1 , M=1					

سیستم عامل زمان را به تعدادی مساوی تقسیم می‌کند. در این الگوریتم از صفر دو بیت M و R استفاده می‌شود. هنگامی که پردازشی شروع می‌شود سیستم عامل این دو بیت را برای تمام صفحات آن پردازش صفر می‌کند. بعد از آن بیت R هر چند وقت یکبار صفر می‌شود زمانی که تعدادی صفحه رخ می‌دهد سیستم عامل براساس مقادیر M و R

Scanned by CamScanner

یکی از صفحات را برای خروج انتخاب می کند.

R=0, M=0 : صفحاتی که اخیراً استفاده نشده اند و تغییر هم نکرده اند این حالت بهترین انتخاب است.

R=0, M=1 : صفحاتی که اخیراً استفاده نشده اند ولی تغییر کرده اند این صفحات قبل از جایگزینی لازم است که بر روی hard نوشته شوند.

R=1, M=0 : صفحاتی که اخیراً استفاده شده اند ولی تغییری نکرده اند به احتمال زیاد به این صفحات دوباره رجوع خواهد شد.

R=1, M=1 : صفحاتی که به آن ها مراجعه شده و تغییر هم کرده اند که بدترین حالت می باشد.

حالت دوم چه مواردی اتفاق می افتد؟ این حالت زمانی رخ می دهد که وقفه تایمر رخ بدهد با رخ دادن این وقفه بیت R صفر می شود. وقفه تایمر تأثیری روی بیت M ندارد.

۸) الگوریتم شانس دوباره:

این الگوریتم با یک تغییر کوچک بر روی الگوریتم Fifo first in first out صورت می گیرد صفحاتی که زیاد استفاده شده اند از حافظه خارج نشوند در این روش از اول صف شروع کرده و بیت R اولین صفحه بررسی می شود اگر این بیت صفر باشد یعنی صفحه هم قدیمی بوده و هم

Scanned by CamScanner

استفاده‌ای از آن نشده است بنابراین این سین یا سینی جزء یا بگزین می‌شود

ولی اگر بیت ۴ آن ۱ باشد آن بیت صفر شده، سین به انتهای لیست اتصال یافته

و زمان ورودش به زمان فعلی تغییری کند به این ترتیب به آن سین نانس دوباره

برای باقی ماندن داده شده است و عین روند ادامه می‌یابد.

Scanned by CamScanner

« به تدکل نام استیت »

جلسه هشتم

مثال:

First fit

20	8	16	8	فضاهای درخواست
20	8	(4)	8	12
(4)	8	(4)	8	16
(4)	8	4	(4)	4
4	(2)	4	4	6
4	2	4	(2)	2
4	2	(2)	2	4

Best fit

20	8	16	8	فضاهای درخواست
20	8	(4)	8	12
(4)	8	4	8	16
4	8	0	8	4
4	8	0	(2)	6
4	8	0	(2)	2
(4)	8	(0)	(0)	4

worst fit

20	8	16	8	فضاهای درخواست
(8)	8	16	8	12
8	8	(8)	8	16
8	8	(8)	(4)	4
8	(2)	8	4	6
(6)	2	8	4	2
(2)	2	8	4	4

Scanned by CamScanner

Next fit

20	8	16	8	حفره‌ها / فرایندها
20	8	④	8	12
①	8	†	8	16
②	8	†	8	†
۰	8	†	②	6
۰	8	†	۰	2
۰	8	۰	۰	†

FIFO

مثال: الگوریتم‌های جایگزینی صفحه ۳ بلاک

shift

	1	1	2	3	2	†	2	†	6	3	1	2	3	1	6	3	5
→	1	1	1	1	1	1	1	*	2	*	*	*	6	6	6	*	1
→			2	2	2	2	2	2	3	3	†	6	1	1	1	1	2
→				3	3	3	3	3	†	†	6	1	2	2	2	2	3
→						†	†	†	6	6	1	2	3	3	3	3	5

صفحه‌هایی که جهت را عوض می‌کنند: ۱۷ / ۹ ← PF = ۹ / ۱۷ × ۱۰۰ ← PF = ۹ / تعداد کل صفحه‌ها

PF = ۰/۶۱

تعداد تغییر جهت / تعداد کل

که عدد را منفی می‌گذاریم.

هر PF یک PR نیست ولی عکس آن همیشه صادق است.

PR = PF - تعداد بلاک‌ها

PR = ۹ - ۴ = ۵ ← PR = ۵ (تعداد بلاک‌ها)

%PR = (PF - تعداد بلاک‌ها) / تعداد کل صفحه‌ها × ۱۰۰

Scanned by CamScanner

مثال:

Best option بلاک 4

چون به صفحه ی 5 نیاز داریم صفحه ی 6 را حذف می کنیم.

بین 4 تا از این چیست 4 حذف می شود

	1	1	2	3	2	1	2	1	6	3	1	2	3	1	6	3	5
→	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	*	5
→			2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
→				3	3	3	3	3	3	3	3	3	3	3	3	3	3
→							*	6	6	6	6	6	6	6	6	6	6
→																	
	+	-	+	+	+	-	+	+	-	+	+	+	+	+	+	+	-

صفحه ی جدید جایگزین صفحه ی حذف شده می شود PR = 6 - 1 - 2 PI = 6

(نیازی به shift نیست)

PF = $\frac{6}{17}$ ٪ PR = $\frac{2}{17}$ ٪

مثال:

Best option بلاک 4

	3	2	1	0	1	4	5	2	6	1	0	3	2	1	5	
→	3	3	3	3	*	*	5	*	6	6	*	3	3	*	5	
→		2	2	2	2	2	2	2	2	2	2	2	2	2	2	
→			1	1	1	1	1	1	1	1	1	1	1	1	1	
→				0	0	0	0	0	0	0	0	0	0	0	0	
	-	-	-	-	+	-	-	+	-	+	+	-	+	+	-	

PI = 9 PR = 9 - 4 - 5

SLRU مانند FIFO، نیازبه shift داریم. صفحه ی جدیدی که در ستون قبلی وجود دارد باید به انتها منتقل شود.

LRU بلاک 4

اولین دوتره حذف میشه

این که اول از همه توسط دستور قبلی گفتم

	3	0	1	5	1	2	4	6	0	4	7	5	4	8	6	
→	3	3	3	3	*	*	*	*	2	*	6	0	0	7	5	
→		0	0	0	0	5	1	2	4	6	0	4	7	5	4	
→			1	1	5	1	2	4	6	0	4	7	5	4	8	
→				5	1	2	4	6	0	4	7	5	4	8	6	
→																
					-	-	-	-	-	+	-	-	+	-	-	

Scanned by CamScanner

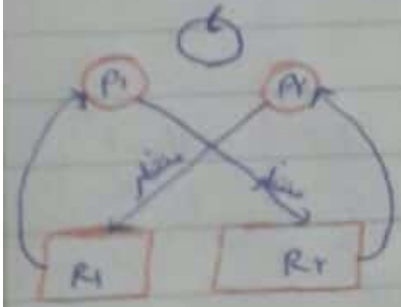
✓	Pid	counter
✓	1	0 1 2 0
✓	2	0 1 2 0
✓	3	0 1 2 0
×	4	0 1 2 0
✓	5	0
×	6	0 1 2 0

«بسم الله الرحمان الرحیم»

جلسه‌ی هشتم

بن بست Deadlock

بن بست حداقل بین دو فرآیند اتفاق می افتد که هر فرآیند نیازبه منبعی داشته باشد که در اختیار منبع دیگری از این مجموعه است.



شرایط ۴گانه کافمن:

۱. انحصار متقابل

۲. نگهداری و انتظار (Hold & wait)

۳. منابع بدون تخلیه پیش هنگام

۴. انتظار چرخشی

Scanned by CamScanner

انسداد متقابل

یکی از شرایط خطرناک این است که منابع قابل استفاده اشتراکی نباشند

نگهداری و انتظار

این که فرآیندی منبعی را داشته باشد (Hold) و برای منبع دیگر درخواست دهد می‌تواند منجر به بن بست شود چون ممکن است منبعی که می‌خواهد آزاد نباشد

منابع بدون تخلیه پیش هنگام

هر وقت فرآیندی منبعی را در اختیار دارد نمی‌شود به زور آن منبع را از اختیارش گرفت و تا هر وقت بخواهد می‌تواند آن منبع را نگه داری کند.

انتظار چرخشی

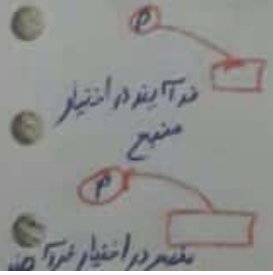
زنجیره‌ای از فرآیندها که هر فرآیند منتظر فرآیند دیگر است.

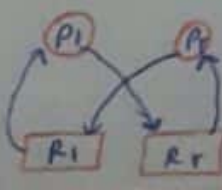
مدیریت بن بست

۱) روش شترمرغی: چشم پوشی کردن یا نادیده گرفتن بن بست - سیستم عامل وقوع بن بست را قبول ندارد و مدیریت آن را به کاربر واگذاری کند

۲) روش کشف و ترمیم: سیستم عامل اجازه‌ی دهد بن بست رخ دهد سپس آن را کشف و ترمیم می‌کند.

روش کشف: گراف تخصیص منابع



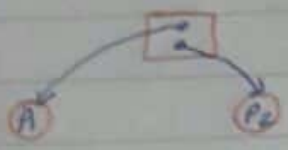


Scanned by CamScanner

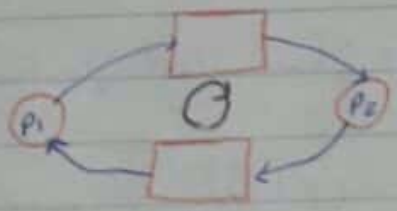
منابع

۱) تک نمونه‌ای: منبع در اختیار تک فرآیند است ولی بقیه فرآیندها می‌توانند آن را درخواست دهند.

۲) چند نمونه‌ای: منبع می‌تواند در اختیار چندین فرآیند باشد.



گفت بن بست در منابع تک نمونه‌ای:



در صورت وجود حلقه حتماً بن بست داریم



بن بست رخ داده است

مالان بن بست: P_1 و P_2

مالان بن بست: فرآیندهای داخل حلقه

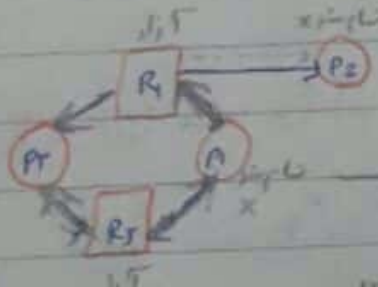


بن بست داریم ⇐ مالان بن بست: P_1 و P_2 و P_3

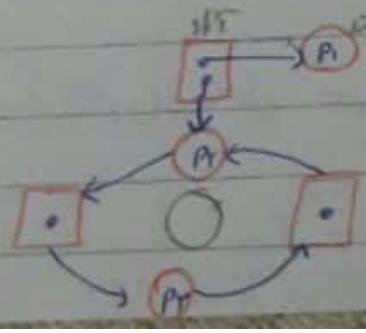
گفت بن بست در منابع چند نمونه‌ای

۱) وجود حلقه شرط لازم برای وقوع بن بست است اما غیرکافی

۲) در صورتی که هیچ فرآیندها اجرا شوند بن بست رخ نمی‌دهد ولی اگر حتی یکی از فرآیندها اجرا نشود بن بست رخ خواهد داد.



مرسوم فرآیند اجرا شده پس بن بست نداریم



مثال: P_2 و P_3 اجرا نمی‌شوند پس بن بست داریم.

Scanned by CamScanner

ترسیم بن بست

۱. انتخاب خدایی

۲. برخورد با خدایی ← گرفتن خدا آید
بازپس گیری منابع

انتخاب خدایی

معیارهای انتخاب خدایی: ۱. خدا آیندی که منابع بیشتری در اختیار دارد ۲. خدا آیندی که زمان کمتری از آغاز آن گذشته است ۳. خدا آیندی که با مت بروز بن بست در سیستم شده است.

برخورد با خدایی

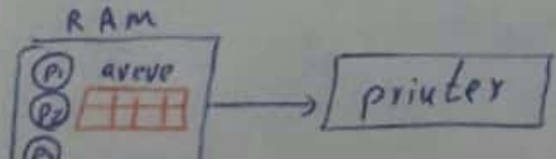
گرفتن خدا آید: یعنی خدا آید را پیدا کنیم تا منابعی که در اختیار داشته آزاد شود. دو مشکل دارد: ۱. عدم رفتن زحمات سیستم عامل ۲. شاکی شدن کاربر.

۲. بازپس گیری منابع: فقط منابع خدایی را می گیرد و به خدا آیندهای دیگری دهد و وقتی که خدا آیندها اجرا شدند همان منابع را دوباره به خود خدا آید پری گرداند.

روش های جلوگیری از بن بست در سیستم

یکی از شرایط 4 گانه کافی را نقض می کنیم.

۱. نقض انحصاری متقابل ⇐ تکنیک spooling (با نرم افزار ـ ست)



Scanned by CamScanner

۲) تضمین شرط نگهداری و انتظار ⇐ - تضمین یکتا (یکی)

- رهاسازی و درخواست مجدد

تضمین یکتا

فرآیند هنگام ورود درخواست ملی خود را مطرح می کند سپس اگر همه منابعی که می خواهد آزاد باشند به آن داده می شود و فرآیند اجرا خواهد شد و دیگر من درخواست منابع جدید ندارد اگر تعدادی از منابعی که می خواهد آزاد نباشد صبر یک به آن داده نمی شود و باید منتظر بماند.

رهاسازی و درخواست مجدد

فرآیند زمانی که می خواهد برای منبع جدیدی درخواست دهد ابتدا هرچه دارد را به سیستم عامل امانت می دهد و بعد درخواست می دهد. اگر درخواست جدیدش آزاد باشد این درخواست جدید به آن داده می شود و منابع قبلی هم به آن برگردانده خواهد شد در غیر این صورت منابع قبلی برگردانده نمی شود و باید منتظر بماند.

تضمین شرط منابع بدون تخلیه پیش هنگام

فرآیند هنگام ورود نیازمندی را اعلام می کند ۲ حالت اتفاق می افتد ۱) اگر منابع مورد نیازش آزاد باشند به آن داده می شود و اجرا می شود ۲) اگر آزاد نباشند ولی دست فرآیندهای مسدود باشند از فرآیندهای مسدود گرفته می شود و به فرآیند تازه وارد داده می شود. ۳) منابع آزاد نیستند و دست فرآیند مسدود هم نیستند در

Scanned by CamScanner

نتیجه خود نرم‌آ نیز مسدود می شود.

روش های اجتناب از بن بست

با تخصیص هوشمندانه منابع سعی می کنیم سیستم در حالت امن باشد.

حالت امن: حالتی که بن بست نداشته باشیم و احتمال وقوع بن بست هم وجود نداشته باشد.

۱) عدم شروع فرآیندی که شروع آن ممکن است باعث ورود به بن بست شود.

۲) عدم پاسخ مثبت به درخواستی که پاسخ مثبت به آن ممکن است باعث بروز بن بست شود. (معروف به الگوریتم بانکداران)

تعداد منابع	5	3	4	
	R_1	R_2	R_3	
P_1 ✓	2	1	1	
P_2 ✓	2	1	1	$1<3$ ✓ $3\leq 3$ ✓
P_3 ✓	1	1	2	$2\leq 3$ ✓ $4\leq 3$ ✗
P_4 ✗	1	1	0	

TIME

Scanned by CamScanner

جلسه نهم

سیستمی با ماتریس زیر مفروض است:

الف) ماتریس باقی مانده را به دست آورید.

ب) آیا سیستم امن است یا ناامن؟ چرا؟

ج) اگر فرآیند P_3 درخواست یک واحد از R_2 را بدهد آیا موافقت می‌شود یا خیر؟

	R_1	R_2	R_3	R_4
P_1	0	0	1	2
P_2	2	7	5	0
P_3	6	6	5	0
P_4	4	3	5	6
P_5	0	6	5	2

* ماکزیمم تقاضا (max)

	R_1	R_2	R_3	R_4
P_1	0	0	1	2
P_2	2	0	0	0
P_3	0	1	3	4
P_4	2	3	5	4
P_5	0	3	3	2

* تخصیص جاری (Allocation)

** max = Alloc + need ⇒ need = max − Alloc

max = تخصیص داده شده + تقاضای باقی‌مانده

(منابع موجود) available: R_1 R_2 R_3 R_4 = 2 1 0 0

ماتریس باقی‌مانده need:

	R_1	R_2	R_3	R_4
P_1	0	0	0	0
P_2	0	7	5	0
P_3	6	6	5	2
P_4	2	0	0	2
P_5	0	3	2	0

ب) اگر به ترتیب‌های امن سیستم امن است

توی ماتریس باقی‌مانده نگاه می‌کنیم که

فقط P_1 می‌شود.

$(2100) \xrightarrow[+(0012)]{P_1} (2112) \xrightarrow[(0332)]{P_5} (4766) \xrightarrow[(2000)]{P_2} (4798)$

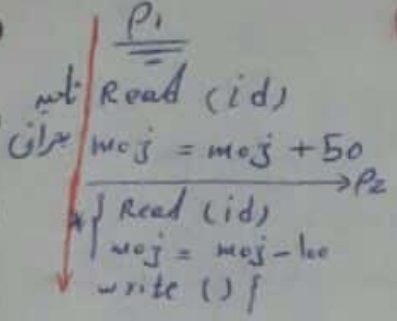
امن است $(6798) \xrightarrow[(0054)]{P_3} (6712\,12)$

ج) $(2000) \xrightarrow[(0012)]{P_1} (2012) \xrightarrow[(2354)]{P_4} (4366) \xrightarrow[(0332)]{P_5} (4698) \rightarrow$

موافقت نمی‌شود.

Scanned by CamScanner

مدیریت ناحیه بحرانی (Critical Region) و مدیریت همزمانی برداری‌ها



به دستوراتی که به داده‌های مشترک دسترسی دارند ناحیه بحرانی می گویند ناحیه‌ای است که در آن نرم‌افزارها داده‌های مشترک را دستکاری می کنند.

هرده روی داده‌های مشترک کار می کنند و زمانی که کار یکی تمام نشده دیگری کار خود را شروع می کند در نتیجه تأثیرات یکی گم خواهد شد.

راه حل های مدیریت ناحیه بحرانی

- راه حل های نرم افزاری
- راه حل های سخت افزاری
- راه حل های ارائه شده توسط سیستم عامل
- راه حل های زبان های برنامه نویسی

راه حل های ارائه شده برای مدیریت ناحیه بحرانی باید 3 ویژگی داشته باشند:

1. انحصار متقابل: در هر زمان به هیچی عنوان نباید چند فرآیند در آن ناحیه اجرایی در ناحیه بحرانی باشند.

Scanned by CamScanner

۲) انتظار محدود: هر فرآیند که متقاضی ورود به ناحیه بحرانی است باید مقدار یک زمان معین مشخص شده وارد شود.

۳) شرط پیشرفت: فرآیندی که در حال اجرای ناحیه‌ی باقی مانده خود است و قابلیت ورود به ناحیه بحرانی ندارد نباید در تعیین فرآیند بعدی متقاضی تأثیر گذار باشد.

* ۱) استفاده از متغیر مشترک ← Turn (نوبت)
۲) استفاده از متغیر مشترک ← flag (پرچم - علامت)
۳) الگوریتم پترسون

→ حل مسئله‌ی عدم انحصاری

int turn ☐

(۱)

ناحیه اول	ناحیه باقی مانده
P1	P2
while (turn != 1) do	while (turn != 2) do
ناحیه بحرانی	ناحیه بحرانی
turn = 1;	turn = 2;
ناحیه باقی مانده	ناحیه باقی مانده

هر فرآیند قبل از ورود به ناحیه بحرانی Turn را بررسی می کند اگر نوبت به ترتیب نباشد آن قدر می چرخد تا نوبتش زمانی اش تمام شود در غیر این صورت نوبت متعلق به خودش بوده وارد ناحیه بحرانی می شود و در نهایت نوبت را به ترتیب می دهد (انحصاری متقابل دارد چون متغیر Turn در هر لحظه فقط یک مقدار دارد و بیش از یک فرآیند اجازه ورود پیدا نمی کند. انتظار محدود دارد چون نوبت دقیقاً یکی در میان عوض می شود. شرط پیشرفت دارد چون اگر یک فرآیند نخواهد وارد ناحیه بحرانی شود نوبت تغییر نمی کند و

Scanned by CamScanner

نتیجه نداریم پدر رقیب هم نمی تواند وارد شود.

int turn ☐ turn = 2 bool flag ☐☐

turn = 1

P1

flag[1] = true;

while (flag[2] == true) do;

ناحیه بحرانی

flag[1] = False;

ناحیه باقی مانده

P2

ناحیه اول

flag[2] = true;

while (flag[1] == true) do;

ناحیه بحرانی

flag[2] = false;

ناحیه باقی مانده

هر ناحیه نداریم متقابل خودرا با True کردن flag شان می دهد. پس اگر flag رقیب نیز true باشد در یک حلقه منتظری ماند در غیراین صورت وارد ناحیه بحرانی شده و هنگام خروجی flag خودرا false می کند. انتظار متقابل دارد چون اگر نداریم دوی وارد ناحیه بحرانی شود flag خودرا true کرده و هم flag رقیبش false نموده است. بنابراین رقیب در ناحیه بحرانی بوده است و چون flag خود نداریم true شده رقیب نمی تواند وارد شود شرط پیشرفت دارد چون اگر نداریم دوی متقابل ورود نباشد flag آن را false می کند و مانع رفتن دیگری نمی شود. انتظار متقید ندارد چون اگر نداریم دوی flag خودرا true کند، موفق به چک کردن flag رقیب نشود. حال اگر رقیب نیز flag خودرا true کند دراین صورت رقیب می تواند وارد شود و نه نداریم و بن بست دگر سنگی ایجاد می شود.

Scanned by CamScanner

((بسمه تعالی))

جلسه دهم

الگوریتم پترسون

int turn, bool flag

P1

flag[1] = True;

turn = 2;

while (flag[2] = true && turn = 2)do;

برای وارد شدن

ناحیه بحرانی

Flag[1] false

P2

Flag[2] = True;

Turn = 1;

while (flag[1] = True && Turn = 1)do;

ناحیه بحرانی

flag[2] = False

هر فرآیند متقاضی flag خودرا True می‌کند و نوبت را به فرآیند رقیب واگذاری کند

سپس اگر flag رقیب true باشد چون نوبت را خودبه فرآیند رقیب واگذار کرده

پس صبری کند تا رقیب وارد ناحیه بحرانی شود در غیر این صورت وارد ناحیه

بحرانی می‌شود و هنگام خروج flag خودرا false می‌کند.

انحصار متقابل دارد زیرا چون اگر هر دو فرآیند متقاضی وارد باشد Turn فقط متعلق به

یکی از آن درخواست بوده و فرآیند دیگر باید منتظر بماند.

شرط پیشرفت دارد چون اگر فرآیندی نخواهد وارد شود flag خودرا True نمی‌کند

و مانع رفتن دیگری نمی‌شود.

Scanned by CamScanner

انتظار مقید دارد چون دقیقاً نوبت یکی در میان عوض می‌شود و عیبی فرآیندی نمی‌تواند نوبت فرآیند دیگر را بگیرد.

راه حل های سخت افزاری

۱) دستور Tsl (Test and set lock)

۲) دستور معاوضه (Exchenge)

۳) از کار انداختن وقفه ها

دستور Tsl

```
Tsl (int i){
if (i==0) {
    i=1;
    Tsl = True;
}
else
Tsl = False;
}
```

int lock

P1	P2	P3
repeat	repeat	repeat
until (Tsl(lock));	until (lock);	until (Tsl(lock));
قفل باز است	قفل باز شد ابتدای حلقه repeat می‌گردد	
قفل باز شد قفل را می‌بندد		
ناحیه بحرانی	ناحیه بحرانی	ناحیه بحرانی
lock=0;	lock=0;	lock=0;

lock=0 ← قفل باز باشد | lock=1 ← قفل بسته است.

repeat: تا زمانی که شرط داخل until برقرار نیست، دستورات داخل repeat تکرار می شود.

دستور Tsl یک دستور اتومیک و تجزیه ناپذیر است و خودش قطع شدن ندارد اولین فرآیندی که موفق شود Tsl را اجرا کند مقدار lock را یکی کند (lock=1)

Scanned by CamScanner

وغیره می‌دارد ناحیه بحرانی می‌شود ولی نخ آدم‌های بعدی نمی‌توانند وارد شوند چون قفل را بسته می‌بینند.

انحصار متقابل دارد چون فقط در هر لحظه فقط یک نخ آدم‌ها می‌تواند Tas را اجرا کند یکی از نخ آدم‌ها مجوز ورود می‌گیرد و بقیه موفق به ورود نمی‌شوند.

شرط پیشرفت دارد چون نخ آدم‌هایی که نمی‌خواهند وارد شود کاری با lock ندارد و به آن‌هایی که می‌خواهند وارد شوند اجازه می‌دهد lock را دستکاری کنند.

شرط انتظار محدود ندارد چون همین مکانیزم نوبت دهی وجود ندارد.

دستور معاوضه یا Exchang

exchang (int a, int b)

int lock

key 1 | key 2 | key 3

int temp

P1:
key1 = 1; lock = 0
repeat
Ex(key1, lock);
until (key1 == 0);
lock = 0;

P2:
key2 = 1;
repeat
Ex(key2, lock);
until (key2 == 0);
lock = 0;

P3:
key3 = 1;
repeat
Ex(key3, lock);
until (key3 == 0);
lock = 0;

temp = m;
m = r;
r = temp

key1 یک خانه Ex(key1, lock) ۱
until (key1 == 0); است مقدار Ex را الباقی کند

نکته: نای این نخ آدم‌ها مشکل به نام انتظار مشغول دارند یعنی نخ آدم‌هایی که (Busy waiting)
می‌خواهند وارد ناحیه بحرانی شود داخل یک حلقه‌ی چرخنتا زمان CPU را هدر دهد.

Scanned by CamScanner

از کار انداختن رشته ها

هر رشته آماده قبل از ورود به ناحیه بحرانی می تواند رشته ها را از کار بیندازد و قبل از خروج دوباره آن ها را فعال کند.

معایب: با این روش cpu نمی تواند بین رشته آماده های دیگر switch کند. شاید فعال کردن مجدد رشته ها زمان بر شود. اگر چندتا cpu داشته باشیم فقط cpu جاری از کار می افتد بقیه آن ها کار می کنند.

«به نام خداوند جان و خرد»

جلسه یازدهم

راه حل ارائه شده توسط سیستم عامل

سمافور semaphore

راه حل های قبلی به سادگی قابل تأمیم به مسائل پیچیده تری باشند برای رفع این مشکل می توان از یک ابزار هزینه ای به نام سمافور استفاده کرد.

سمافور s یک متغیر صحیح (integer) ی باشد که جدا از مقدار دهی اولیه فقط از طریق دو عمل اتومیک استاندارد به نام های wait و سیگنال قابل دسترسی است

تعاریف wait و سیگنال به صورت زیری باشد.

Scanned by CamScanner

```
wait ( ){
value = value(-1)
if (value < 0){
Block process p.
Add p to Q
}
}
```

صف انتظار

```
signal ( ){
value = value(+1);
if (value ≤ 0){
remove process from Q;
Add p to Ready
wake up P;
}
}
```

الگوریتم سمافور

```
int S(1)
P1:
ناحیه اولیه
wait (s);
ناحیه بحرانی
signal (s);
ناحیه باقی مانده
```

```
P2:
wait(s);
signal(s);
```

```
P3
wait(s);
signal(s);
```

عرفرآیند قبل از ورود به ناحیه بحرانی مثل wait و بعد از اتمام اجرای ناحیه بحرانی مثل signal را انجام می دهد. وقتی فرآیندی مثل wait را اجرا کرد و سمافور را غیرمثبت می کند (value<0) فرآیند در صف انتظار (Q) قرار می گیرد و طبق آن سمافور قرار داده می شود

وحالت فرآیند به Block تغییری یابد سپس control به زمانبند cpu منتقل می گردد تا فرآیند دیگری را برای اجرا انتخاب کند. فرآیندی که Block شده برای اجرای عمل سیگنال توسط فرآیند دیگر دوباره از سر گرفته می شود. این فرآیند توسط یک عمل wakeup بیدار شده و از حالت انتظار به حالت آماده می رود

(۵۲)

و در ست Ready قرار داده می شود.

$P_1 \xrightarrow{-1} P_2 \xrightarrow{-1} P_3 \xrightarrow{-1}$ (wait) $P_1 \xrightarrow{-1} P_2 \xrightarrow{-1} P_3 \xrightarrow{-1} \cdots$ (signal)



این شرط بیزی است P2 را از خواب بیدار کرده و در صف Ready قرار داده می گیرد و P2 هم به همین ترتیب است.

یک سمافور با مقدار اولیه یک به صورت مشترک بین برنامه های متفاوتی گرفته می شود هر عملیات بند هنگام ورود wait انجام می دهد و هنگام خروج signal را.

انتظار متقابل دارد چون تنها یک نخ آنها مقدار آن را از یک منبع تغییری دهد و خود آن نخ آنها وارد ناحیه بحرانی شده ولی بقیه چون مقدار آن را منفی می کنند امکان ورود به ناحیه را ندارد.

شرط پیشرفت دارد چون نخ آنهایی که نمی خواهد وارد ناحیه بحرانی شود اصلاً wait انجام نمی دهد و تأخیری در رفتن سایر نخ آنها ندارد.

شرط انتظار محدود دارد چون دقیقاً یک تعداد نخ بیزی وجود دارد و نخ آنهایی که منتظر می شوند در صف قرار می گیرند و به ترتیب از صف برداشته شده و وارد ناحیه بحرانی می شوند.

مسائل کلاسیک همزمانی

۱. مسئله تولید کننده و مصرف کننده با بافر محدود
Producer Consumer

۲. مسئله خوانندگان و نویسندگان

۳. تولید کننده- مصرف کننده با استفاده از monitor

Scanned by CamScanner

تعدادی مسئله‌ی کلاسیک وجود دارد که برای تست هر روش جدید هم‌زمانی مورد استفاده قرار می‌گیرد به عبارت دیگر هر عبارت الگوریتم جدید در ابتدا با هم‌زمانی پردازش ها باید بتواند این مسائل کلاسیک را به درستی حل کند.

مسئله تولید کننده و مصرف کننده با بافر محدود

دو فرآیند یک بافر با اندازه ثابت n به اشتراک می گذارند یکی از آن ها (تولید کننده) اطلاعات را در بافر قرار می دهد و دیگری (مصرف کننده) اطلاعات را از بافر برمی دارد مشکل زمانی به وجود می آید که تولیدکننده می خواهد چیزی را در بافر قرار دهد اما بافر قبلاً پر شده است که در این حالت تولیدکننده باید بخوابد تا مصرف کننده یک یا چند قلم داده را از بافر بردارد و آن را بیدار کند به همین ترتیب اگر مصرف کننده بخواهد داده‌ای را از بافر بخواند ولی بافر خالی باشد باید به خواب برود تا تولیدکننده یک یا چند داده را در بافر قرار داده و آن را بیدار کند.

semaphor s(1) , Empty(N) , Full(0)

producer	consumer	empty
Emptywait();	fullwait();	wait: تعداد خانه‌های خالی زیاد
s.wait();	s.wait();	signal: تعداد خانه‌های پر کم
s.signal();	Empty.signal();	
wait: تعداد خانه‌های خالی کم	s.signal();	
signal: تعداد خانه‌های پر زیاد		

Scanned by CamScanner

- Full: تعداد خانه های پر بافر
- Empty: تعداد خانه های خالی بافر
- s برای این که همزمان دارد ناحیه بحرانی شوند. برای این که consumer و producer همزمان وارد ناحیه بحرانی نشوند.

از ۳ سمافور استفاده شده اولی به نام full برای شمارش تعداد خانه های پر سمافور Empty برای شمارش خانه های خالی و سمافور s برای آن است که مطمئن شویم که تولید کننده و مصرف کننده به بافر همزمان به بافر دسترسی ندارند در ابتدا مقدار اولیه Full مساوی صفر است، Empty مساوی n است یعنی تعداد کل خانه های بافر n مساوی یکی باشد.

- نکته: سمافورهایی که مقدار اولیه آن ها یک است اغلب برای این استفاده می شود که از ورود همزمان دو یا چند پردازش جلوگیری کند.

@jozveotamrin

Scanned by CamScanner