



دانشگاه کردستان

دانشکده مهندسی

گروه مهندسی کامپیوتر

جزوه درس:

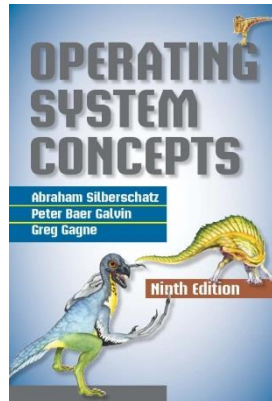
سیستم‌های عامل

تهیه کننده و مدرس:

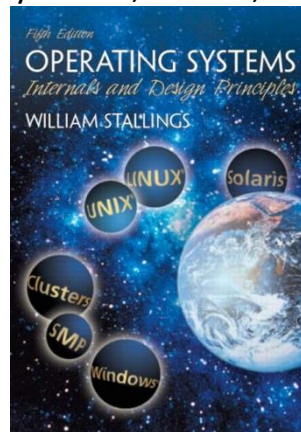
دکتر علیرضا عبدالله پوری

References:

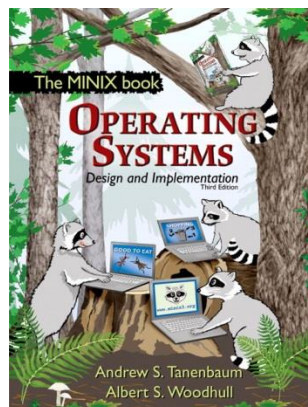
- Abraham Silberschatz, Galvin, and Gagne. “Operating System Concepts”, 9th ed., John Wiley and Sons.



- William Stallings. “Operating Systems”, 4th ed., Prentice Hall, 2001.



- Andrew S. Tanenbaum. “Operating Systems design and implementation”, 3rd ed., Prentice Hall, 2006.



Course Homepage: <http://prof.uok.ac.ir/abdollahpouri/os.html>

۱

۲

۳

۴

۵

۶

۷

۸

فصل اول

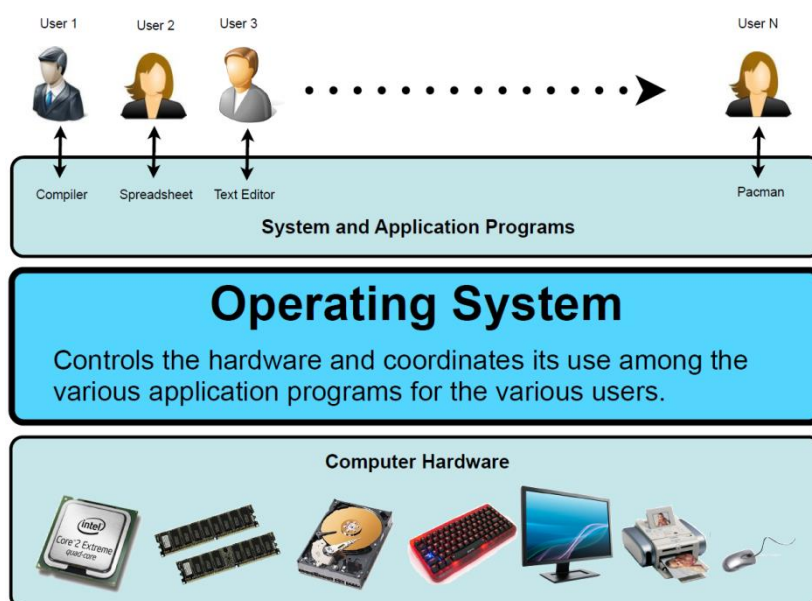
مقدمه و تاریخچه

تعریف سیستم عامل در کتاب سیلبرشاتس (Silberschatz)

سیستم عامل مهم‌ترین برنامه‌ای است که در یک سیستم کامپیوتری مدام در حال اجرا است و یک واسطه بین کاربر و سخت افزار فراهم می‌کند؛ همچنین، مدیریت تمامی منابع سیستمی سخت‌افزارها و سایر برنامه‌ها را انجام می‌دهد.

ایراد این تعریف این است که چون بحث ما بر روی سیستم تک پردازنده‌ای است، اجرای واقعی همزمان سیستم عامل در کنار برنامه‌های دیگر ممکن نیست. بلکه سیستم عامل در بازه‌های زمانی مختلف می‌آید و نقش کنترلی خود را انجام می‌دهد و در زمانی که CPU در اختیار برنامه دیگر است، سیستم عامل یا قبلاً تأثیر خود را گذاشته یا بعداً می‌گذارد. بنابراین اگر می‌گوییم سیستم عامل برنامه‌ای است که همیشه در حال اجراست از دید سطح بالا و انسان است و نه از دید CPU.

سیستم عامل استفاده از کامپیوتر را ساده می‌سازد. این بدان معناست که مثلاً کاربر یا برنامه نویس بدون درگیر شدن با مسائل سخت افزاری دیسکها به راحتی فایلی را بر روی دیسک ذخیره و یا حذف کند. این کار در واقع با به کار بردن دستورات ساده‌ای که فراخوان‌های سیستمی (System Calls) را صدا می‌زنند، انجام می‌پذیرد. در صورت عدم وجود سیستم عامل، کاربر و یا برنامه نویس می‌بایست آشنایی کاملی با سخت افزارهای مختلف کامپیوتر (مثل مونیتر، فلاپی، صفحه کلید و غیره) داشته باشد و روتین‌هایی برای خواندن و یا نوشتن آنها به زبانهای سطح پائین بنویسد. بنابراین، سیستم عامل باعث می‌شود کسی که از بالا نگاه می‌کند یک دید انتزاعی، سطح بالا و کاربرپسند نسبت به سیستم پیدا کند.



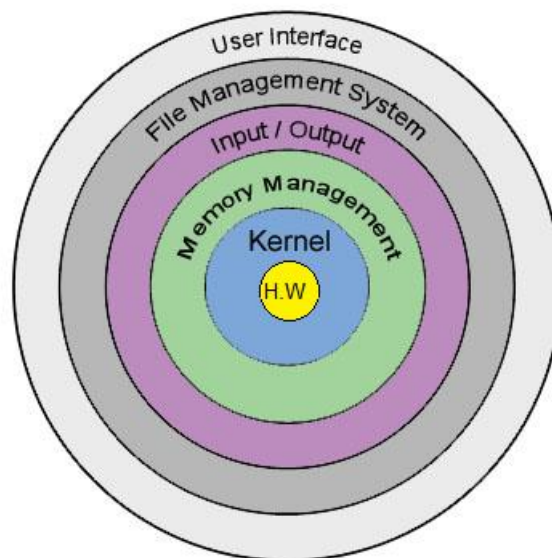
منابع } فیزیکی: CPU و دستگاههای ورودی/خروجی
منطقی: خانه های حافظه، فایلها و داده ها، و ...

مزایای سیستم عامل

- ✓ جزئیات سخت افزار را از کاربر پنهان می کند (برنامه نویسی سخت افزار بسیار مشکل است)
- ✓ سهولت استفاده از کامپیوتر
- ✓ استفاده بهینه و عادلانه از منابع
- ✓ سهولت ارتقاء و توسعه پذیری
- ✓ جلوگیری از خرابکاری کاربران/برنامه ها روی همدیگر

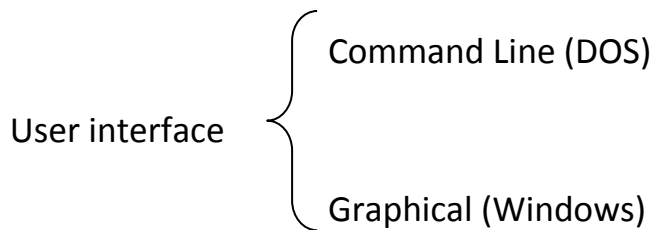
ساختار و جایگاه سیستم عامل

مدلهای متفاوتی در کتابهای مختلف برای ساختار سیستم عامل و جایگاه آن در سیستمهای کامپیوتری پیشنهاد شده است مانند: مدل هرمی، مدل لایه ای و مدل پله ای.



مدل لایه ای در سیستم عامل

در مدل لایه ای، بیرونی ترین لایه واسط کاربر و درونی ترین لایه سخت افزار می باشد. واسط کاربر می تواند به دو صورت "خط فرمان" و "گرافیکی" باشد.



```
C:\>dir
Volume in drive C is Windows
Volume Serial Number is B4F7-C128

Directory of C:\

01/10/2007  19:38    <DIR>          25c7843ff7527e04ca3749
19/09/2007  16:56    <DIR>          97a61370812ce95093e63d
19/09/2007  16:42    <DIR>          0 AUTOEXEC.BAT
19/09/2007  16:36    211 boot.ini.comodofirewa.
19/09/2007  16:42    <DIR>          0 CONFIG.SYS
19/09/2007  16:48    <DIR>          Documents and Settings
19/09/2007  16:53    <DIR>          e4a32567d68c0d02f07b4d
19/09/2007  16:54    <DIR>          Intel
01/03/2008  21:48    <DIR>          Program Files
19/09/2007  16:57    516 RHDSetup.log
25/11/2007  13:03    <DIR>          UBcd4Win
13/03/2008  11:32    <DIR>          WINDOWS
               4 File(s)      727 bytes
               8 Dir(s)    1,479,077,888 bytes free

C:\>
```

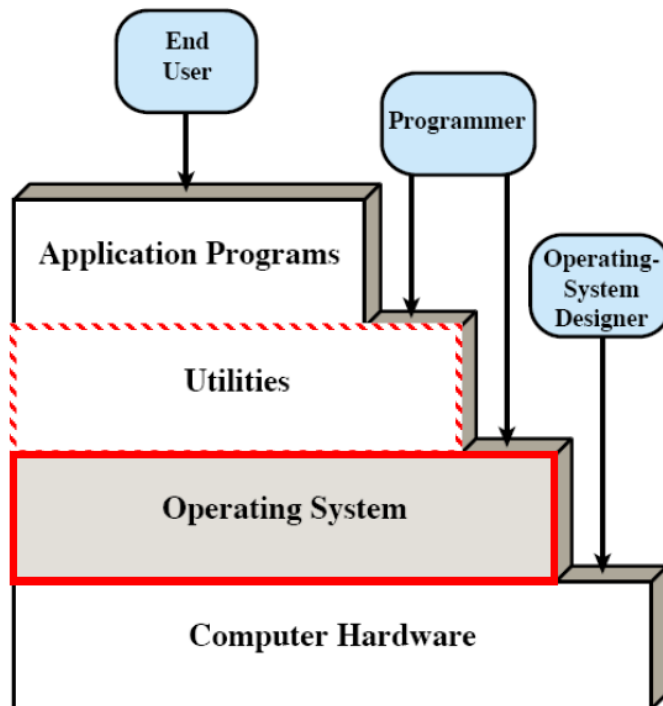
هسته سیستم عامل که به آن Kernel می‌گویند، در حافظه اصلی مقیم است و شامل توابعی است که مکرراً استفاده می‌شود. هسته به سخت افزار دسترسی مستقیم دارد. سیستم حداقل از دو حالت اجرا حمایت می‌کند.

۱- حالت ممتاز (حق بالا):

یک وضعیت اجرایی است که به تمام دستورالعمل‌های سخت افزار اجازه اجرا می‌دهد، که به این حالت سیستم، حالت ناظر یا حالت هسته (Supervisor/kernel mode) می‌گویند.

۲- حالت عادی (حق پایین):

حالتی است که اجازه اجرای دستورالعمل‌های حساس سخت افزاری مثل دستورالعمل توقف و دستورالعمل‌های ورودی/خروجی را نمی‌دهد. این حالت را حالت کاربر (user mode) نیز گویند چرا که برنامه‌های کاربران معمولاً در این حالت اجرا می‌شوند.



تقسیم بندی سیستم عامل از جهت ارتباط با کاربر

✓ **محویره‌ای (Interactive):** به سیستم‌هایی گویند که در آن کاربر به صورت مستقیم با کامپیوتر در ارتباط است، دستوراتی وارد می‌کند و منتظر پاسخ می‌ماند. پس از دریافت پاسخ، مجدداً دستوراتی را وارد می‌نماید.

✓ **دسته‌ای (Batch):** برنامه‌های مختلفی که دارای نیازهای مشابه هستند در یک دسته قرار گرفته و به صورت یک‌جا توسط کامپیوتر پس از بار شدن کامپایلر مورد نیازشان اجرا می‌شوند.



نمونه‌ای از یک سیستم دسته‌ای ساده

تقسیم بندی سیستم عامل از جهت ارتباط با دستگاههای ورودی/خروجی

✓ **مستقیم (online):** پردازنده به صورت مستقیم با دستگاه ورودی/خروجی در ارتباط است. به دلیل

کند بودن دستگاههای I/O کارایی پردازنده به صورت قابل توجهی کاهش می‌یابد.

✓ **غیر مستقیم (offline):** در این دسته پردازنده به صورت مستقیم با دستگاه ورودی و خروجی در

ارتباط نیست. بلکه ابتدا عمل خواندن ورودی توسط یک سری کارت خوان انجام می‌گیرد. داده‌ها به

حافظه جانبی یا نوار مغناطیسی انتقال می‌یابد. سپس پردازنده داده‌ها را از این حافظه جانبی یا نوار

مغناطیسی می‌خواند. در این روش امکان همپوشانی عملیات ورودی/خروجی و کار پردازنده وجود

دارد.

بافر کردن (Buffering)

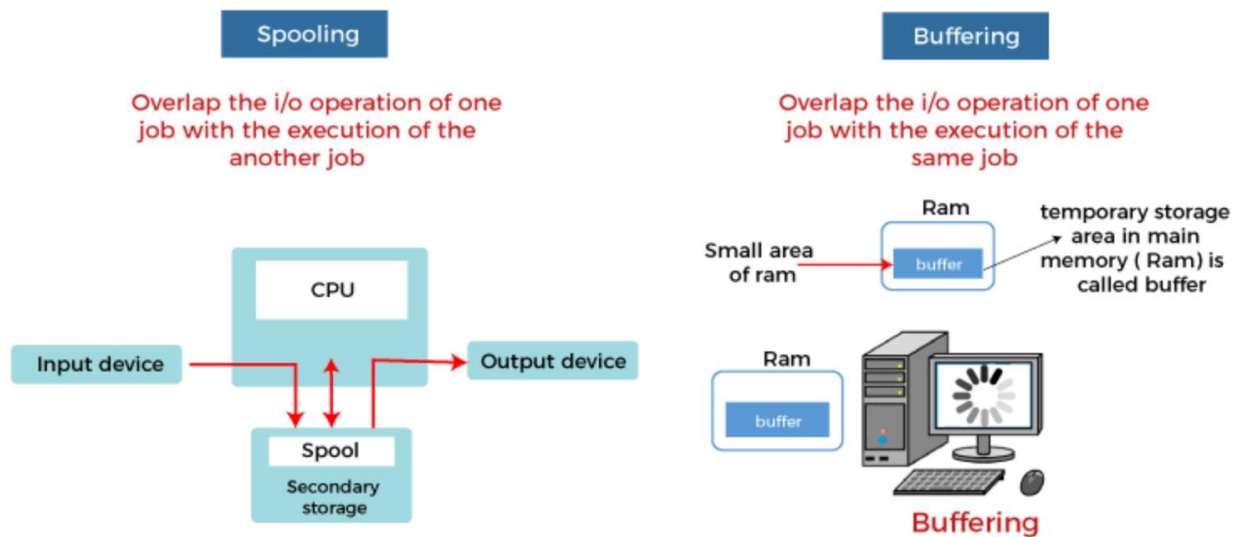
علی‌رغم استفاده از نوارهای مغناطیسی باز هم بهره‌وری سیستم به دلیل کند بودن عملیات ورودی و خروجی

کاهش می‌یابد. به این دلیل می‌توان از حافظه‌های میانی (بافرها) عملیات ورودی و خروجی یک برنامه را با اجرای

آن به صورت همزمان انجام داد. این عمل در مورد کارهایی که دارای حجم عملیات محاسباتی و ورودی/خروجی

متناسب می‌باشند، باعث افزایش کارایی می‌شود.

Spooling: در سیستم‌هایی که دارای دیسک‌های سریع و با حجم زیاد باشند می‌توان بجای انتقال داده به نوار مغناطیسی و سپس استفاده از آن، داده را روی دیسک ذخیره نمود و به این ترتیب در حین پردازش یک کار عمل انتقال داده‌ها و برنامه‌های دیگر به روی دیسک به وجود آمده و در نتیجه همزمانی اجرای چندین برنامه امکان پذیر می‌شوند.

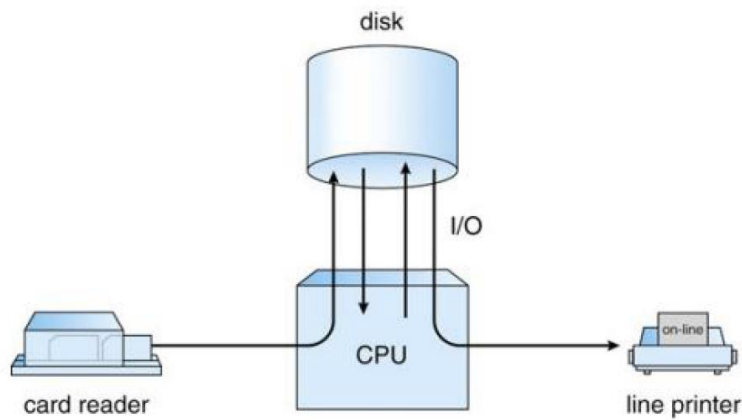


مزایای Spooling

- ۱) افزایش بهره‌وری چرا که همه دستگاه‌ها با هم کار می‌کنند.
- ۲) استفاده از راه دور ساده شد یعنی شرکت‌های کوچک می‌توانند یک کامپیوتر ارزان بخرند برنامه را بوسیله کارت‌خوان خوانده و برنامه را به صورت نوار به سرور اصلی انتقال دهند .

معایب Spooling

- ۱- ارتباط با کاربر همچنان offline است.
- ۲- زمان برگشت کار طولانی است.
- ۳- چک کردن و برطرف کردن ایرادات وقت‌گیر است.

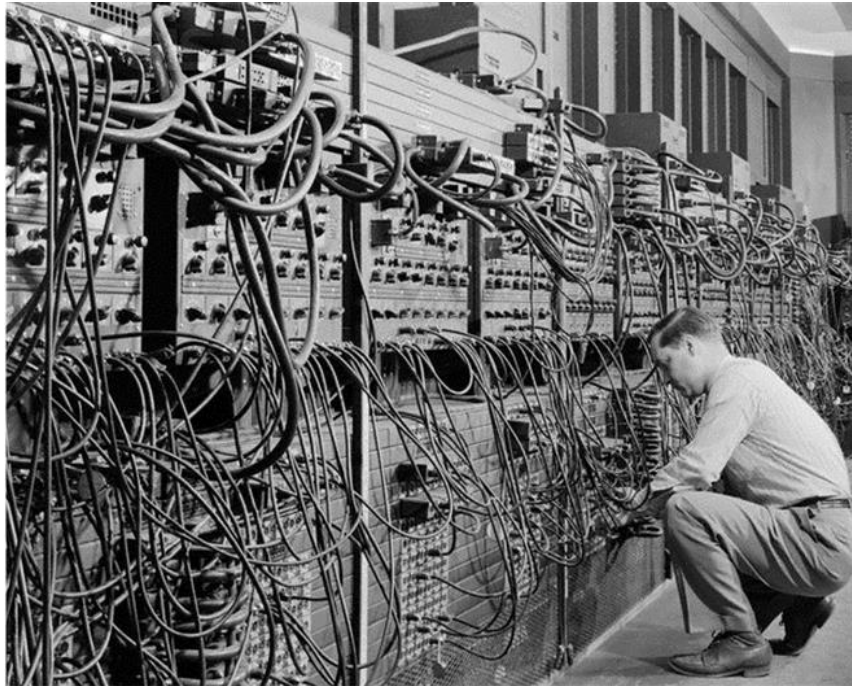


بررسی تاریخی سیر تکاملی سیستم عامل

بررسی تاریخچه سیستم‌های کامپیوتری، مشکلات پیش روی طراحان و اپراتورهای اولیه را نشان می‌دهد و ایده‌ها و راه‌حلهای کلیدی را بیان می‌کند و نشان می‌دهد که علم کامپیوتر اساساً یک علم تجربی است و ویژگی‌ها عمدتاً از نیازها تکامل یافته‌اند. سیستم‌های عامل به طور کامل وابسته به ساختار معماری کامپیوترهایی هستند که روی آنها اجرا می‌شوند.

نسل اول کامپیوترها (۱۹۴۵-۱۹۵۵)

در نسل اول کامپیوترها که از لامپ خلأ برای ساخت آنها استفاده می‌شد (مانند کامپیوتر ENIAC). زبانهای برنامه نویسی (حتی اسمبلی) ابداع نشده بودند و سیستم عامل نیز اصلاً وجود نداشت. روند کار به این صورت بود که برنامه نویسان تنها در یک فاصله زمانی مشخص حق استفاده از کامپیوتر بزرگ و گران قیمت را داشتند. آنها برنامه‌های خود را توسط تخته مدار سوراخدار و به زبان ماشین به کامپیوتر می‌دادند. برنامه نویس در واقع خودش یک اسمبلر بود یعنی برنامه را با زبان ماشین و با صفر و یک‌ها باید می‌نوشت. اطلاعات به صورت دستی و مستقیم با استفاده از سوییچ وارد می‌شوند.



ایراد سیستم عامل های نسل اول

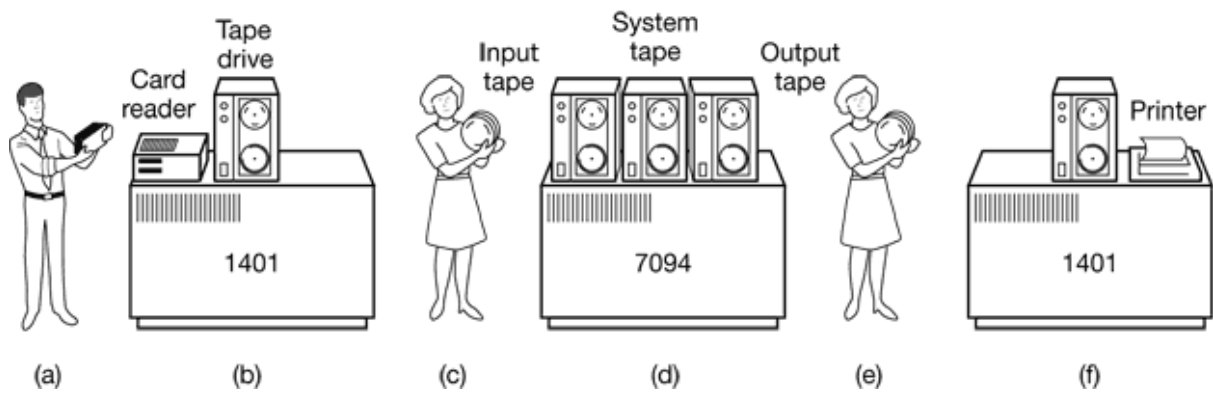
- (۱) منابع را هدر می دادند چون وقتی کار می کند خروجی بیکار است و ...
- (۲) Turnaround time آن ها بالا بود یعنی زمان پاسخ خیلی طولانی بود.
- (۳) ارتباط با کاربر offline بود.
- (۴) Debug کردن برنامه ها سخت بود.

نسل دوم کامپیوترها (۱۹۶۵-۱۹۵۵)

در این نسل سخت افزارهای جدیدی پدید آمد مثلاً ترانزیستور به جای لامپ خلاء، card reader به جای تخته مدار سوراخ دار، پرینتر به جای لامپ هایی که خاموش و روشن می شدند، tape drive به عنوان ورودی که می توانیم OS آن را لود کنیم. همچنین، کامپایلر به وجود آمد و زبان هایی مثل COBOL، Fortran، PL1.

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

در این نسل، تمایز بین برنامه نویس و اپراتور مشخص گردید.



برنامه نویس، برنامه خود را به صورت دسته‌ای از کارتهای سوراخ‌دار (job) تحویل کارت خوان می‌دهد. اپراتور دسته‌ای از jobها را به نوار مغناطیسی منتقل می‌کند (مفهوم Batch). سپس برنامه‌ها از نوار خوانده شده و پس از اجرا، خروجی به نوار دیگری منتقل می‌گردد. در نهایت، اپراتور نوار خروجی را برای چاپ بوسیله پرینتر به IBM 1401 تحویل می‌دهد.

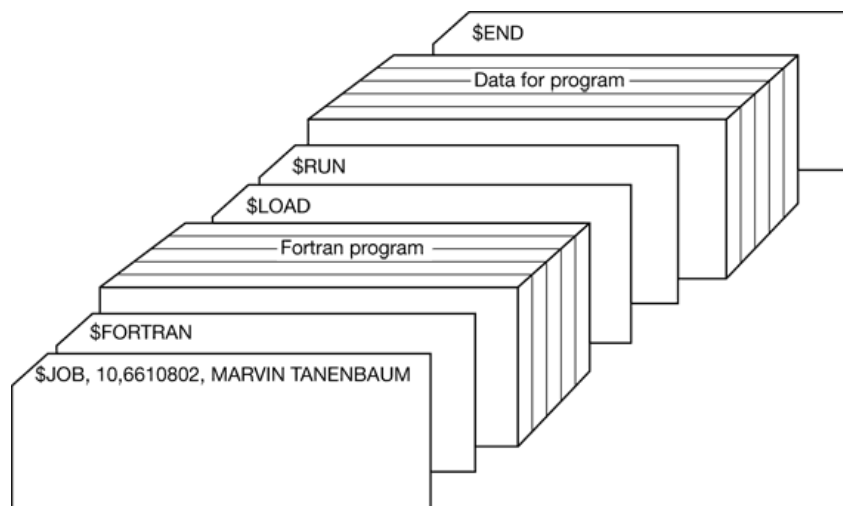
IBM 1401: برای I/O (ارزانتر و کندتر)

IBM 7094: برای پردازش (گرانتر و سریعتر)

IBM 7094



- ✓ نسل دوم اولین نسلی است که در آن سیستم عامل به وجود آمد. سیستم‌های عامل این نسل را Batch می‌گویند.
- ✓ دستگاه‌های جانبی مستقیماً به پردازنده وصل نیست و اپراتور این کار را انجام می‌دهد. برنامه‌نویس‌ها، برنامه‌های خود را در اختیار اپراتور قرار می‌دهند و سپس اپراتور برنامه‌هایی که الزامات یکسان دارند در دسته‌هایی گروه‌بندی می‌کند.

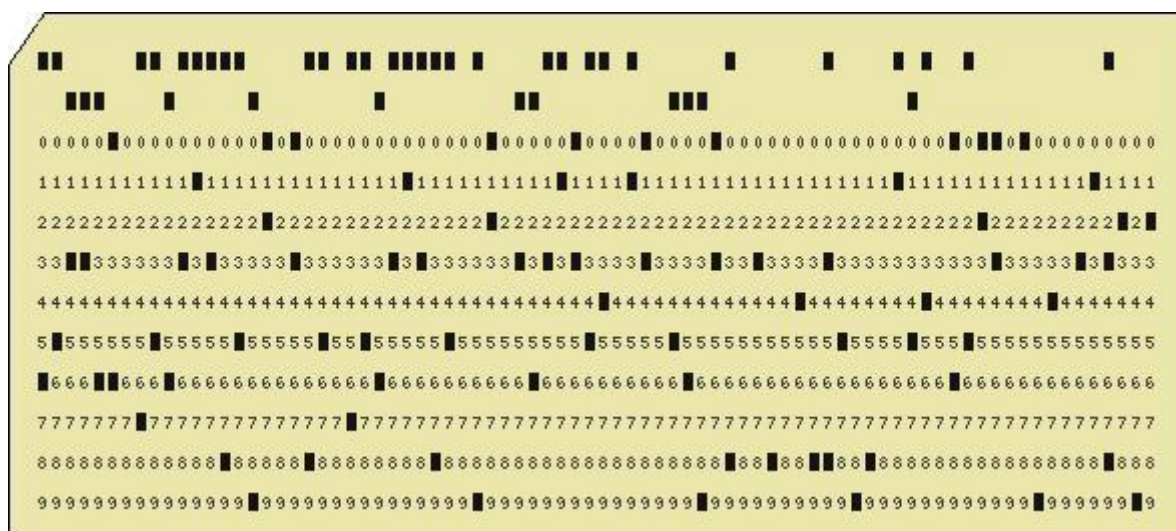


ساختار یک job به صورت دسته‌ای از کارت‌ها (job)

(کارت‌های کنترلی باید به زبان JCL باشند)

Job control language (JCL)

در سیستم‌های دسته‌ای قدیمی، کاربر نیازهای خود را از طریق یک سری دستور به نام JCL به سیستم عامل می‌داد. سیستم عامل نیز بر مبنای دستورات JCL منابع مورد نیاز را در اختیار آن برنامه می‌گذاشت. به عبارتی دیگر اطلاعات یک Job به صورت یک بسته متشکل از JCL، برنامه‌ها و داده‌ها (Data + JCL + Program) به سیستم عامل داده می‌شود. بنابراین JCL حاوی اطلاعاتی است که کاربر به سیستم عامل می‌دهد و به آن می‌گوید چه کار کرده (از چه کامپایلری استفاده کند، روی چه داده‌ای کار کند و ...) و از چه منابعی استفاده کند.



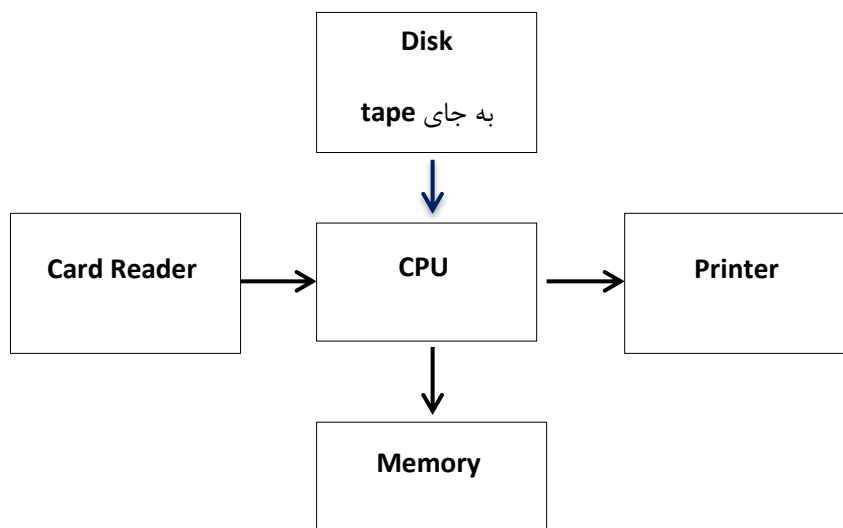
شمای یک کارت پانچ

نسل سوم کامپیوترها (۱۹۸۰-۱۹۶۵)

در این نسل در سخت افزار اتفاقات جالبی افتاده بود:

- ۱- ظهور ICها
- ۲- ظهور هارددیسک که یکی از تحولات این نسل است. آنقدر مهم که مایکروسافت نام اولین سیستم عامل خود را DOS به معنای Disk operating System گذاشت.
- ۳- مکانیزم‌هایی مانند buffering، interrupt، online spooling در این نسل آمده است.
- ۴- در اواخر این نسل ظهور ترمینال‌ها باعث شد ارتباط با کاربر online و interactive شود.

سازمان کامپیوترهای این نسل:



چند برنامه‌ریزی (Multi Programming)

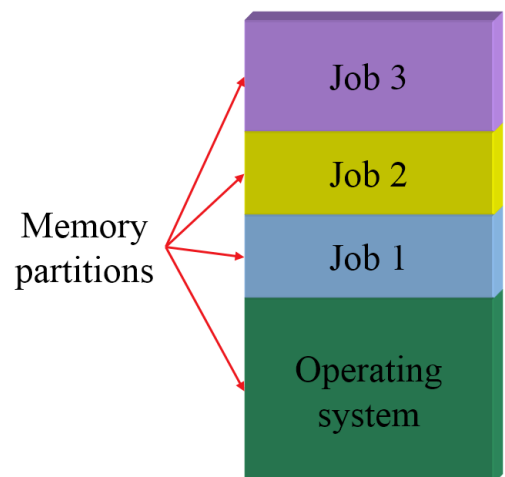
اصولا کارها در کامپیوتر به دو دسته تقسیم می‌شود:

- ۱- کارهای محدود به محاسبه یا محدود به CPU (CPU-bound) که به محاسبات علمی سنگین می‌پردازند و I/O به ندرت در آن‌ها به کار می‌رود. زمان تلف شده در این نوع کارها اهمیت چندانی ندارد.
- ۲- برنامه‌های محدود به I/O (I/O-bound) مانند پردازش داده‌های تجاری که زمان انتظار I/O اغلب ۸۰ تا ۹۰ درصد کل زمان را به خود اختصاص می‌دهد.

در سیستم‌های تک برنامه‌ی، درصد بالایی از وقت CPU به هدر می‌رود؛ بنابراین باید برای پرهیز از این همه بیکاری پردازنده چاره‌ای بیابیم. راه حل ارائه شده این بود که حافظه را به چند تکه تقسیم کنیم و همان طور که در شکل دیده میشود در هر پارتیشن یک کار مجزا قرار دهیم. البته باید ترکیب مناسبی از کارهای I/O bound و CPU bound برای بارگذاری در حافظه انتخاب شود. وقتی که یک کار برای تکمیل عملیات I/O منتظر می‌ماند بلافاصله یکی دیگر از کارهای درون حافظه، پردازنده را در اختیار می‌گیرد. اگر تعداد کارهای موجود در حافظه کافی باشد می‌توان پردازنده را صد در صد مشغول نگه داشت. البته نگهداری همزمان چند برنامه در حافظه نیاز به مدیریت خاص حافظه دارد تا برنامه‌ها بر همدیگر اثر سوء نداشته باشند.

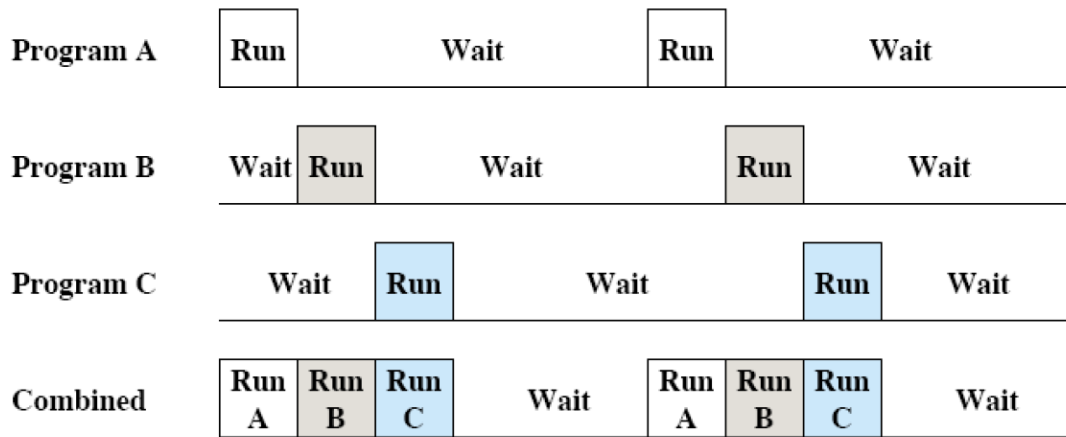
کاربر درخواست خود را وارد می‌کند. بعد کامپیوتر آن را اجرا می‌کند و کاربر نتایج خود را بعداً می‌گیرد (چند دقیقه یا چند ساعت بعد). در چند برنامه‌ی اجرای یک کار تا زمان نیاز به ورودی یا خروجی انجام می‌گیرد. سپس پردازنده اجرای کار دیگری را شروع کرده یا ادامه می‌دهد.

- Multiple jobs in memory
- Memory partitioning
- Protected from one another
- Resources (time /hardware) split between jobs
- Still **not** interactive



مزیت Multi Programming

- ✓ زمان کمتری هدر می‌رود.
- ✓ استفاده بهینه از منابع (حافظه ، CPU ، I/o)



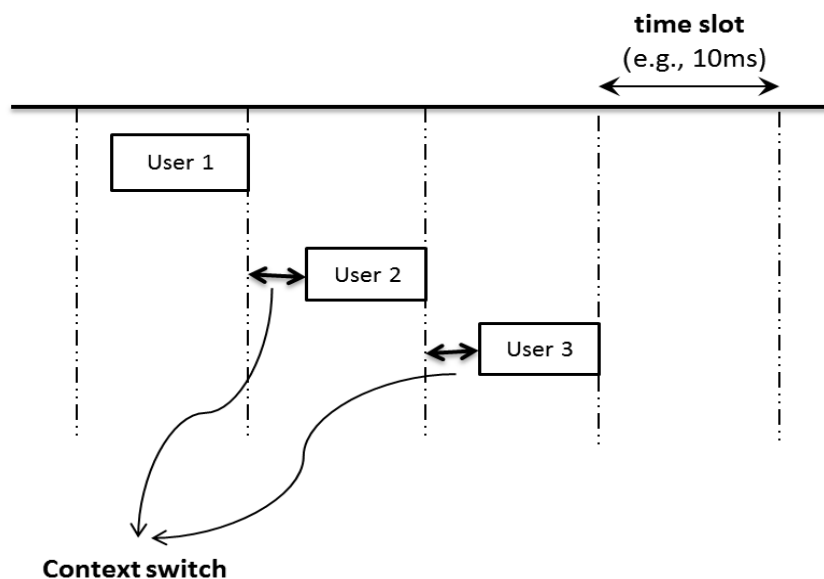
سیستم عامل دارای قابلیت چند برنامه‌گی به مراتب پیچیده‌تر از حالت تک برنامه‌گی است:

- نیاز به مدیریت حافظه داد تا حافظه را به برنامه‌های متعددی اختصاص دهد.
- نیاز به زمانبندی CPU دارد تا از بین job های آماده، یکی را برای اجرا انتخاب نماید.
- متکی به تواناییهای سخت افزاری ویژه‌ای برای ورودی/خروجی می‌باشد. (مانند وقفه و DMA)

مفهوم Time_sharing (اشتراک زمانی)

برای کاربردهای محاوره‌ای (Interactive) لازم است که میزان معطلی اجرای برنامه‌ها کم شود. پس زمان اجرا بین آنها پخش می‌شود. در این روش زمان به بازه‌های کوچکی به نام کوانتوم یا time slice تقسیم می‌شود. در پایان کوانتوم، زمان‌سنج، یک Interrupt سخت افزاری می‌فرستد. وقتی برنامه‌ای وقتش به پایان رسید، کنترل به زمانبند سپرده می‌شود و برنامه بعدی را مشخص می‌کند.

در این حالت هر کاربر فکر می‌کند فقط خودش با سیستم کار می‌کند. چون معمولاً پردازنده بسیار سریع است.



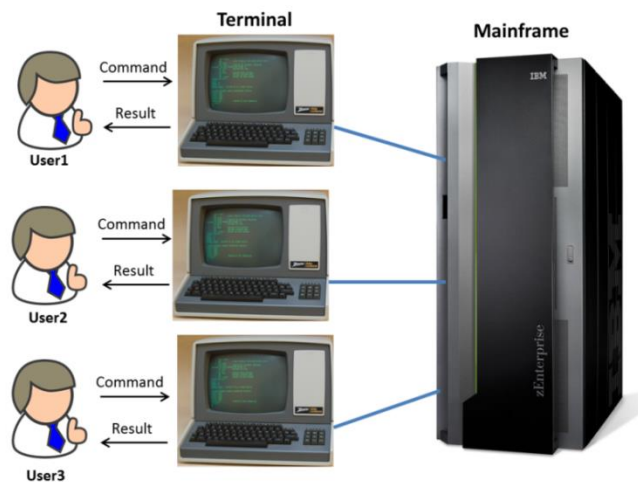
مزایای Time sharing

✓ ارزان بودن ترمینال‌ها (Cheap terminals)

✓ Interactive (تعاملی) بودن: یعنی اینکه جواب درخواستی که داده‌ایم زود به ما داده می‌شود. مثل

کامپیوترهای امروزی

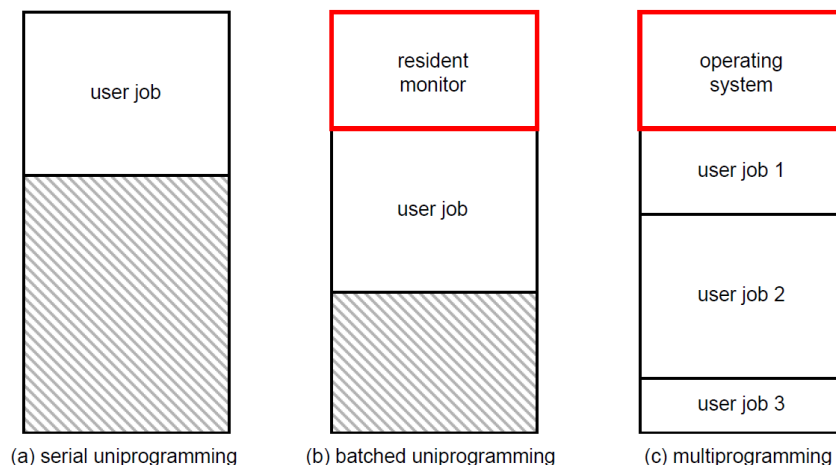
✓ زمان پاسخ (response time) بهینه‌تر می‌گردد.



توجه: Terminal یک کامپیوتر نیست بلکه یک دستگاه ورودی/خروجی است.

Context switch time: زمان تعویض متن. هنگامی که وقفه در سیستم عامل رخ می‌دهد ابتدا سیستم عامل وضعیت کامل برنامه در حال اجرا را حفظ می‌کند. سپس وقفه را بررسی می‌کند و کنترل را به یک روال وقفه مناسب تحویل می‌دهد. پس از آن سپس به برنامه جدید مراجعه می‌شود. به این جریان کاری "تعویض متن" گفته می‌شود.

Summary: serial, batched uni-, and multiprogramming



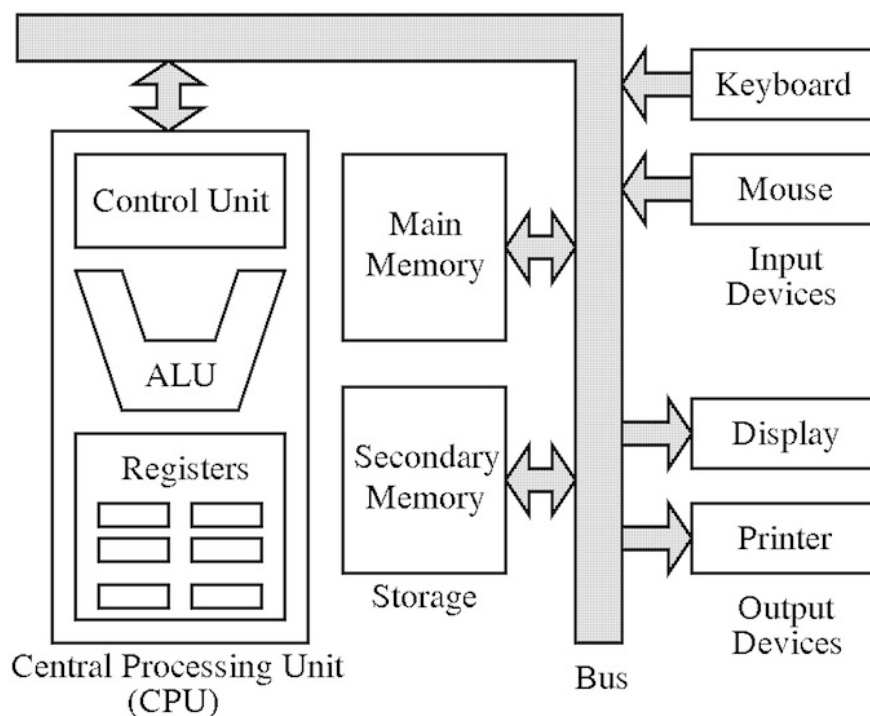
نسل چهارم (ظهور تکنولوژی VLSI)

در این نسل مدارهای مجتمع بسیار فشرده VLSI ظهور کردند که به آنها میکرو پروسور یا ریزپردازنده می‌گویند. تراکم و اندازه IC ها کوچک‌تر می‌شود که خود موجب کاهش حجم و کاهش قیمت کامپیوترها می‌شود و در نتیجه باعث ظهور کامپیوترهای شخصی و حرکت از چندکاربره به تک‌کاربره گردید. این امر خود باعث سهولت استفاده از کامپیوترهای شخصی می‌شود.

سیستم عامل و معماری کامپیوتر اثر زیادی بر روی یکدیگر داشته‌اند. یعنی جهت سهولت کار با سخت افزارهای جدید، سیستم عامل‌ها توسعه یافتند و همچنین در اثنای طراحی سیستم عامل‌ها، مشخص شد که تغییراتی در طراحی سخت افزار می‌تواند سیستم عامل‌ها را ساده‌تر و کارآمدتر سازد. به همین دلیل، یادآوری مختصری در مورد ساختار و معماری سیستمهای کامپیوتری در ادامه آورده شده است.

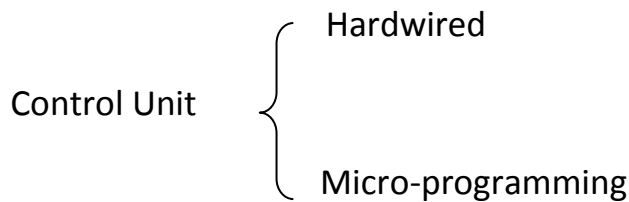
اجزای یک سیستم کامپیوتری (مروری بر برخی مفاهیم درس معماری کامپیوتر)

یک سیستم کامپیوتری دارای واحدهای ورودی و خروجی، پردازنده و حافظه است.



(معماری Von-neumann)

واحد پردازش مرکزی خود به دو بخش واحد کنترل و مسیر داده (datapath) تقسیم می‌گردد. مسیرهاده شامل واحدهای اجرایی (مانند ALU)، بخشهایی برای نگهداری اطلاعات (cache ها و رجیسترها) و مدارتی برای انتقال اطلاعات (باسها و MUX ها) می‌باشد. واحد کنترل وظیفه تولید سیگنالهای کنترلی برای مسیرهاده را برعهده دارد و می‌تواند به دو صورت پیاده سازی شود:



رجیسترها را می‌توان به دو دسته تقسیم نمود:

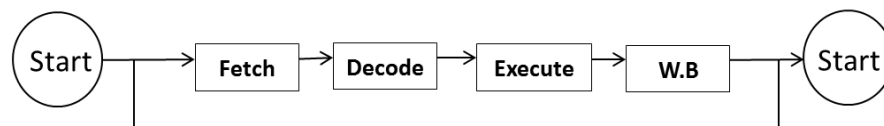
✓ رجیسترهایی که توسط برنامه نویس قابل مشاهده هستند مثل DR و یا رجیسترهای R0-R31 در پردازنده

MIPS

✓ رجیسترهایی که برنامه نویس نمی‌تواند از آنها استفاده کند و توسط واحد کنترل استفاده می‌شوند.

مثل PC , IR

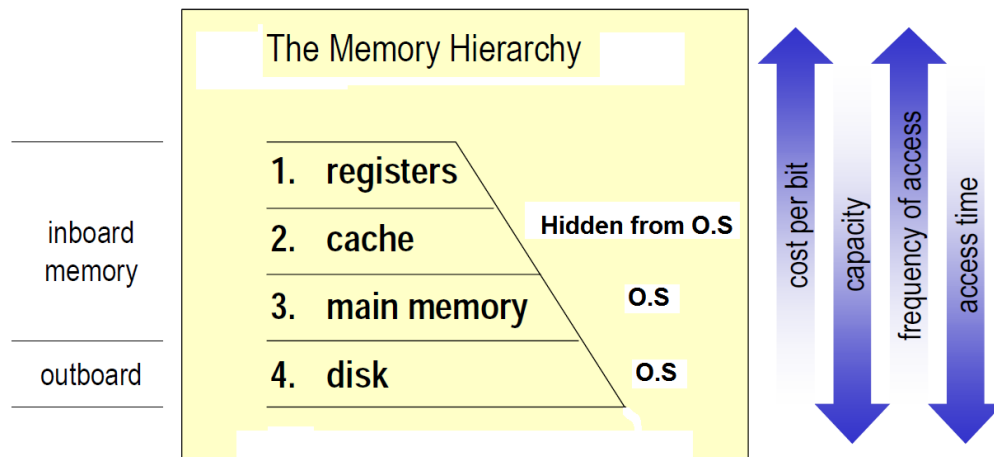
Instruction Cycle



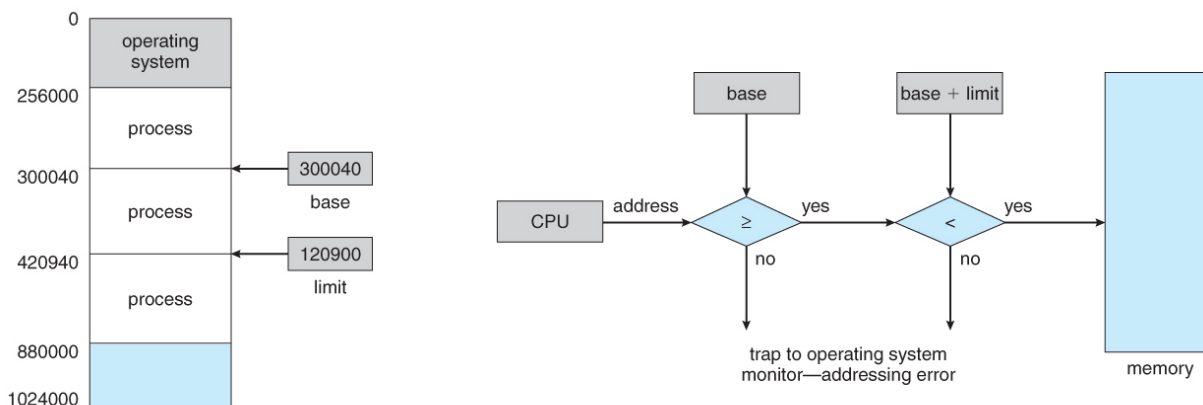
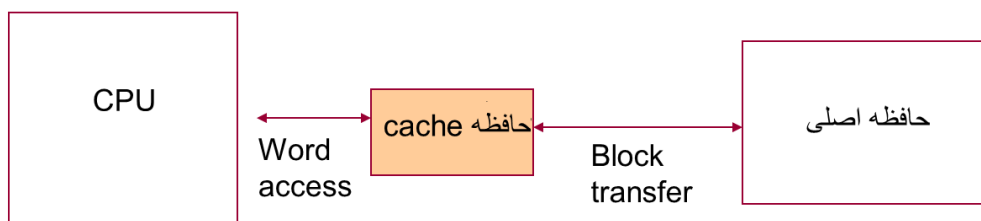
Fetch: واکنشی دستورالعمل از حافظه (یا cache) به یک رجیستر خاص در داخل پردازنده

W.B: نوشتن نتیجه در حافظه یا رجیستر

سلسله مراتب حافظه:



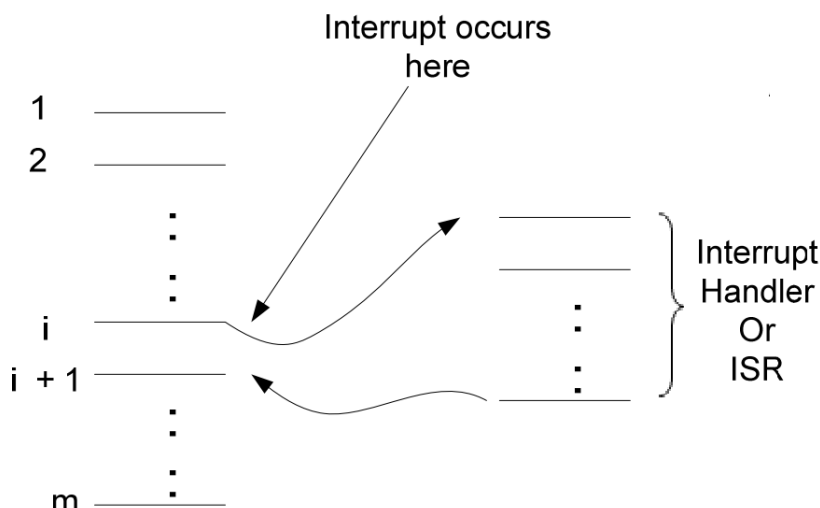
سیستم عامل وظیفه مدیریت حافظه اصلی و ثانویه (دیسک) را بر عهده دارد. حافظه cache کاملاً از دید سیستم عامل پنهان است.



استفاده از رجیسترهای **base** و **limit** برای محافظت برنامه‌ها از دسترسی غیرمجاز همدیگر

مفهوم وقفه:

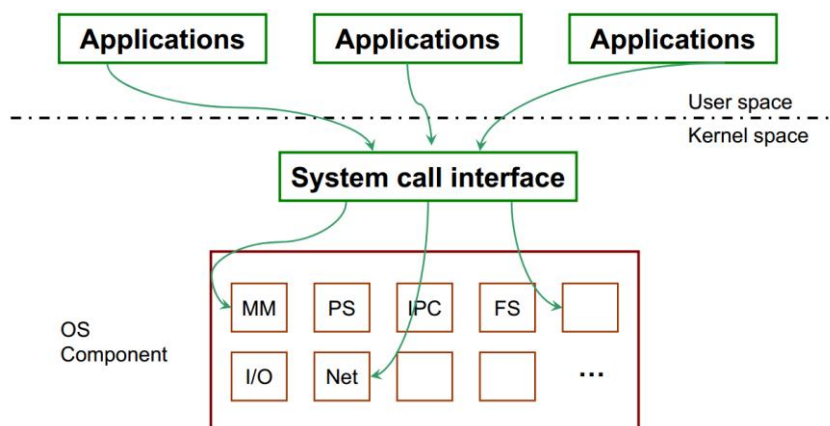
سیگنال یا دستوری است که باعث توقف قراربند جاری و رفتن به روالی موسوم به ISR می‌گردد. وقفه یک مکانیسم حیاتی برای سیستم عامل است.



سخت افزاری (وقفه های I/O - تایمر): ناهمگام
 نرم افزاری (تقسیم بر صفر - overflow - system calls): همگام

انواع وقفه

System call یا فراخوانی سیستمی نوعی وقفه نرم افزاری بوده و درخواستی است که یک فرایند سطح کاربر از هسته سیستم عامل (برای هدف خاصی) می‌نماید. در حقیقت فراخوانی سیستمی نقاط ورود به هسته سیستم هستند.

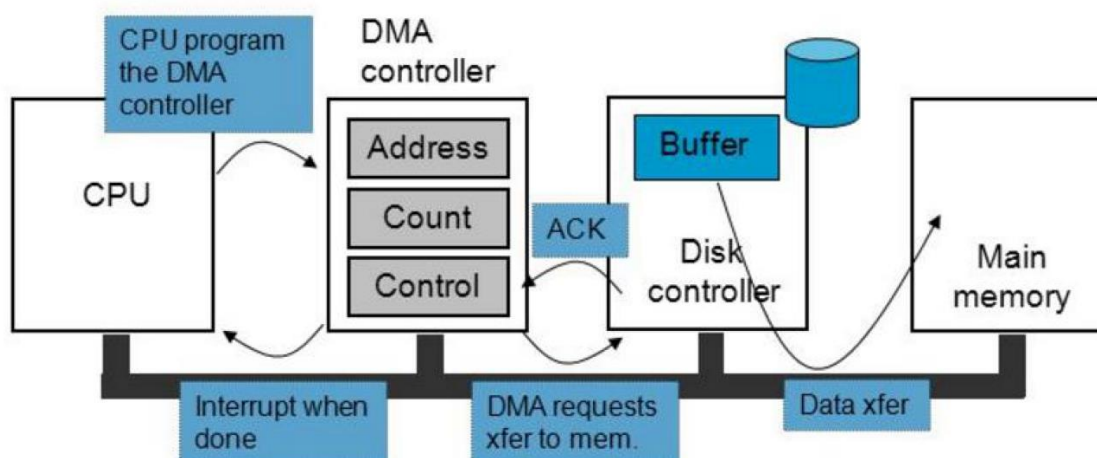


در سیستم عامل Unix بیش از ۳۰۰ فراخوانی سیستمی و Windows نزدیک به ۲۰۰۰ فراخوانی سیستمی دارد. در جدول زیر تعدادی از فراخوانیهای سیستمی در سیستم عاملهای Unix و Windows و همچنین ارتباط یک تابع کتابخانه‌ای در زبان C با یک فراخوانی سیستمی در هسته سیستم عامل نشان داده شده است.

UNIX	Win32	Description
fork	CreateProcess	Create a new process
waitpid	WaitForSingleObject	Can wait for a process to exit
execve	(none)	CreateProcess = fork + execve
exit	ExitProcess	Terminate execution
open	CreateFile	Create a file or open an existing file
close	CloseHandle	Close a file
read	ReadFile	Read data from a file
write	WriteFile	Write data to a file
lseek	SetFilePointer	Move the file pointer
stat	GetFileAttributesEx	Get various file attributes
mkdir	CreateDirectory	Create a new directory
rmdir	RemoveDirectory	Remove an empty directory
link	(none)	Win32 does not support links
unlink	DeleteFile	Destroy an existing file
mount	(none)	Win32 does not support mount
umount	(none)	Win32 does not support mount
chdir	SetCurrentDirectory	Change the current working directory
chmod	(none)	Win32 does not support security (although NT does)
kill	(none)	Win32 does not support signals
time	GetLocalTime	Get the current time

مفهوم DMA (دسترسی مستقیم به حافظه)

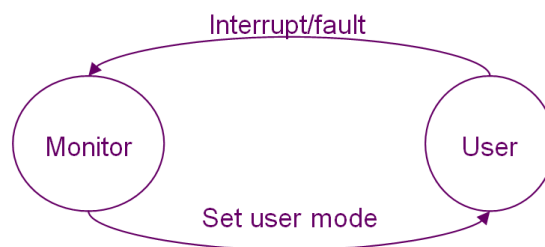
یک ویژگی در سیستمهای کامپیوتری است که به بعضی از دستگاههای I/O اجازه دسترسی به حافظه سیستم را بصورت مستقل از واحد پردازش مرکزی می‌دهد (تحت نظارت DMA controller). با استفاده از DMA، پردازنده تنها فرایند انتقال را آغاز کرده، در هنگامی که انتقال در جریان است مشغول کارهای دیگر می‌شود.



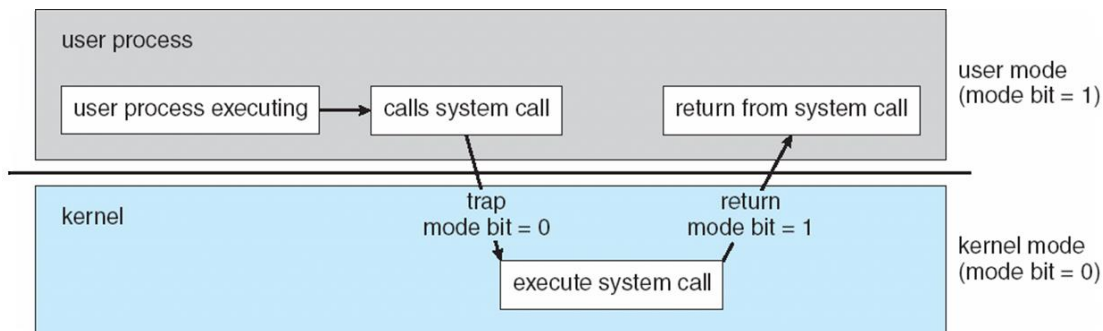
پشتیبانی سخت افزاری از سیستم عامل

سیستم عامل برای عملکرد مؤثر به چندین ویژگی کلیدی از سخت افزار زیرین متکی است. در اینجا برخی از پشتیبانی‌های سخت افزاری ضروری آورده شده است:

حالت‌های پردازش چندگانه: CPU باید حداقل از دو سطح امتیاز پشتیبانی کند که معمولاً به عنوان حالت هسته و حالت کاربر شناخته می‌شود. حالت هسته به سیستم عامل اجازه می‌دهد تا دستورالعمل‌های ممتاز را با دسترسی کامل به منابع سیستم اجرا کند. حالت کاربر کاری را که برنامه‌ها می‌توانند انجام دهند محدود می‌کند و از تداخل آنها با سیستم عامل یا سایر برنامه‌ها جلوگیری می‌کند. در سیستم‌های کامپیوتری بین کد مربوط به هسته سیستم عامل و کد مربوط به برنامه کاربر تمایز وجود دارد. برای این کار در سخت افزار یک بیت به نام بیت حالت (mode bit) وجود دارد. با این کار دو حالت ممتاز و عادی خواهیم داشت. به حالت ممتاز monitor mode یا kernel mode یا system mode گفته می‌شود.



برخی دستورات به طور خاص مجاز (designated as privileged) به حساب می‌آیند که تنها در مد هسته اجرا می‌شوند. فراخوانی سیستمی (System Call) مد را به هسته تغییر می‌دهد و پس از خروج از تابع مربوطه مد به حالت کاربری تغییر می‌کند.



مدیریت حافظه: سیستم عامل برای مدیریت کارآمد حافظه به پشتیبانی سخت افزاری نیاز دارد. این شامل ویژگی‌هایی مانند واحدهای مدیریت حافظه (MMU) برای ترجمه آدرس‌های حافظه مجازی مورد استفاده برنامه‌ها به آدرس‌های حافظه فیزیکی است. علاوه بر این، پشتیبانی سخت افزاری از حافظه مجازی به سیستم عامل اجازه می‌دهد تا برنامه‌های بزرگتر از حافظه فیزیکی موجود را با مبادله داده‌ها بین حافظه رم و دستگاه‌های ذخیره سازی اجرا کند. همچنین، پشتیبانی سخت افزاری برای مکانیسم‌های حفاظت از حافظه بسیار مهم است. این کار تضمین می‌کند که برنامه‌هایی که در حالت کاربر اجرا می‌شوند نمی‌توانند به حافظه مورد استفاده هسته یا سایر برنامه‌ها دسترسی پیدا کنند و از خرابی‌ها و آسیب‌پذیری‌های امنیتی جلوگیری می‌کند (با استفاده از ازجیسترهای مخصوص). تکنیک‌هایی مانند بخش بندی حافظه (segmentation) و صفحه‌بندی (paging) با کمک و حمایت سخت افزار اجرا می‌شوند.

مدیریت وقفه: وقفه به دستگاه‌ها اجازه می‌دهد زمانی که نیاز به توجه دارند، مانند زمانی که انتقال داده‌ها کامل شده است، CPU را مطلع کنند. سیستم عامل برای مدیریت کارآمد دستگاه‌های ورودی/خروجی به وقفه‌ها متکی است. برای مدیریت وقفه نیاز به یک جدول سخت افزاری است (جدول بردار وقفه) که آدرس‌روتین‌های سرویس وقفه خاص (ISR) را ذخیره کند. هنگامی که یک وقفه رخ می‌دهد، CPU برای مکان‌یابی و اجرای ISR مناسب برای مدیریت رویداد به این جدول مراجعه می‌کند.

تایمر: تایمر یک دستگاه سخت افزاری است که وقفه‌های دوره‌ای را در فواصل زمانی مشخص ایجاد می‌کند. سیستم عامل می‌تواند از تایمر برای کارهای مختلف از جمله زمان‌بندی فرآیندها و جابجایی متن بین برنامه‌های در حال اجرا استفاده کند.

دسترسی مستقیم به حافظه (DMA): همانطور که قبلاً گفته شد، این امکان به دستگاه‌ها اجازه می‌دهد تا داده‌ها را مستقیماً و بدون دخالت CPU به حافظه منتقل کنند و عملکرد کلی سیستم را بهبود بخشد.

فراهم کردن دستورالعمل‌های اتمی (Atomic Instructions): دستورالعمل‌های اتمی یک قطعه حیاتی از پشتیبانی سخت افزاری برای سیستم عامل‌ها، به ویژه در محیط‌های چند هسته‌ای هستند. دستورالعمل‌های اتمی برای اطمینان از اینکه عملیات خاصی روی داده‌های مشترک به عنوان واحدهای تقسیم‌ناپذیر اجرا می‌شوند، وارد عمل می‌شوند. به عبارت ساده‌تر، یا کل عملیات با موفقیت کامل می‌شود یا اصلاً اتفاق نمی‌افتد. با استفاده از دستورالعمل‌های اتمی، سیستم عامل‌ها می‌توانند عملکردهای حیاتی مانند همگام‌سازی و قفل کردن را انجام دهند. همگام‌سازی به معنای اطمینان از دسترسی رشته‌ها یا فرآیندها به منابع مشترک به شیوه ای کنترل شده، می‌باشد.

الزامات سخت افزاری خاص برای یک سیستم عامل بسته به پیچیدگی و پلتفرم هدف آن می تواند متفاوت باشد. سیستم‌های تعبیه‌شده ساده ممکن است به حداقل قابلیت‌ها متکی باشند، در حالی که سیستم‌عامل‌های سرور از طیف وسیع‌تری از ویژگی‌های سخت‌افزاری برای بهبود عملکرد، امنیت و قابلیت‌های مجازی‌سازی استفاده می‌کنند. با این حال، عملکردهای اصلی ذکر شده در بالا برای هر سیستم عاملی برای مدیریت موثر منابع و فراهم کردن بستری برای اجرای برنامه‌ها ضروری است.

انواع سیستمهای عامل:

• سیستم عاملهای Mainframe

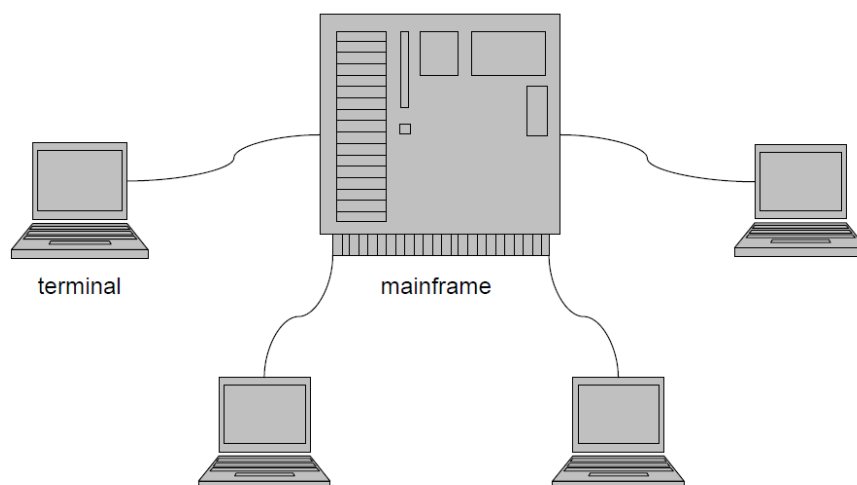
سیستم‌های عامل mainframe بیشتر برای برنامه‌هایی که شامل بارهای کاری سنگین ورودی/خروجی هستند، مانند مدیریت پایگاه داده، پردازش تراکنش و انبارداری داده، مناسب هستند. مین فریم‌ها کانال‌های اختصاصی برای مدیریت عملیات ورودی/خروجی دارند که امکان انتقال داده‌های موازی را فراهم می‌کنند و دخالت پردازنده را کاهش می‌دهند. آن‌ها می‌توانند حجم عظیمی از داده‌ها را ذخیره کنند و نیاز به عملیات ورودی/خروجی مکرر را کاهش دهند.

- ظرفیت I/O: ۱۰۰۰ دیسک و هزاران Gigabyte داده

- به عنوان Server های دارای کاربران همزمان خیلی زیاد

- به صورت اشتراک زمانی (Time sharing)

مثالهایی از Mainframe operating systems : OS/360 , OS/390 , MVS



• سیستم عامل‌های Desktop و laptop

- تک کاربره

- هدف اصلی راحتی کاربر است

مثال: Windows , Linux, Mac OS , Free BSD

این نوع سیستم‌های عامل، به کاربر امکان می‌دهند تا به راحتی با کامپیوتر و نرم‌افزارها تعامل کند. برای این منظور، رابط‌های کاربری گرافیکی (GUI) و متنوع که از ماوس، منوها، و صفحات لمسی و حتی ورودی صوتی استفاده می‌کنند، برای تعامل با کامپیوترها ارائه می‌دهند. همچنین، امکاناتی برای مدیریت فایل‌ها، شبکه‌سازی و امنیت را فراهم می‌کنند.

• سیستم‌های چند پردازشی، شبکه‌ای و توزیع شده

در سازمان‌هایی که نیازمند انجام محاسبات سنگین یا سرویس دهنده‌هایی با حجم وسیع درخواست‌ها هستند، تمایلاتی به سمت سیستم‌های چند پردازشی یا موازی وجود دارد. در این سیستم‌ها چندین پردازنده به طور موازی به اجرای کارها می‌پردازند.

سیستم‌های موازی به دو دسته تقسیم می‌شوند :

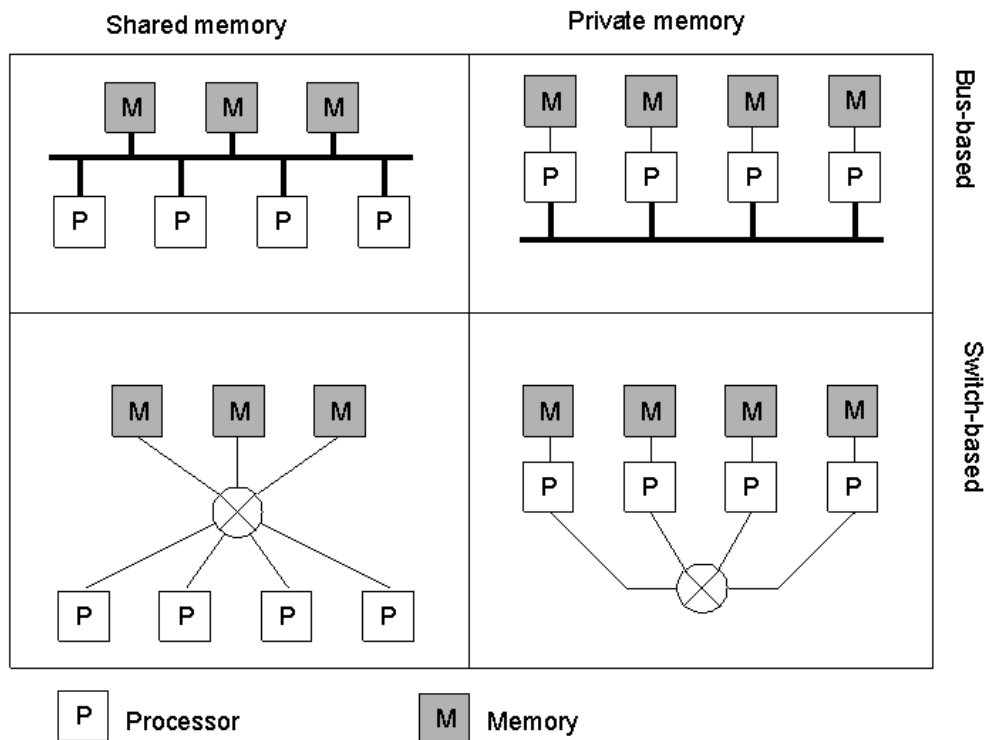
۱- چند پردازنده (Multi-processors): این نوع سیستم‌ها خود به دو دسته تقسیم می‌شوند.

✓ چند پردازنده متقارن (Symmetric): هر پردازنده به طور مجزا می‌تواند هسته سیستم عامل واحد روی حافظه مشترک را اجرا کند و مجموعه فرایندها و نخ‌های خود را زمانبندی کند.

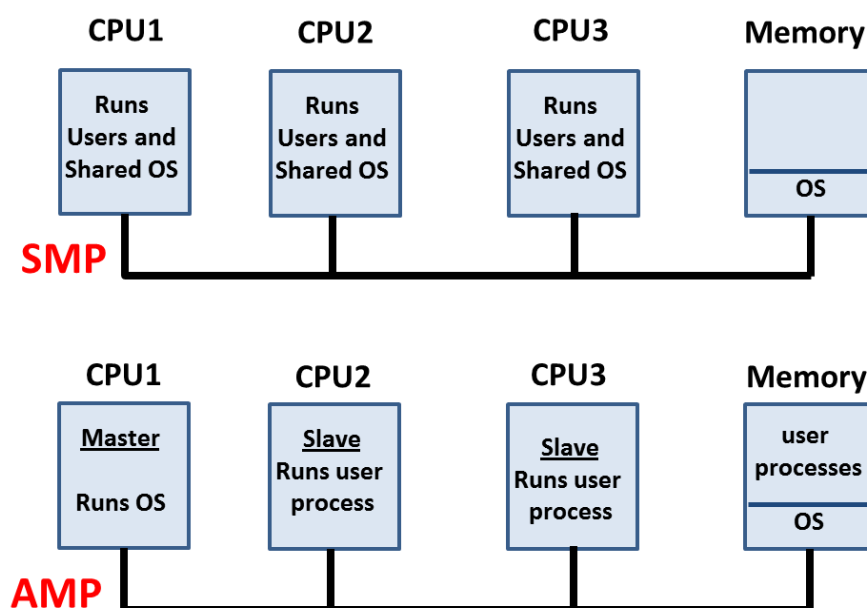
✓ چند پردازنده نامتقارن (Asymmetric): مانند پردازنده‌های (MASTER/SLAVE) که معمولاً سیستم عامل روی یک پردازنده MASTER اجرا می‌شود و زمان بند واحد آن، برنامه‌های سطح کاربر را بر روی سایر پردازنده‌ها توزیع می‌کند.

۲- چند کامپیوتری (Multi-computers): چندین پردازنده که هر کدام دارای حافظه اختصاصی

خود هستند.



- **Symmetric multiprocessing (SMP)**
 - Each processor runs an identical copy of the operating system.
 - Many processes can run at once without performance deterioration.
 - Most modern operating systems support SMP
- **Asymmetric multiprocessing**
 - Each processor is assigned a specific task; master processor schedules and allocates work to slave processors.
 - More common in extremely large systems



نقاط قوت سیستم‌های موازی عبارتند از :

۱- توان عملیاتی بالا

۲- صرفه جویی اقتصادی به علت امکان به اشتراک گذاشتن منابع

۳- افزایش قابلیت اطمینان و تحمل پذیری خطا

معایب اساسی سیستم‌های موازی عبارتند از:

۱- پیچیدگی در ایجاد توازی و همگام سازی فرآیندها و نخ‌ها

۲- مشکلات در راه ایجاد توازی و همگام سازی در مدیریت حافظه

• سیستم‌های عامل بلادرنگ

در سیستم‌های بلادرنگ (Real-Time)، هر فرایند مهلت زمانی خاصی برای انجام دارد. زمان پاسخ در این سیستم‌ها الزاماً باید به موقع و تضمین شده باشد. در مقابل، در سیستم‌های اشتراک زمانی، داشتن زمان پاسخ کوتاه مطلوب است ولی الزامی نیست. در سیستم‌های دسته‌ای نیز اصلاً محدودیت زمانی وجود ندارد. این سیستم‌ها به دو دسته تقسیم می‌شوند:

۱- **بلادرنگ سخت:** در این سیستم‌ها در تمام مواردی که رویدادی به وقوع می‌پیوندد که نیازی به

اجرای یک وظیفه بلادرنگ (Real-Time Task) دارد، اجرای آن وظیفه باید حتماً قبل از پایان

مهلت تعیین شده تمام شود؛ در غیر این صورت، ممکن است فاجعه‌ای به بار آید و با مشکل یا

خرابی غیر قابل جبرانی مواجه شویم. یعنی پس از پایان مهلت زمانی، اجرای وظیفه بی‌معنی است.

مانند بعضی از رویدادها در یک نیروگاه هسته‌ای، هواپیماهای نظامی، روبات‌های صنعتی، سیستم کنترل پرواز،

پرتونگاری پزشکی و

این سیستم‌ها تضمین می‌کند که کارهای بحرانی را به موقع انجام دهد. این سیستم‌ها به دلیل محدودیت زمانی

بسیاری از خصوصیات سیستم‌های عامل مدرن را استفاده نمی‌کنند.

۲- **بلادرنگ نرم:** در این سیستم‌ها، رعایت مهلت زمانی مطلوب است ولی اجباری نیست و تضمین نمی

شود. به عبارت دیگر، سیستم سعی می‌کند کار خود را در مهلت زمانی خاص انجام دهد، ولی اجباری

در آن وجود ندارد. حتی پس از پایان مهلت زمانی، اجرای وظیفه معنا دارد. مانند: سیستم‌های چند

رسانه‌ای (صوتی و تصویری)، واقعیت مجازی و غیره ... از ویژگی‌های این روش آن است که برخی کارها اولویت بالایی دارند. اصولاً این سیستم‌ها قابل ترکیب با سیستم‌های دیگر نیز هستند. در هر حال، در اینجا احتمال خطا یا تأخیر وجود دارد.

• سیستم‌های تعبیه شده (توکار)

معمولاً سیستم‌های تعبیه شده (Embedded System) به این منظور ایجاد می‌شوند که اجرای وظایف معینی را در داخل یک دستگاه خاص، مدیریت کنند. درواقع این سیستم‌ها، خاص منظوره و معمولاً کوچک هستند. برای مثال برای کنترل وسایلی مانند وسایل خانگی و اتومبیل‌ها استفاده می‌شوند. سخت افزار این سیستم ها، اغلب به صورت یک تراشه یا یک برد کوچک است. این سیستم‌ها معمولاً دو مد کاربر و هسته ندارند و اجرای وظایف در بعضی از آنها به صورت بلادرنگ است.

• سیستم عامل های شبکه‌ای

پیشرفت‌های جالبی که در اواسط دهه‌ی 1980 انجام شد، گسترش شبکه‌های کامپیوترهای شخصی بود که سیستم عامل‌های شبکه‌ای را به اجرا در می‌آورد. بر خلاف سیستم عامل تک کاربره که امور اساسی مربوط به عملکرد یک کامپیوتر را بر عهده می‌گیرد، سیستم عامل شبکه باید به درخواستهای ایستگاههای کاری متعدد پاسخ داده و اموری چون ایجاد و اداره حساب کاربران، دستیابی به شبکه، استفاده و به اشتراک گذاری منابع، حفاظت از داده ها و کنترل خطاها را نیز انجام دهد. از سیستمهای عامل شبکه به عنوان مثال می‌توان به Novel networkware و Windows 2000 server اشاره کرد.

• سیستم های عامل توزیع شده

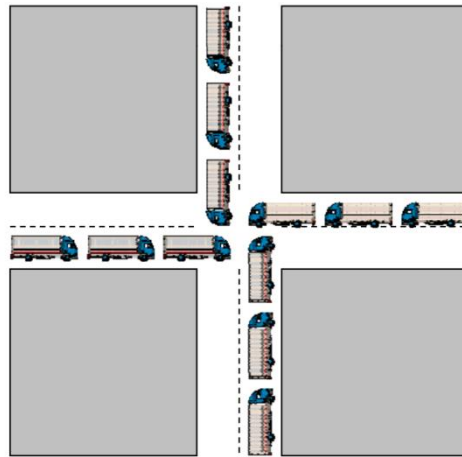
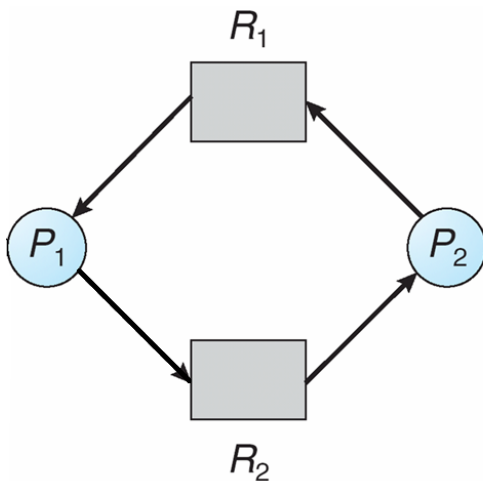
این سیستم‌های عامل در محیط‌های چند پردازنده و چند کامپیوتری همگن به وجود آمده‌اند. اگرچه در این محیط‌ها چندین پردازنده مستقل وجود دارد اما بر خلاف سیستم‌های عامل شبکه‌ای (که در آن کاربران از وجود چندین کامپیوتر باخبرند)، یک سیستم عامل توزیع شده (Distributed Operating System)، خود را مانند یک سیستم تک پردازنده‌ای قدیمی به کاربران نشان می‌دهد. در واقع این سیستم‌ها برای کاربران مانند یک سیستم یکتا، متمرکز و منسجم به نظر می‌آیند. کاربران نباید از این امر آگاه شوند که برنامه آن‌ها در کجا به اجرا

در می‌آید و یا فایل‌های آن‌ها در کجا قرار دارد به عبارت دیگر سیستم باید از دیدگاه کاربر، شفاف یا نامرئی باشد. سیستم‌های متمرکز و توزیع شده، از اساس با یکدیگر متفاوت‌اند و سیستم‌های توزیع شده بسیار پیچیده‌تر هستند.

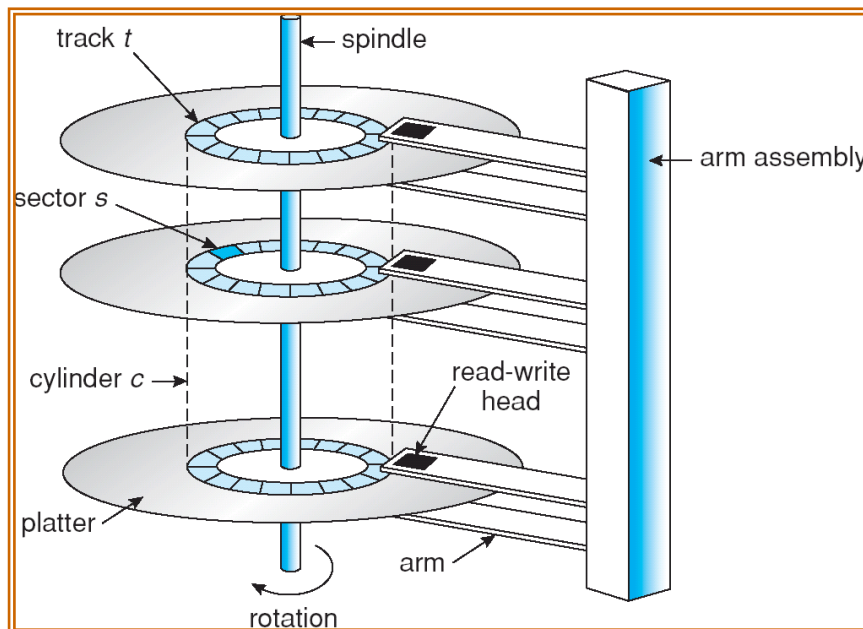
سیستم‌های دیگری نیز وجود دارند مانند: *peer-to-peer* , *Clustered* , *Handheld* , ...

اجزای یک سیستم عامل:

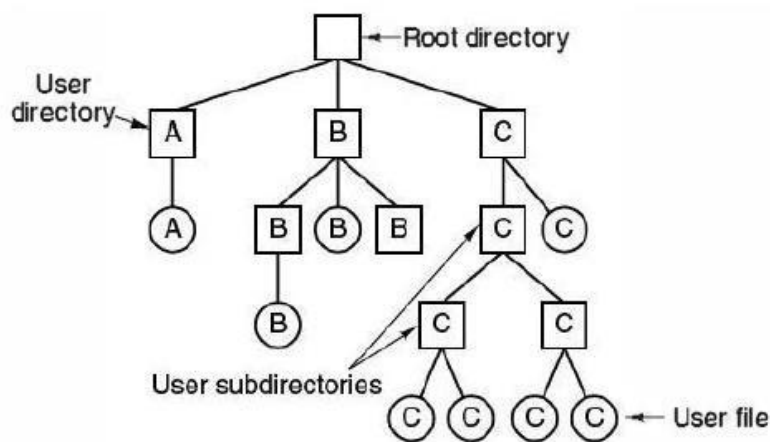
- مدیریت فرایند (Process management): شامل ایجاد و حذف فرایندها - همگام سازی فرایندها - معلق کردن و شروع مجدد یک فرایند - ارتباط فرایندها
- بن بست: شامل روشهایی برای کشف بن بست - جلوگیری و یا اجتناب از بن بست



- مدیریت حافظه: شامل تخصیص حافظه و پس گیری آن- پیگیری قطعات استفاده شده و قطعات آزاد
- حافظه - مکانیسمهای صفحه بندی و قطعه بندی
- مدیریت حافظه ثانویه (دیسک): شامل الگوریتمهایی برای پاسخ به درخواستهای استفاده از دیسک با توجه به موقعیت فعلی *head* و لیست سکتورهای درخواست شده



- مدیریت ورودی/خروجی: نحوه ارتباط با پورتهای I/O – مدیریت صفهای I/O
- مدیریت فایل: ارائه یک دید منطقی و یکسان از دیتا به نام "فایل" به کاربر، مستقل از رسانه ای که روی آن ذخیره شده است. شامل ایجاد و حذف فایلها – ایجاد و حذف directoryها – مکانیسمهایی برای کار کردن با فایل

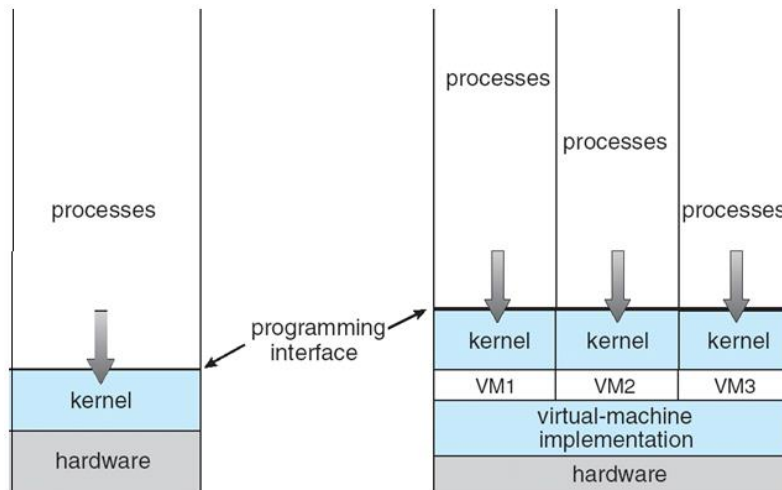


- شبکه‌سازی: شامل مکانیسمهایی برای تعریف حساب کاربران و اشتراک منابع و ...
 - زمانبند فرآیند: شامل الگوریتمهایی برای اولویت بندی برای اجرای فرآیندهای آماده
 - مفسر فرمان: واسطه ای برای کاربر که به وسیله آن می‌تواند به سیستم عامل فرمان دهد. در حقیقت فرمانهای کاربر به مجموعه‌ای از system call ها تبدیل می‌شود.
- (تبدیل میشود به مجموعه از فراخوانیها مانند open- read- write-creat-close → Copy F1 F2)
- مدیریت امنیت: شامل برای تامین امنیت کاربران از دسترسیهای غیرمجاز

مفهوم ماشین مجازی (Virtual machine)

یک ماشین مجازی در حکم واسطی است که بر روی سخت افزار قرار می گیرد و آن را بین کاربران مختلف به گونه ای تقسیم می کند که هریک از کاربران فکر می کنند به شکل انحصاری بر روی سخت افزار کار می کنند.

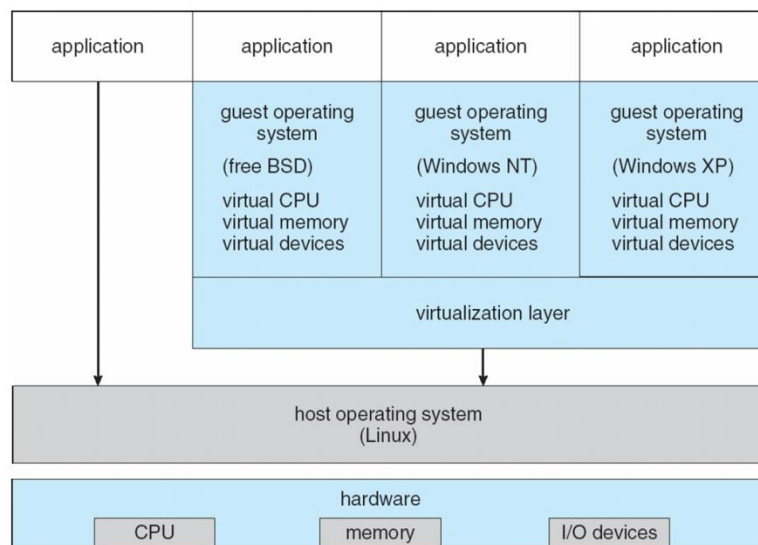
- برنامه هایی از یک سیستم عامل و سخت افزار خاص را می توان روی دیگری اجرا نمود.
- هر پروسه فکر می کند که تمام منابع سیستم را در اختیار دارد.
- دستگاههای متفاوت به نظر می رسد که یک interface واحد دارند.
- هر وقفه ای در اجرای برنامه ای ایجاد میشود، برنامه نمی داند که به دلیل اشتراک منابع است و آن را به صورت تاخیر احساس می کند.



(a) Nonvirtual machine

(b) virtual machine

مثال: ماشین مجازی VMware



۱

۲

۳

۴

۵

۶

۷

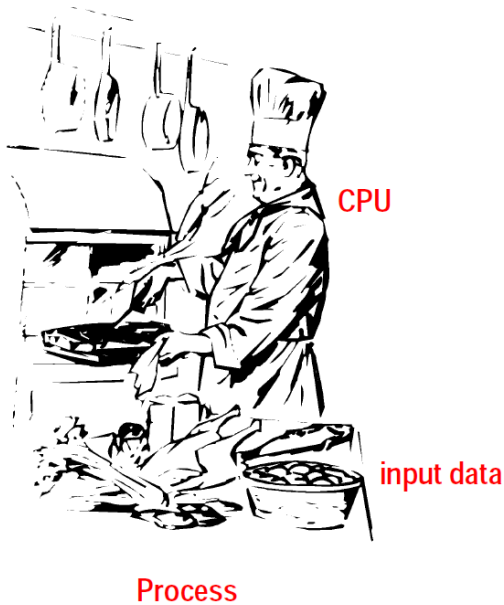
۸

فصل دوم

فرایند و نخ

مدیریت فرآیندها

فرآیند: یک برنامه است که برای اجرا در حافظه بار شده است و یک موجودیت فعال است که به منابع سیستم نیاز دارد.



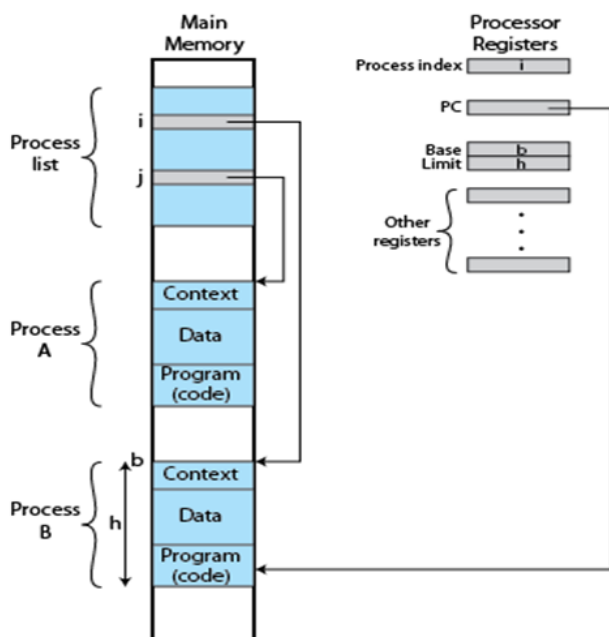
- ✓ یک برنامه در حال اجرا است.
- ✓ رویدادی از یک برنامه که روی یک کامپیوتر اجرا می‌شود.
- ✓ موجودیتی که می‌تواند به یک پردازنده اختصاص داده شده یا روی آن اجرا گردد.
- ✓ واحدی از فعالیت که با اجرای دنباله‌ای از دستورالعمل‌ها، وضعیت فعلی و مجموعه‌ای از منابع سیستمی مربوط به آن مشخص می‌شود.

اجزای یک فرآیند:

- ۱- کد برنامه (Code)
- ۲- داده‌های برنامه (متغیرها، بافرها و ...)
- ۳- وضعیت اجرای برنامه (Context)

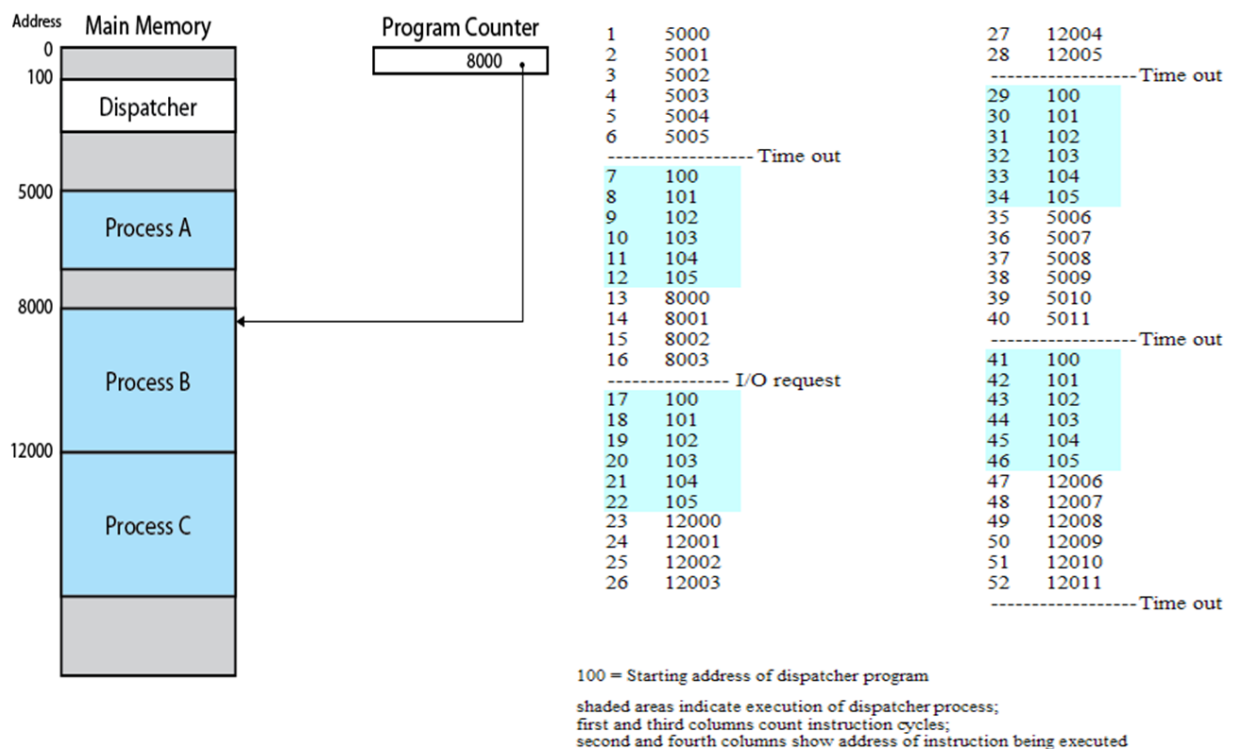
Context: اطلاعات لازم برای اجرای برنامه توسط سیستم‌عامل مانند محتوای رجیستر، اولویت فرآیند، حالت

فرآیند و ...

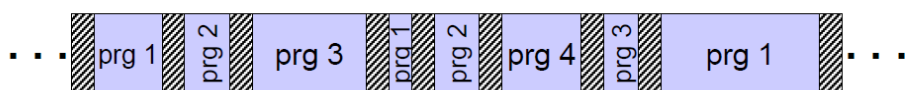


یک برنامه مثلاً word editor - چند فرآیند
مثلاً نسخه‌های متعدد با داده‌های متفاوت

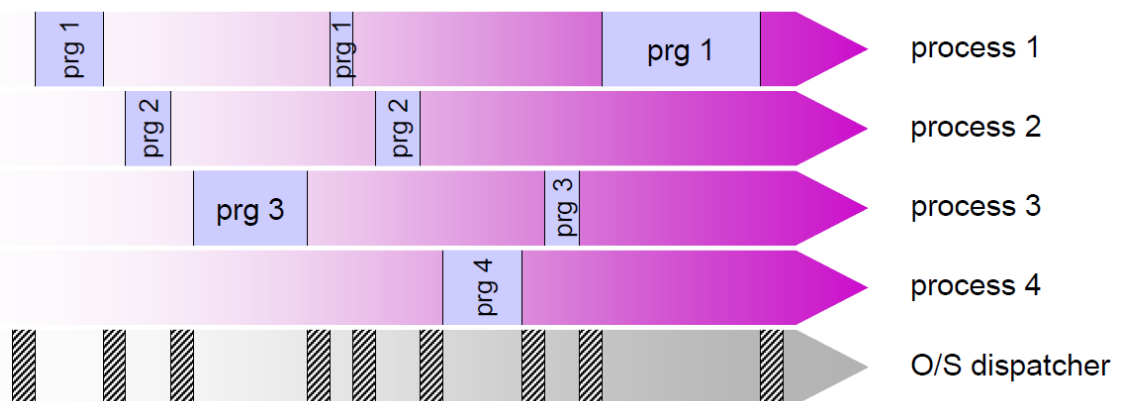
- ▶ اساسی ترین عمل پردازنده اجرای دستورالعمل های موجود در حافظه است.
- ▶ رفتار یک فرآیند به خصوص را می توان با فهرست کردن دنباله دستورالعمل هایی که برای آن فرآیند اجرا می شود مشخص نمود که به آن رد (Trace) آن فرآیند گویند.
- ▶ با نمایش چگونگی تداخل ردهای فرآیندهای مختلف، می توان رفتار پردازنده را مشخص کرد.



the dispatcher is a routine program in kernel memory space

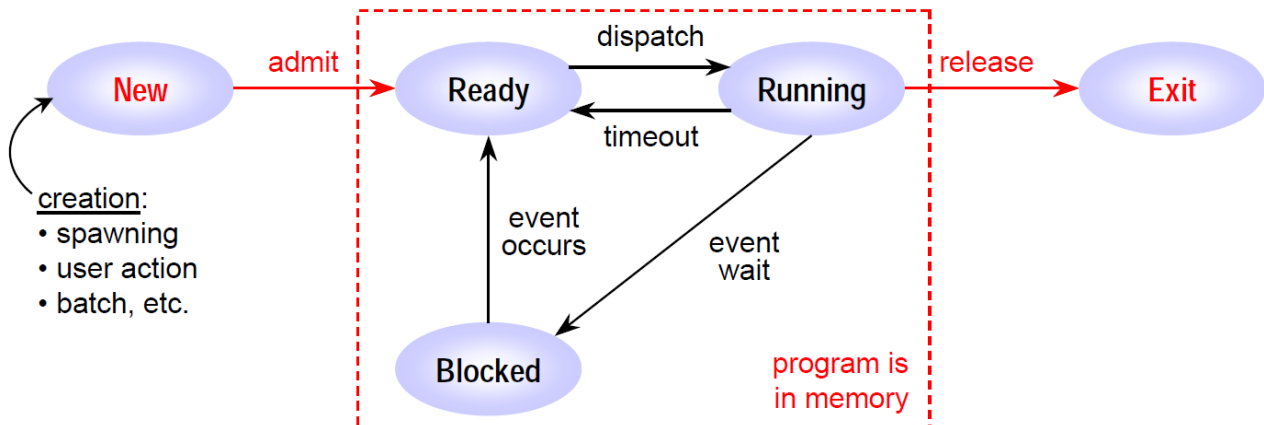


(a) Multitasking from the CPU's viewpoint



حالت‌های یک فرآیند

یک فرآیند در طول زندگی خود حالت‌های مختلفی را طی می‌کند. (مدل پنج حالت)



New: فرآیندی که تازه ایجاد شده، اما هنوز به عنوان عضوی از فرآیندهای قابل اجرای سیستم عامل تأیید نشده و در حافظه بار نشده است.

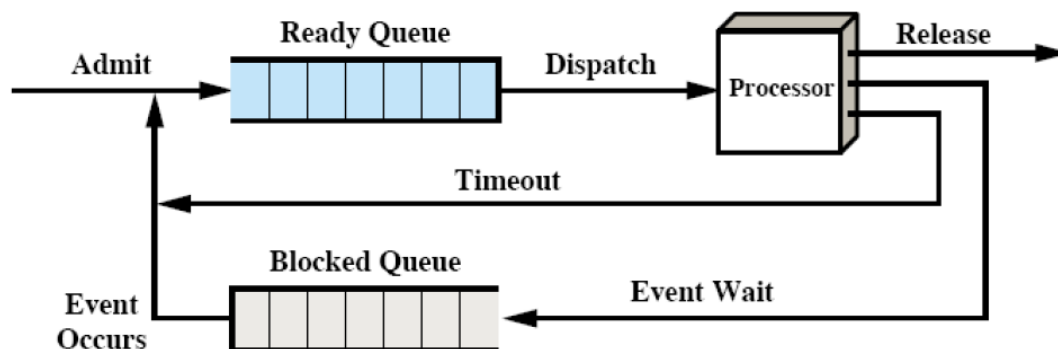
Ready: فرآیندی که آماده اجرا است و وقتی به آن فرصت داده شود، اجرا می‌شود. (همه منابع بجز CPU را در اختیار دارد)

Running: فرآیندی که فعلاً در حال اجراست. (فرآیند CPU را در اختیار می‌گیرد)

Blocked: فرآیندی که تا وقوع بعضی از رخدادها (مثل تکمیل یک عمل ورودی/خروجی) نمی‌تواند اجرا شود.

(Exit) Terminate: فرآیندی که به علت تکمیل شدن یا قطع شدن، از مخزن فرآیندهای قابل اجرای سیستم عامل خارج شده است.

نکته: قسمت‌های Ready و Waiting/Blocked دارای صف هستند. در وضعیت blocked می‌توان برای هر دستگاه I/O یک صف مجزا در نظر گرفت.



زمانبند (Scheduler)

به سه دسته تقسیم می‌شوند:

- زمانبند بلندمدت Long term scheduler (Job Scheduler) زمانبند کار – این نوع زمانبند انتخاب می‌کند کدام فرآیند به صف آماده منتقل شود. (Invoked infrequently- seconds, minutes)

در حد ثانیه و دقیقه عمل می‌کند و درجه چند برنامه‌ریزی را مشخص می‌کند.

- زمانبند میان مدت – این نوع زمانبند بنا به دلایلی ممکن است فرایندی را از حافظه به دیسک و یا بالعکس منتقل کند (swapping).

- زمانبند کوتاه مدت Short-term Scheduler (CPU Scheduler): در حدود میلی‌ثانیه است (Invoked frequently – milliseconds)

مشخص می‌کند که به کدام یک از فرآیندهای آماده، پردازنده اختصاص داده شود.

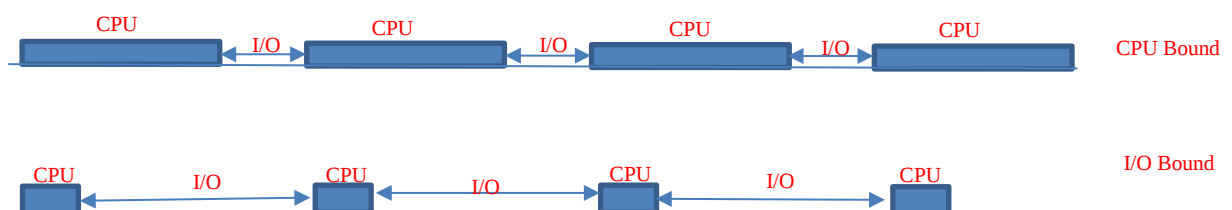
شرایط ایجاد فرآیند:

- ✓ دستور اجرا از سوی کاربر
- ✓ ورود یک کاربر جدید (در سیستم‌های Multi-User)
- ✓ شروع یک فرآیند داخلی سیستم عامل مانند چاپ و ...
- ✓ یک فرآیند والد فرزند جدیدی را ایجاد نماید.

شرایط خاتمه فرآیند:

- تکمیل اجرای فرآیند
- خروج کاربر از سیستم‌های چندکاربره
- ایجاد خطا در سیستم مثل عدم وجود فضای کافی
- خطاهای محاسباتی (مانند تقسیم بر صفر)
- ختم فرآیند والد
- خطای I/O
- درخواست پدر برای خاتمه فرآیند فرزند

فرآیند CPU Bound و I/O Bound



فرآیند CPU-Bound: فرآیندی که زمان بیشتری را به پردازش و محاسبات اختصاص می‌دهد. مانند: برنامه‌های شبیه‌ساز علمی، پردازش تصویر، رمزگذاری/رمزگشایی ویدیو و یادگیری ماشینی.

فرآیند I/O-Bound: فرآیندی که زمان بیشتری را صرف گرفتن داده یا ارسال آن به دستگاه‌های ورودی/خروجی می‌کند. مانند: انتقال فایل‌های بزرگ، پرس و جوهای پایگاه داده، و سرورهای وب.

مدل پنج‌حالت دارای اشکالات زیر است:

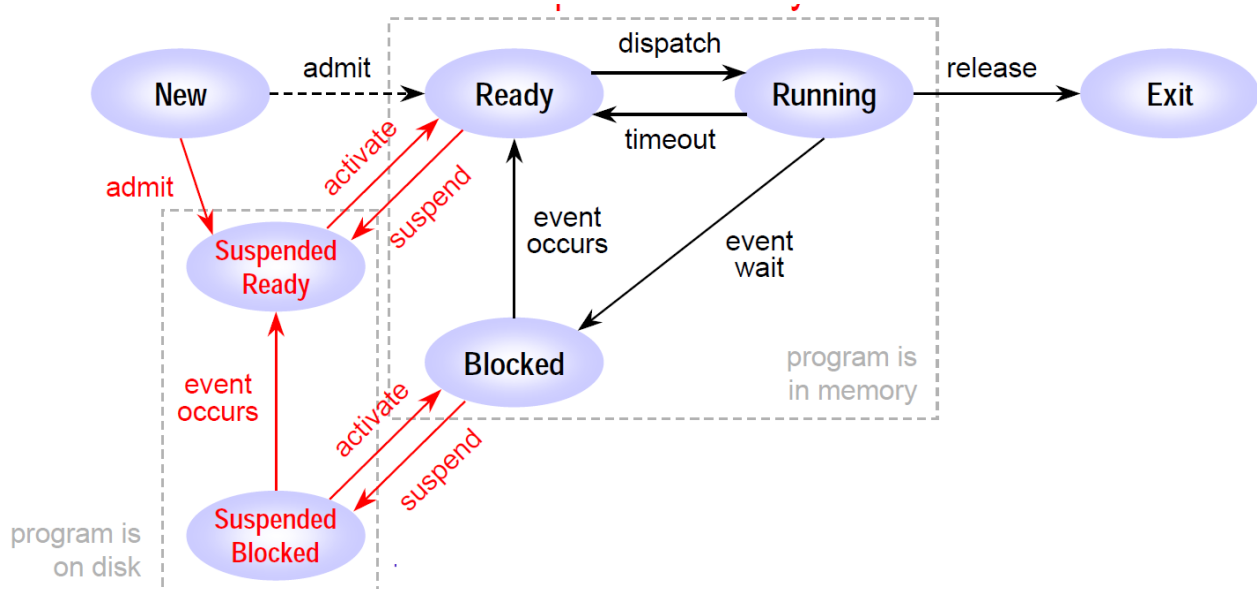
۱- اگر یک فرآیند در انتظار رخ دادن یک Event را به دلیل کمبود حافظه بخواهیم از حافظه خارج کنیم، این مدل جوابگو نیست.

۲- گاهی اوقات مجبوریم به دلیل کمبود حافظه حتی یک فرآیند آماده اجرا را نیز از حافظه اصلی خارج کنیم.

بنابراین دو حالت جدید تعریف کرده و مدل هفت حالت زیر را معرفی مینماییم.

حالت مسدود-معلق: در حافظه جانبی و منتظر یک event است.

حالت آماده-معلق: در حافظه جانبی است و به محض بار شدن در حافظه اصلی آماده اجراست.



هنگامی که هیچ‌یک از فرآیندهای موجود در حافظه‌ی اصلی در حالت آماده نیستند، و با کمبود فضای حافظه مواجه هستیم، سیستم‌عامل می‌تواند، فرآیند را از حافظه‌ی اصلی خارج کرده و به Disk منتقل کند. این عمل

Swap Out نام دارد. در صورتی که فضای کافی در حافظه مجدداً در دسترس باشد، فرآیند معلق، دوباره به حافظه برگردانده می‌شود. (Swap In).

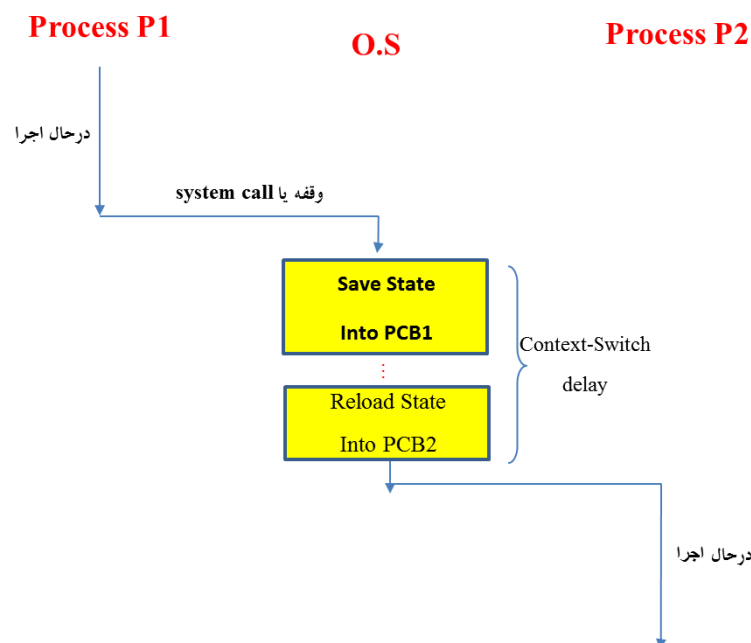
PCB (Process Control Block): یک ساختمان داده است که حاوی تمام اطلاعات مورد نیاز سیستم‌عامل درمورد یک فرآیند می‌باشد. شامل Fieldهای مختلفی مانند اطلاعات زیر است:

- ✓ شناسه فرآیند: شناسه فرآیند، شناسه والد، شناسه کاربر
- ✓ اطلاعات وضعیت فرآیند: مقادیر رجیسترها، PC (برای اینکه پس از شروع مجدد هر فرآیند، سیستم‌عامل بداند که از کجا ادامه دهد)، اشاره‌گرهای حافظه
- ✓ اطلاعات کنترل فرآیند: حالت فرآیند، اولویت، اطلاعات و وضعیت I/O، اطلاعات حسابداری (تا به حال چقدر از منابع سیستم استفاده کرده است؟)

Identifier
State
Priority
Program counter
Memory pointers
Context data
I/O status information
Accounting information
⋮

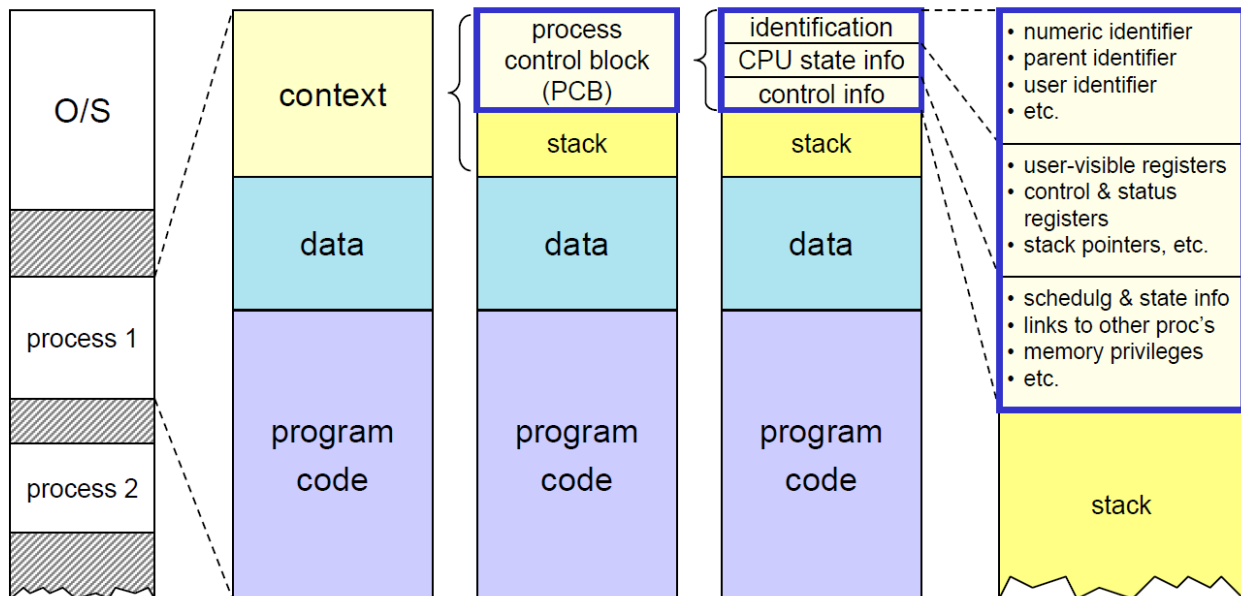
از دید سیستم‌عامل، PCB، نماینده یک فرآیند است. با ایجاد یک فرآیند، برای آن PCB ساخته می‌شود. وقتی فرآیند به اتمام رسید، PCB مربوط به آن حذف می‌شود و منابع در اختیار گرفته شده توسط فرآیند، آزاد می‌شوند.

Context Switch یا تعویض متن: این عمل جزء سربار به حساب می‌آید.



زمان تعویض فرآیند: تعویض فرآیند ممکن است در هر زمانی که سیستم عامل کنترل را از فرآیند در حال اجرای فعلی گرفته است انجام شود

Example of process and PCB location in memory



➔ The PCB is the most important O/S data structure

رویدادهایی که باعث می‌شوند سیستم عامل تعویض متن انجام دهد:

راهکار	دلیل	استفاده
وقفه	خارج از اجرای دستورالعمل جاری	واکنش به یک رخداد خارجی ناهمگام
تله	مربوط به اجرای دستورالعمل جاری	اداره کردن یک خطا یا شرایط استثنا
فراخوانی سیستمی system call	درخواست صریح	فراخوانی یک تابع سیستم عامل

تقسیم بندی فرآیندها از نظر استقلال

- ✓ فرآیندهای مستقل (Independent): سایر فرآیندها روی چنین فرآیندهایی تأثیر نمی‌گذارند.
- ✓ فرآیند مشارکت کننده (Cooperative): چنین فرآیندهایی یا باید با هم هماهنگ باشند، یا به ترتیب خاصی اجرا شوند.

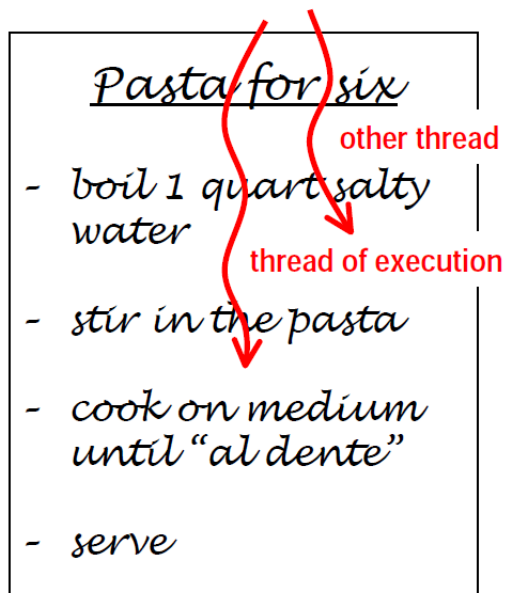
نخ (Thread)

مفهوم فرآیند از آنچه تا به حال ارائه شد، پیچیده‌تر و ظریف‌تر است و در واقع شامل دو مفهوم جداگانه و بالقوه مستقل است. یک مفهوم مربوط به مالکیت منبع و مفهوم دیگر مربوط به اجراست. این تمایز منجر به توسعه ساختاری به نام نخ (thread) در بسیاری از سیستم‌های عامل شده است.

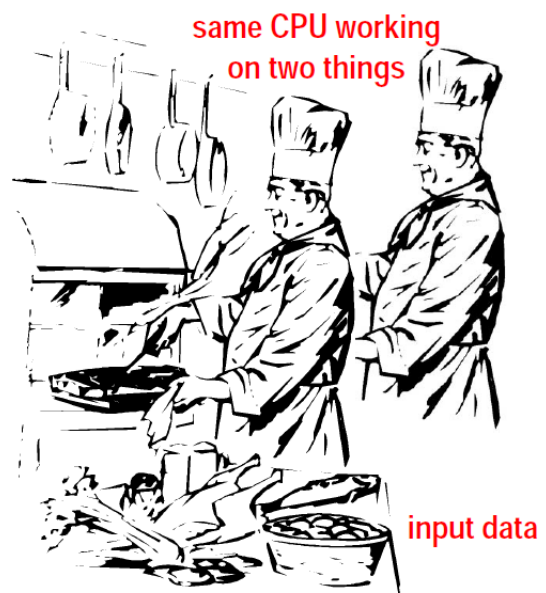
تا کنون مفهوم فرآیند را با دو ویژگی ارائه کرده ایم:

- ✓ مالکیت منبع: فرآیند شامل یک فضای آدرس مجازی است که تصویر فرآیند را نگه می‌دارد.
- تصویر فرآیند شامل مجموعه برنامه، داده، پشته و خصیصه‌هایی که در بلوک کنترل فرآیند تعریف شده‌است، می‌باشد.
- ✓ زمانبندی/اجرا: اجرای یک فرآیند یک مسیر (رد) اجرا را در یک یا چند برنامه دنبال می‌کند. ممکن است در بین اجرا، فرآیندهای دیگری نیز اجرا شوند. بنابراین فرآیند دارای یک حالت اجرا و یک اولویت توزیع است و موجودیتی است که توسط سیستم عامل زمانبندی و توزیع می‌شود.

The execution part is a "thread" that can be multiplied



Program



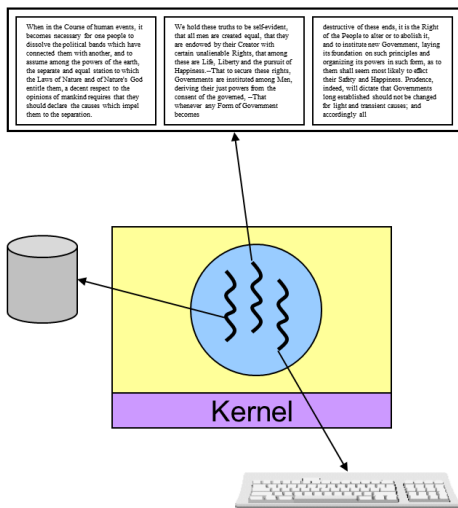
Process

به نخ یک فرآیند سبک وزن هم گفته می‌شود. یک فرآیند می‌تواند شامل چند نخ باشد. یک فرآیند می‌تواند نقاط اجرای متعددی داشته باشد که بالقوه توسط چند پردازنده قابل اجرا باشد. در این حالت وقتی یکی از

نخها به دلیلی متوقف شد، فرآیند نخ دیگری را فعال کند. تعویض نخها بسیار سریعتر از تعویض فرآیندهاست.

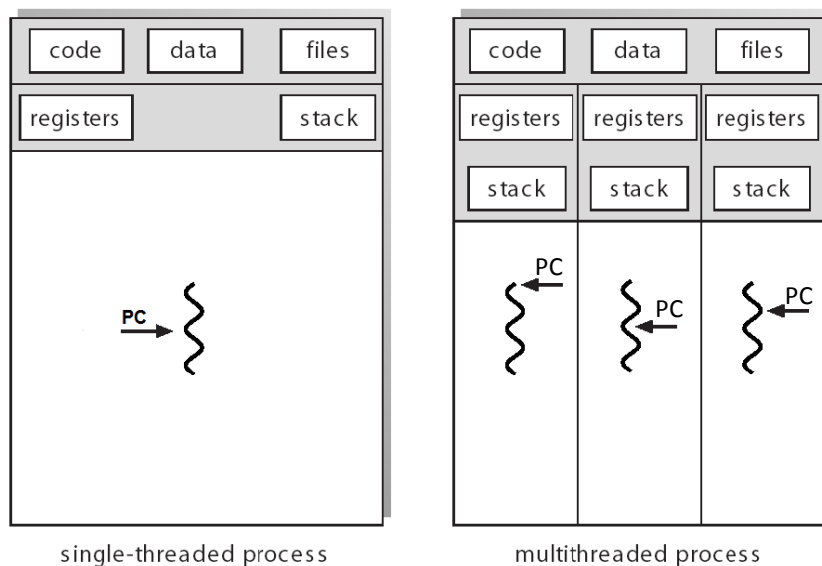
به عنوان مثال اگر آشپزی را به صورت یک فرآیند در نظر بگیریم که شامل مراحل مختلفی است، تا زمانی که آب به جوش می آید می توان به کار دیگری (مثلا خرد کردن پیاز یا سرخ کردن سیب زمینی) پرداخت. مثال

دیگر کاربرد نخ در ویرایشگرهای متن است که یک نخ می تواند مسئول خواندن از بافر صفحه کلید باشد، نخ دیگر فرمت بندی متن را انجام دهد و نخ دیگر مسئول نوشتن در دیسک باشد.



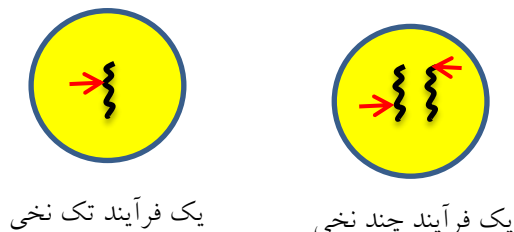
هر Thread دارای PC، مجموعه رجیسترها و فضای استک مربوط به خود می باشد. سایر قسمتهای یک فرآیند توسط تمام نخهای آن فرآیند به صورت اشتراکی استفاده می گردد.

ختم یک فرآیند منجر به ختم تمام نخهای آن خواهد شد.

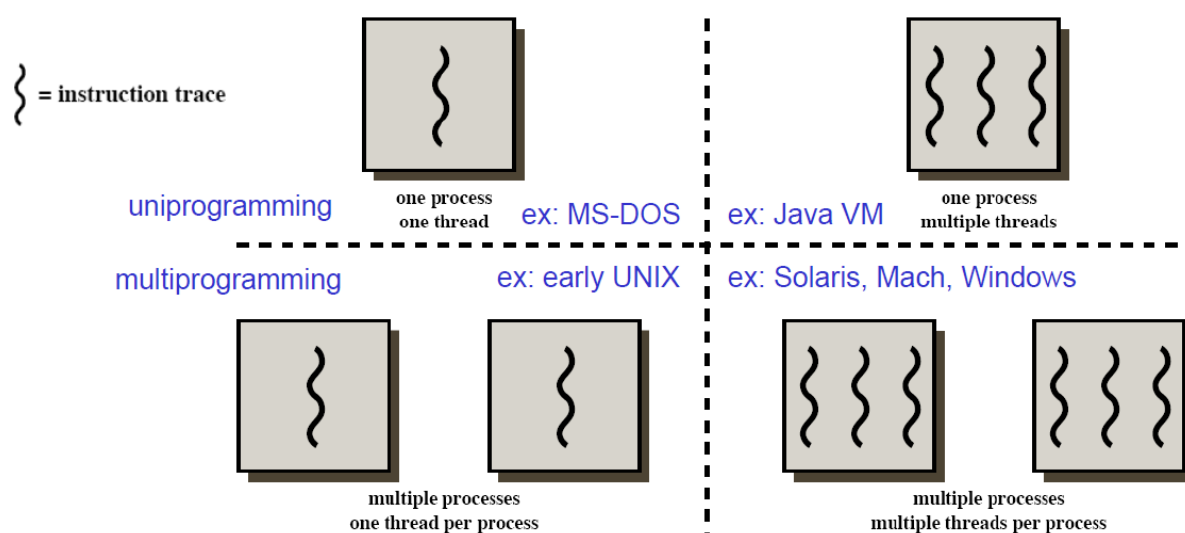


چند نخ

چند نخ (Multithreading) به توانایی سیستم عامل در پشتیبانی از نخ های اجرای متعدد در یک فرآیند واحد اطلاق می شود. رویکرد سنتی که در آن یک نخ اجرای واحد در هر پردازش وجود دارد (و مفهوم نخ در آن وجود ندارد) رویکرد تک نخ نامیده می شود.



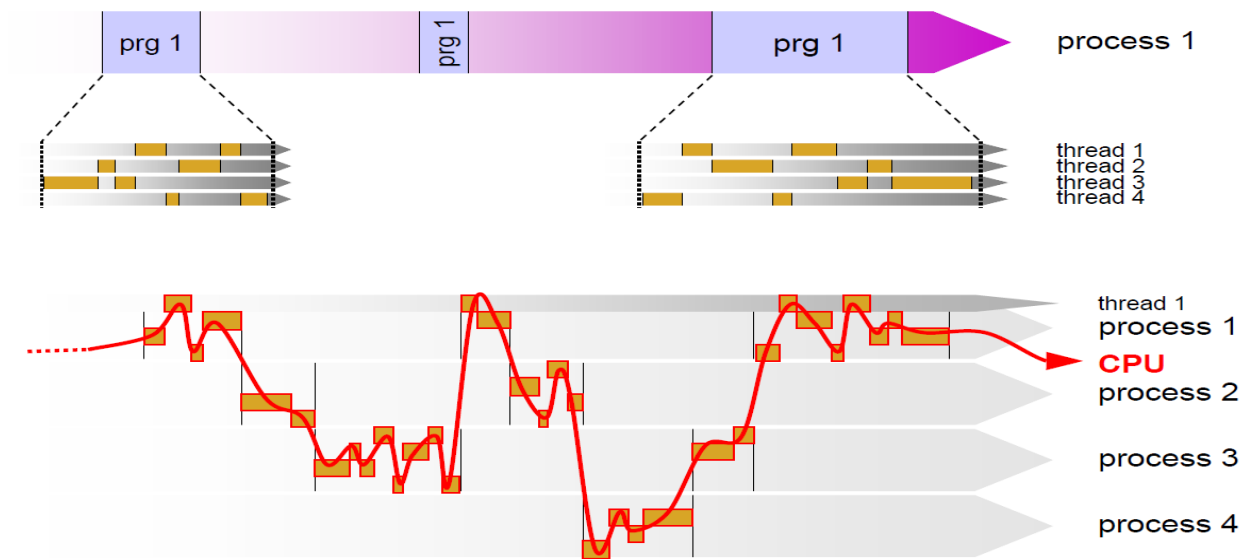
یک فرآیند تک نخ را می توان به چند نخ تبدیل کرد. نخ به تنهایی موجودیت ندارد و سیستم عامل نمی تواند یک نخ تولید کند.



مثال ها:

- ✓ MS-DOS سیستم عاملی است که از یک فرآیند کاربر و از نخ واحد پشتیبانی می کند.
- ✓ بسیاری از گونه های یونیکس، از فرآیندهای کاربر متعدد پشتیبانی می کنند، اما فقط با یک نخ در هر فرآیند این کار را انجام می دهد.
- ✓ محیط زمان اجرای جاوا (Java Runtime Environment)، نمونه ای از سیستمی با یک فرآیند چندنخی است.
- ✓ چندین فرآیند که هریک از چندین نخ پشتیبانی می کنند در ویندوز، Solaris و Match استفاده می شود.

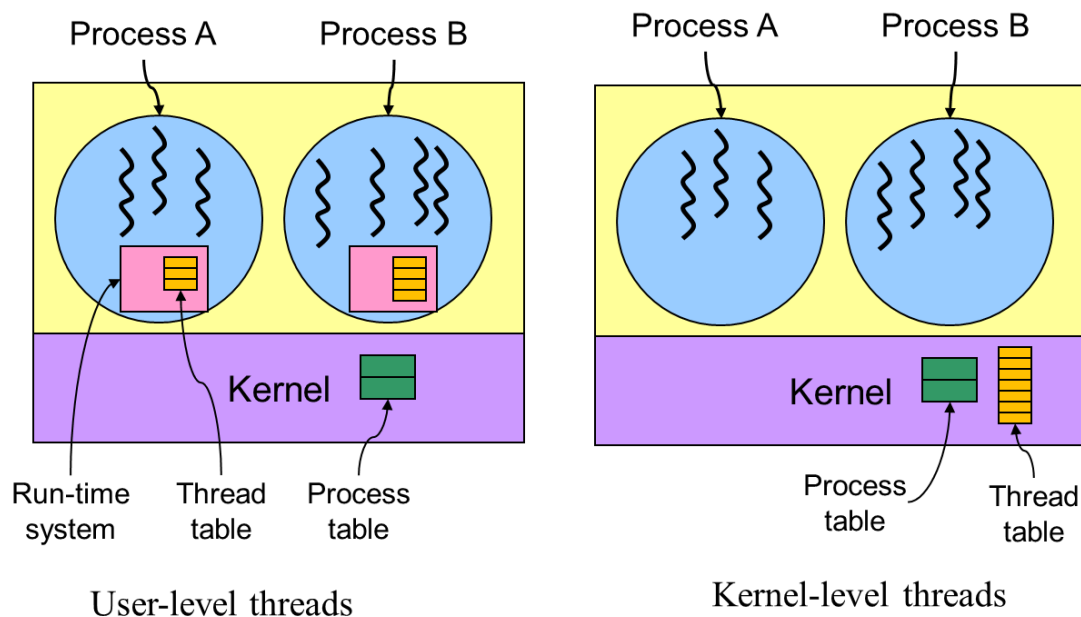
شکل زیر نحوه پردازش چهار فرآیند چندنخی توسط یک سیستم تک پردازنده‌ای را نشان می‌دهد.



نخ‌ها را می‌توان به دو دسته تقسیم نمود:

نخ سطح کاربر (user thread): مدیریت نخ‌ها و ایجاد آن به عهده کاربر است و هسته سیستم عامل از وجود آن‌ها باخبر نیست.

نخ هسته (kernel thread): هسته مسئول ایجاد زمانبندی و مدیریت نخ‌ها است. زمانبندی بین فرآیندها انجام نمی‌شود بلکه بین نخ‌ها انجام می‌گیرد.



Context switch برای نخ کاربر امکان‌پذیر نیست ولی برای نخ kernel امکان‌پذیر است.

مثال: A1,A2,A3,A1,B1,B3,B2,B3 در هر دو فرآیند امکان‌پذیر است.

A1,A2,B1,B2,A3,B3,A2,B1 فقط در حالت kernel امکان‌پذیر است.

استفاده از نخ‌های سطح کاربر به جای نخ‌های سطح هسته مزایایی دارد که برخی از آنها عبارتند از:

۱- از آنجا که همه ساختمان داده‌های مدیریت نخ در فضای آدرس کاربر یک فرآیند واحد قرار دارند، تعویض نخ نیازی به امتیازات حالت هسته ندارد. بنابراین حالت فرایند برای مدیریت نخ، به حالت هسته تغییر نمی‌کند.

۲- زمانبندی می‌تواند خاص-کاربرد باشد. الگوریتم زمانبندی نوبتی می‌تواند برای یک کاربرد مفیدتر باشد و در عین حال کاربرد دیگری ممکن است از یک الگوریتم زمانبندی مبتنی بر اولویت استفاده کند. می‌توان بدون این که در زمانبند سیستم عامل خللی ایجاد شود، الگوریتم زمانبندی مناسب را به کار گرفت.

۳- نخ‌های سطح کاربر روی هر سیستم عاملی اجرا می‌شوند. برای پشتیبانی از نخ‌های سطح کاربر نیازی به تغییر هسته نیست. کتابخانه نخ‌ها مجموعه‌ای از برنامه‌های سودمند در سطح کاربرد است که در همه کاربردها مشترک است.

دو عیب آشکار نخ‌های سطح کاربر:

۱- در یک سیستم عامل متداول، بسیاری از فراخوانی‌های سیستم مسدودکننده (Blocking) هستند. در نتیجه وقتی که یک نخ سطح کاربر یک فراخوانی سیستمی را اجرا می‌کند، نه تنها آن نخ، بلکه همه نخ‌های آن فرآیند مسدود می‌شوند.

۲- در یک راهبرد ULT محض، یک کاربرد چند نخی نمی‌تواند از مزایای چند پردازشی استفاده نماید. هسته در هر زمان هر فرآیند را فقط به یک پردازنده اختصاص می‌دهد. بنابراین در یک زمان فقط یک نخ از هر فرآیند می‌تواند اجرا شود.

حالت‌های عملکرد نخ

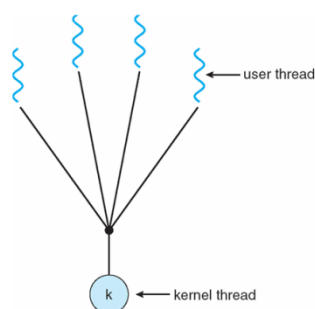
نخ‌ها مانند فرآیندها دارای حالات اجرا هستند و ممکن است با یکدیگر همگام شوند. این دو جنبه از عملکرد نخ را به ترتیب بررسی می‌کنیم. حالات اصلی نخ مثل فرآیندها عبارتند از اجرا، آماده و مسدود. به طور کلی، منظور کردن حالات معلق برای نخ‌ها بی‌معنی است؛ زیرا این حالات، مفاهیم سطح فرآیند می‌باشند. به ویژه اگر فرآیند به خارج از حافظه اصلی مبادله شود، از آنجا که همه نخ‌های آن فرآیند از فضای آدرس اشتراکی استفاده می‌کنند، همه‌ی نخ‌ها لزوماً به خارج مبادله می‌شوند.

مدلهای چند نخی:

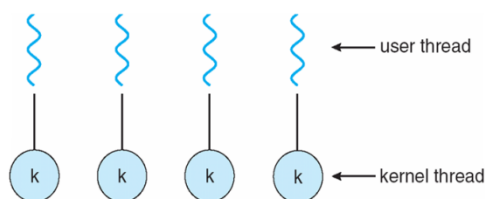
• مدل چند به یک (Many-to-One): در این مدل تعداد زیادی نخ سطح کاربر به یک نخ سطح هسته نگاشت می‌شوند.

• مدل یک به یک (One-to-One): در این مدل هر نخ سطح کاربر به یک نخ سطح هسته نگاشت می‌شود. مانند Windows NT/XP/2000 و Linux

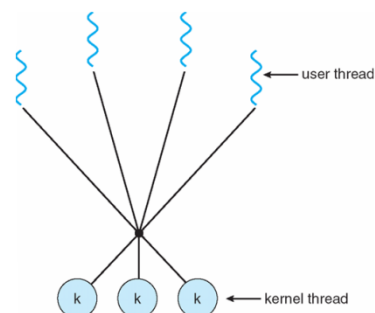
• مدل چند به چند (Many-to-Many): این مدل اجازه می‌دهد چندین نخ سطح کاربر به چندین نخ سطح هسته نگاشت شوند. مانند Solaris



Many-to-One



One-to-One



Many-to-Many Model

۱

۲

۳

۴

۵

۶

۷

۸

فصل سوم

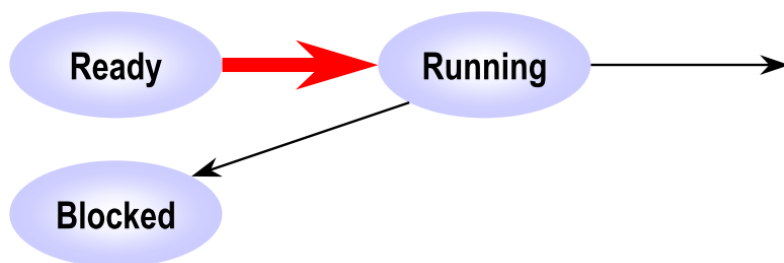
زمان‌بندی فرایندها

در این فصل به مبحث زمان‌بندی فرایندها در یک سیستم تک‌پردازنده ای خواهیم پرداخت. فرآیندهایی که در صف آماده قرار گرفته اند، لزوماً به ترتیب سرویس دهی نمی‌شوند. تصمیم در مورد اینکه کدام یک از فرآیندهای موجود در صف آماده برای اجرا گرفتن CPU انتخاب شوند مورد بحث ماست. الگوریتم‌های مختلفی برای این کار وجود دارند.

حالت‌های تصمیم‌گیری

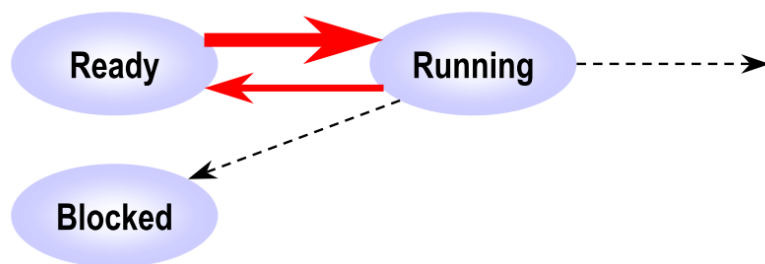
۱- بدون قبضه کردن (non-preemptive) ← انحصاری

فرآیند در حال اجرا، آنقدر به اجرا ادامه می‌دهد تا خاتمه یابد، یا اینکه خودش برای انتظار I/O یا درخواست خدمتی از سیستم‌عامل، مسدود گردد.



۲- با قبضه کردن (preemptive) ← غیر انحصاری

سیستم‌عامل می‌تواند، فرآیند در حال اجرا را متوقف کند (CPU را از آن بگیرد) و به حالت آماده منتقل کند.



اهداف الگوریتم‌ها

۱- عادل باشد

۲- سازگار بودن با سیاست‌های سیستم (فرآیندهای الویت دار به بدون اولویت ترجیح داده می‌شود)

۳- مشغول نگه داشتن منابع سیستم (CPU و دستگاههای I/O)

معیارها از دیدگاه کاربر

- ✓ زمان پاسخ: (response time) فاصله زمانی ارائه تقاضا تا شروع دریافت یک پاسخ.
 - ✓ زمان کل: (turnaround time) فاصله زمانی بین پذیرش یک فرآیند تا تکمیل آن.
 - ✓ زمان انتظار: (waiting time) مدت زمانی که یک فرآیند در صف آماده، منتظر اجرا شدن می ماند.
- نکته: یک الگوریتم زمانبندی مناسب، الگوریتمی است که زمان پاسخ و زمان کل و زمان انتظار minimize باشد.

معیارها از دیدگاه سیستم

- ✓ توان عملیاتی: (throughput) تعداد فرآیندهای تکمیل شده در واحد زمان.
- ✓ بارزدهی پردازنده: (CPU utilization) درصد زمانی که پردازنده مشغول می باشد.

الگوریتم های زمان بندی

۱- (First Come First Served)FCFS

به فرآیندها به ترتیبی که درخواست داده اند CPU اختصاص می یابد. اگر یک فرآیند بلوکه شود اولین فرآیند در صف آماده، اجرا می شود. هرگاه یک فرآیند از حالت بلوکه به حالت آماده برود به انتهای صف آماده فرستاده می شود.

مزیت: ساده بودن و عملی بودن (سهولت پیاده سازی)

معایب: - انحصاری بودن

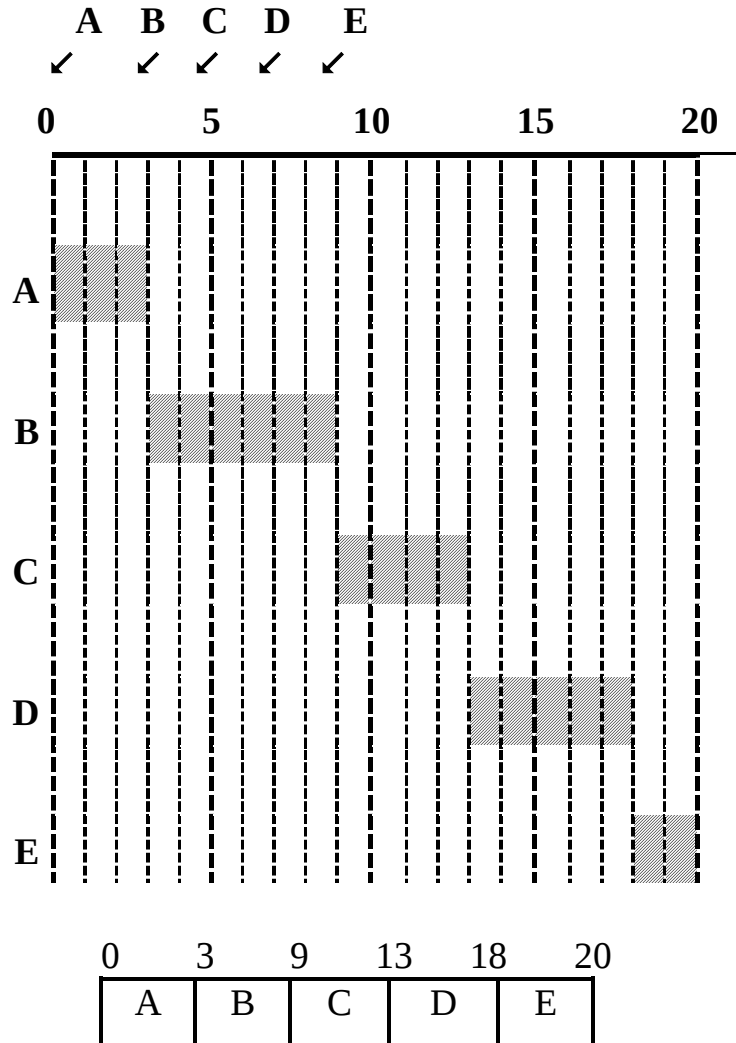
- بالا بودن زمان انتظار

- ممکن است یک فرآیند کوتاه زمان زیادی منتظر بماند

- امکان استفاده در سیستم های اشتراک زمانی وجود ندارد

مثال ۱ :

Process	Arrival Time	Service Time
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2



	A	B	C	D	E
زمان ورود	0	2	4	6	8
زمان خروج	3	9	13	18	20

$$\Rightarrow \text{میانگین زمان کل} = \frac{(3-0) + (9-2) + (13-4) + (18-6) + (20-8)}{5} = 8.6$$

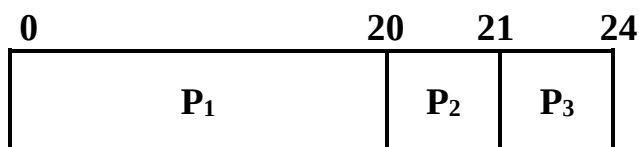
$$\Rightarrow \text{میانگین زمان انتظار} = \frac{0+1+5+7+10}{5} = 4.6$$

مثال ۲: فرآیندهای زیر در صف قرار گرفته اند. متوسط زمان انتظار را برای ترتیب‌های زیر بر اساس الگوریتم FCFS پیدا کنید (فرض: زمان ورود هر سه لحظه صفر باشد)

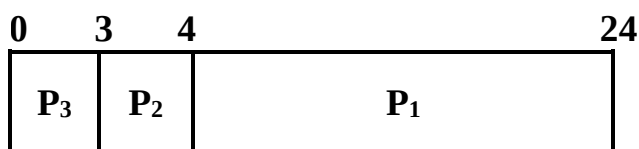
فرآیند	زمان مورد نیاز برای اجرا
P ₁	20
P ₂	1
P ₃	3

P₃ P₂ P₁

P₁ P₂ P₃



$$\Rightarrow AWT = \frac{0 + 20 + 21}{3} = 13.67$$



$$\Rightarrow AWT = \frac{4 + 3 + 0}{3} = 2.33$$

متوسط زمان انتظار به ترتیب اجرای فرآیندها بستگی دارد.



Short CPU burst



Short CPU burst



Long CPU burst

۲- SPN (Shortest Process Next) Algorithm

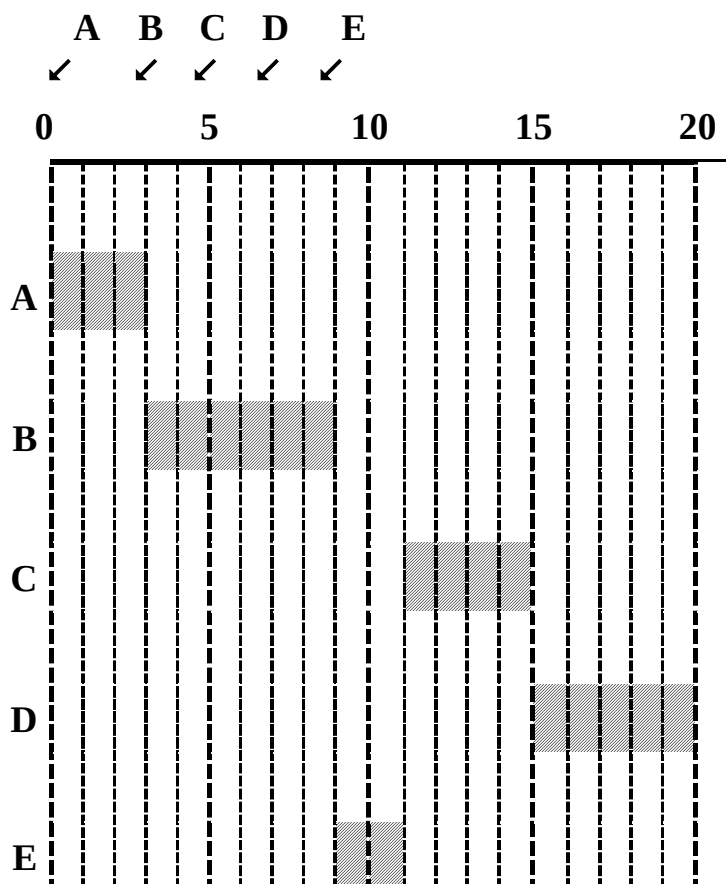
یک راه برای بهبود رفتار FCFS، برای فرآیندهای کوتاه مدت است. در این الگوریتم فرآیندی که کوتاهترین زمان پردازش را دارد از صف فرآیندهای آماده انتخاب می شود.

معایب:

- ✓ نیاز داشتن اطلاعات زمانی قبل از شروع اجرا
- ✓ احتمال به تعویق افتادن کارهای طولانی (گرسنگی)
- ✓ انقطاع ناپذیر بودن (انحصاری)

مثال:

Process	Arrival Time	Service Time
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2



0	3	9	11	15	20
A	B	E	C	D	

PROCESS	A	B	C	D	E
---------	---	---	---	---	---

زمان ورود

0 2 4 6 8

$$\text{میانگین زمان کل} = \frac{3+7+11+14+3}{5} = 7.6$$

زمان خروج

3 9 15 20 11

$$\Rightarrow \text{میانگین زمان انتظار} = \frac{0+1+7+9+1}{5} = 3.6$$

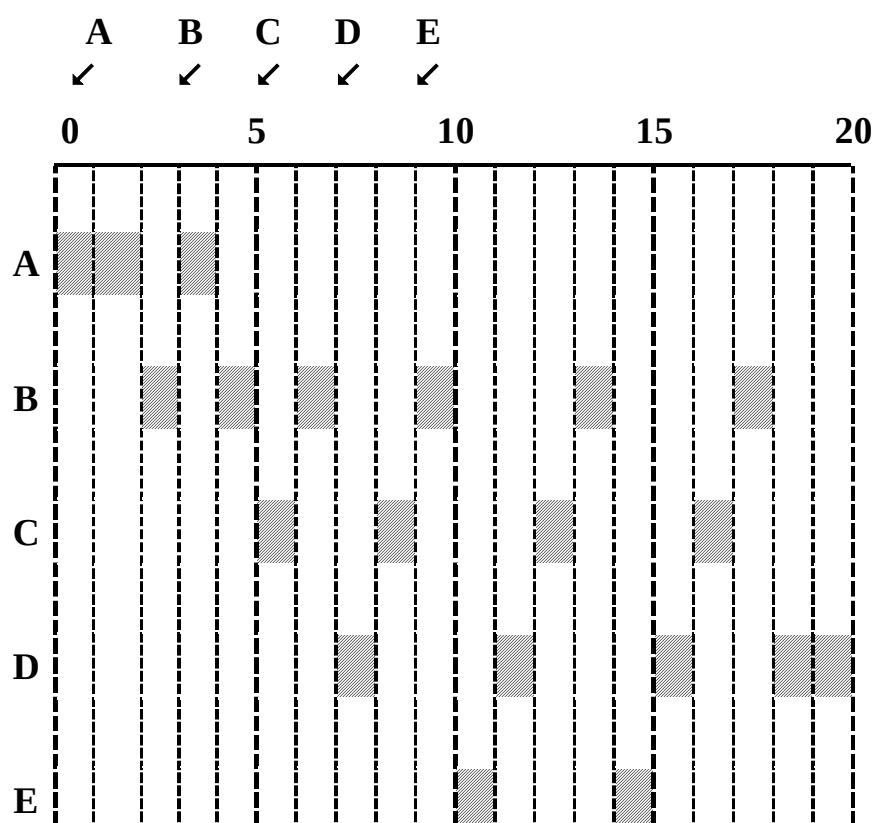
۳- الگوریتم نوبتی گردشی (Round Robin):

یک الگوریتم غیرانحصاری بوده و براساس بازه‌های زمانی (کوانتوم) کار می‌کند. در این الگوریتم صف فرآیندهای آماده به صورت حلقوی در نظر گرفته شده و به هر فرآیند، حداکثر به مدت یک بازه زمانی کوتاه (برش زمانی) پردازنده اختصاص می‌یابد.

اگر زمان اجرای فرآیند کمتر از برش زمانی باشد، آنگاه فرآیند بعد از اتمام زمان اجرا پردازنده را آزاد می‌کند. اگر زمان اجرای فرآیند بیشتر از برش زمانی باشد، آنگاه فرآیند بعد از اتمام برش زمانی، به وسیله وقفه پردازنده را آزاد کرده و به انتهای صف فرآیندهای آماده می‌رود و فرآیند بعدی از ابتدای صف انتخاب خواهد شد.

نکته: مساله مهم در این الگوریتم، طول برش زمانی است. اگر برش زمانی طولانی انتخاب شود، رفتار آن شبیه الگوریتم FCFS می‌شود. اگر برش زمانی کوتاه انتخاب شود، درصد بازدهی کم می‌شود ← زمان زیادی صرف تعویض متن می‌شود.

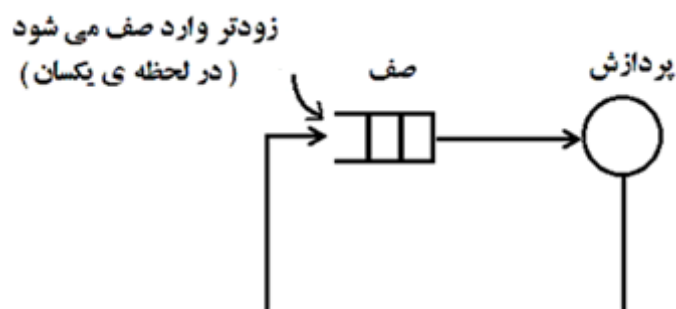
مثال: کوانتوم ← $q=1$



زمان پاسخ: A:0 B:0 C:1 D:1 E:2

	A ↓		B ↓		C ↓		D ↓		E ↓	
لحظه	0	1	2	3	4	5	6	7	8	
صف			A	B	C	B	C D	B C	D E B	
پردازش	A	A	B	A	B	C	B	D	C	

نکته:



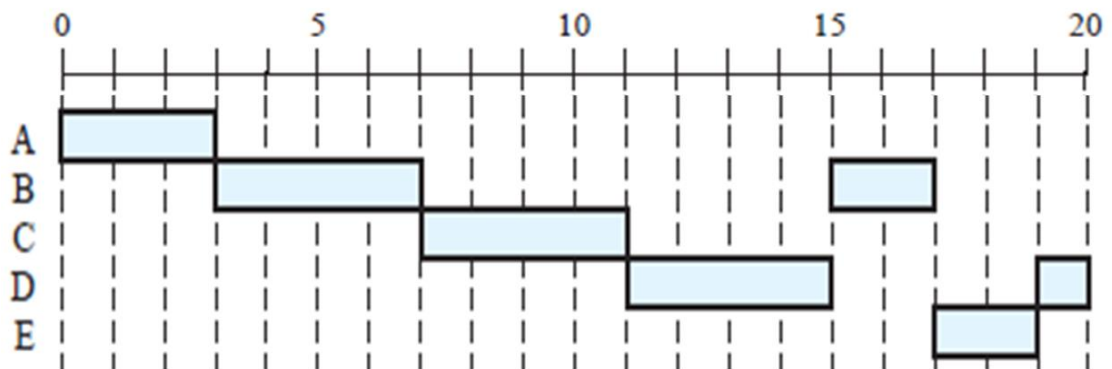
A ↓		B ↓		C ↓		D ↓		E ↓		
0	1	2	3	4	5	6	7	8	9	10
A(2)	A(1)	B(5)	A(0)	B(4)	C(3)	B(3)	D(4)	C(2)	B(2)	
10	11	12	13	14	15	16	17	18	19	20
E(1)	D(3)	C(1)	B(1)	E(0)	D(2)	C(0)	B(0)	D(1)	D(0)	

$$\text{میانگین زمان کل} = \frac{(4-0) + (18-2) + (17-4) + (20-6) + (15-8)}{5} = 10.8$$

$$\text{میانگین زمان انتظار} = \frac{1+10+9+9+5}{5} = 6.6$$

مثال: کوانتوم $q=4$ ←

زمان پاسخ: A:0 B:1 C:3 D:5 E:9



$$\text{میانگین زمان کل} = \frac{(3-0) + (17-2) + (11-4) + (20-6) + (19-8)}{5} = 10$$

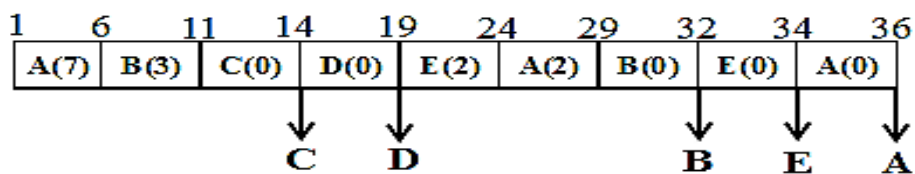
$$\text{میانگین زمان انتظار} = \frac{0+9+3+9+9}{5} = 6$$

زمان پاسخ: از لحظه‌ای که فرآیند وارد می‌شود تا لحظه‌ای که اولین بار نوبتش می‌شود.

هر چه کوانتوم کمتر باشد ← زمان پاسخ نیز کمتر است.

مثال: متوسط زمان کل براساس الگوریتم RR با برش زمانی 5 را محاسبه کنید.

نام فرآیند	زمان ورود	مدت زمان اجرا
A	1	12
B	2	8
C	3	3
D	4	5
E	5	7



$$\text{متوسط زمان کل} = \frac{(36-1) + (32-2) + (14-3) + (19-4) + (34-5)}{5} = 24$$

۴- الگوریتم SRT (Shortest Remaining Time)

این الگوریتم یک نوع SPN غیرانحصاری است و همواره فرآیندی را انتخاب می‌کند که کوتاه‌ترین زمان پردازش باقی مانده را داشته باشد. هنگامی که فرآیند جدیدی به صف آماده ملحق می‌گردد، اگر زمان باقیمانده پردازش فرآیند در حال اجرا، از زمان مورد نیاز فرآیند تازه وارد بیشتر باشد، فرآیند جاری معوق می‌گردد و به صف فرآیندهای آماده منتقل می‌گردد و پردازنده در اختیار فرآیند تازه وارد قرار می‌گیرد.

مزیت: در مقایسه با SPN معمولاً زمان کل بهتری ارائه می‌کند.

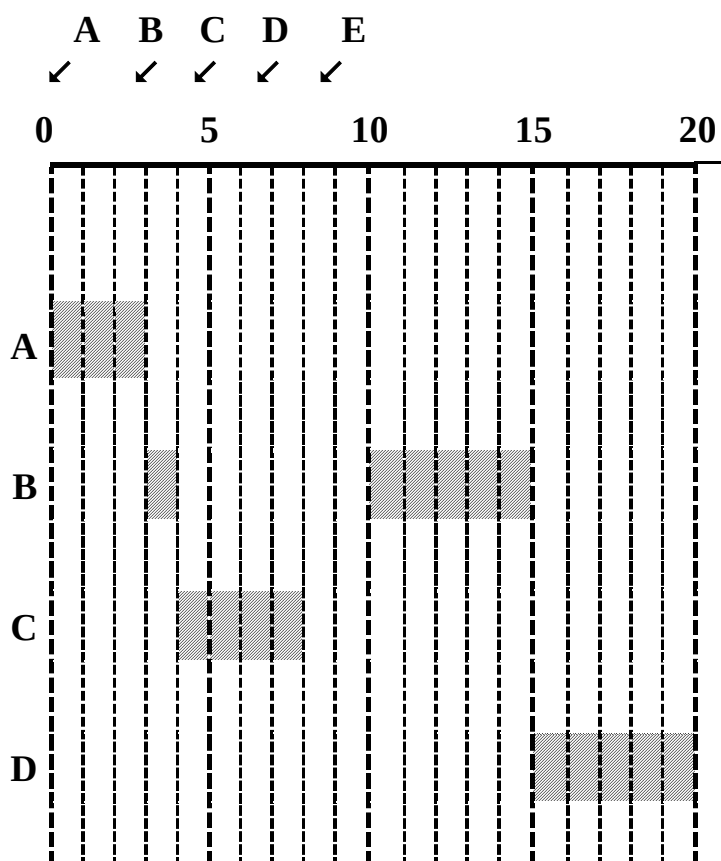
معایب: - نیاز به دانستن اطلاعات مقاطع زمانی قبل از اجرا.

- احتمال به تعویق افتادن کارهای طولانی (گرسنگی)

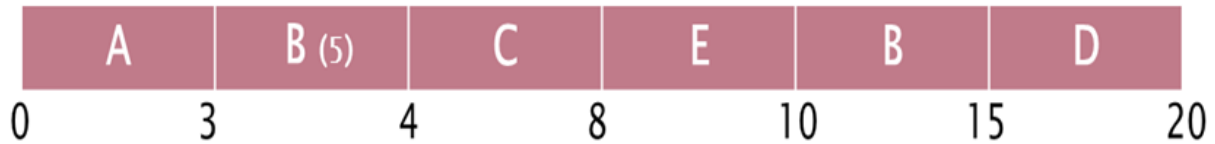
- نسبت به SPN سربار بیشتری دارد.

مثال:

فرآیند	زمان مورد نیاز
A	3
B	6
C	4
D	5
E	2



	Process	A	B	C	D	E	
	Arrival Time	0	2	4	6	8	
	Service Time (T_s)	3	6	4	5	2	Mean
SRT	Finish Time	3	15	8	20	10	
	Turnaround Time (T_T)	3	13	4	14	2	7.20
	T_T/T_s	1.00	2.17	1.00	2.80	1.00	1.59



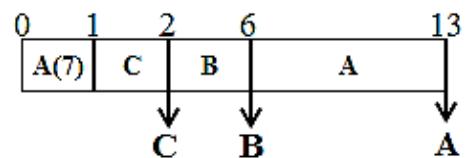
$$\text{میانگین زمان کل} = \frac{(3-0) + (15-2) + (8-4) + (20-6) + (10-8)}{5} = 7.2$$

$$\text{میانگین زمان انتظار} = \frac{0+7+0+9+0}{5} = 3.2$$

✓ همان طور که قبلاً ذکر شد، میانگین زمان انتظار و میانگین زمان کل کمتری نسبت به SPN دارد.

مثال: با توجه به جدول زیر، میانگین زمان کل و انتظار را براساس الگوریتم SRT محاسبه نمایید.

نام فرآیند	زمان ورود	مدت زمان اجرا
A	0	8
B	2	4
C	1	1



$$\text{میانگین زمان کل} = \frac{13+4+1}{3} = 6$$

$$\text{میانگین زمان انتظار} = \frac{5+0+0}{3} = 1.6$$

۵- الگوریتم HRRN (Highest Response Ratio Next)

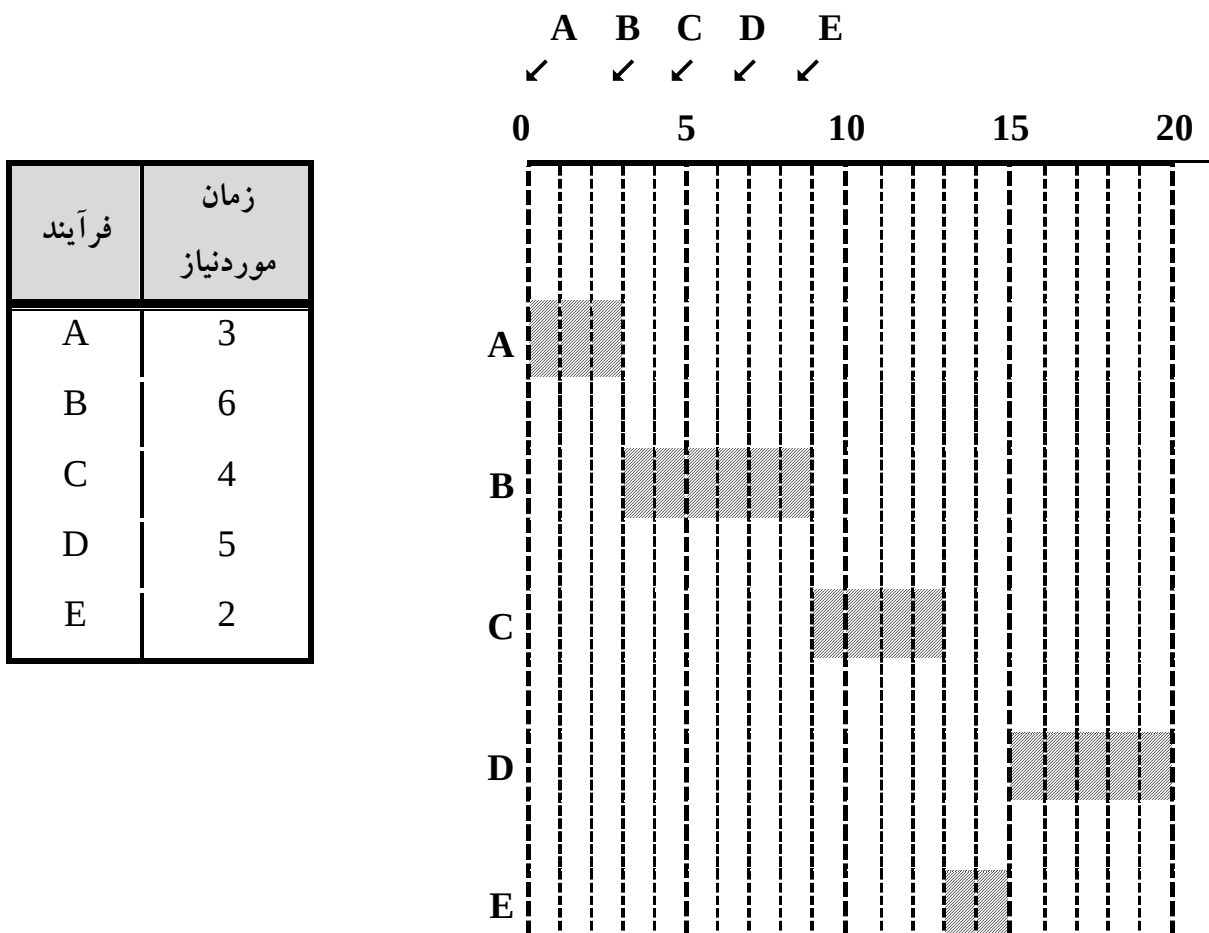
یک الگوریتم زمان بندی انحصاری است که در آن اولویت هر فرآیند، تابعی است از مدت زمان اجرا (زمان سرویس آن) و مدت زمانی که برای بدست آوردن پردازنده منتظر بوده است. بنابراین فرآیندهای کوتاه تر اولویت بیشتری دارند اما فرآیندهای طولانی تر بعد از مدتی که منتظر ماندند، مورد توجه قرار می گیرند.

$$\text{نسبت پاسخ} = \frac{\text{زمان سرویس (s)} + \text{زمان انتظار (w)}}{\text{زمان سرویس (s)}}$$

✓ هرکدام بیشتر بود برای اجرا انتخاب می شود.

✓ در این الگوریتم پدیده ی گرسنگی به وجود نخواهد آمد.

مثال:



در لحظه 0 فقط فرآیند A است، پس اجرا می‌گردد. در لحظه 3، بعد از اتمام اجرای A، فقط B وجود دارد. پس اجرا می‌گردد. اما در لحظه 9 بعد از اتمام اجرای B باید برای تمامی فرآیندهای تازه وارد، C, D, E، نسبت پاسخ محاسبه گردد.

$$RR(D) = \frac{3+5}{5} = 1.6 \quad RR(C) = \frac{5+4}{4} = 2.25 \quad \text{در لحظه 9:}$$

$$RR(E) = \frac{1+2}{2} = 1.5$$

✓ در لحظه ۹ فرآیند C دارای نسبت به پاسخ بیشتری است، بنابراین برای اجرا انتخاب می‌گردد.

✓ در لحظه ۱۳ بعد از اتمام فرآیند C، نسبت پاسخ برای D, E محاسبه می‌شود و فرآیند E برای اجرا انتخاب می‌گردد.

$$RR(E) = \frac{5+2}{2} = 3.5 \quad RR(D) = \frac{7+5}{5} = 2.4 \quad \text{در لحظه 13:}$$



$$\frac{3+7+9+14+7}{5} = \frac{40}{5} = 8$$

میانگین زمان کل

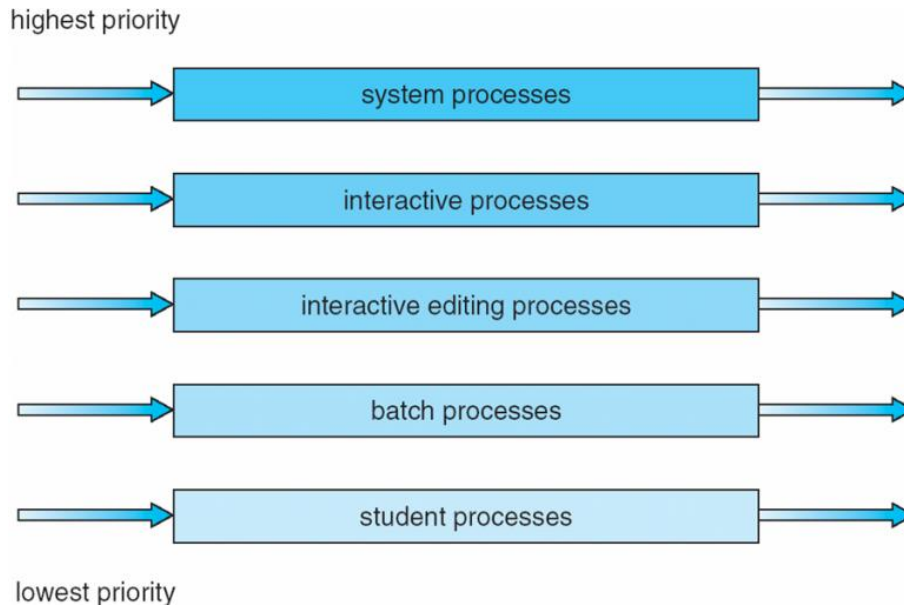
$$\frac{0+1+5+9+5}{5} = \frac{20}{5} = 4$$

میانگین زمان انتظار

۶- الگوریتم multi-level feedback queue scheduling

در این الگوریتم فرآیندها به سادگی به گروه‌های مختلف رده بندی می‌شوند، این الگوریتم صف فرآیندهای

آماده را به صف‌های مجزا تقسیم می‌کند. فرآیندها بر حسب ویژگی‌هایی در یک صف قرار می‌گیرند.

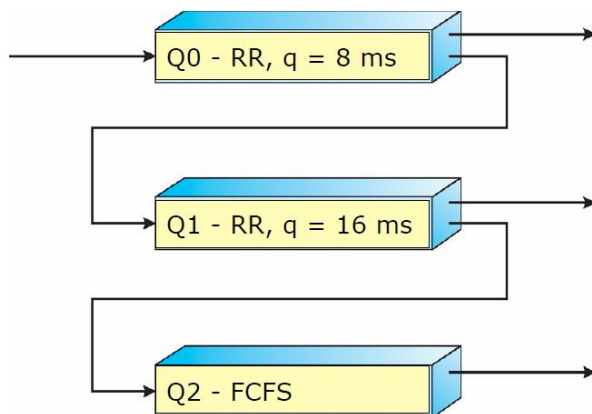


هر صف می‌تواند الگوریتم زمان‌بندی مخصوص به خود را داشته باشد. اگر فرآیندی پردازنده را مدت

زیادی به خدمت گیرد به صف با الویت کمتر منتقل می‌شود و اگر فرآیندی زمان طولانی در انتظار باشد به صف

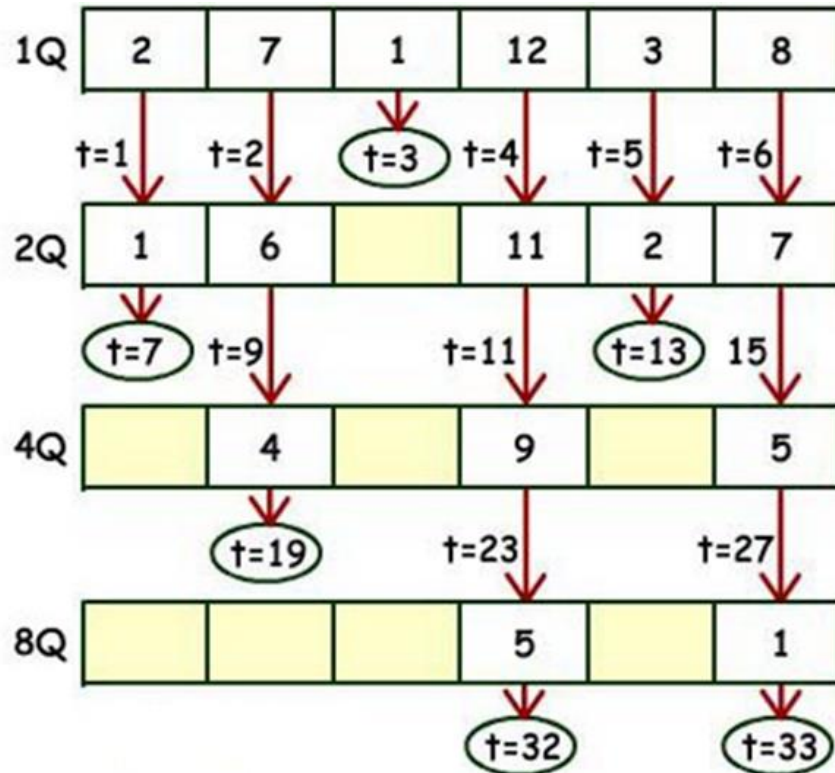
با الویت بیشتر منتقل می‌شود.

Multilevel Queue Scheduling



Multilevel feedback Queue Scheduling

مثال: 6 کار داریم که همگی در زمان صفر وارد شده اند و به ترتیب 2، 7، 1، 12، 3 و 8 واحد زمانی پردازش احتیاج دارند. با صف های چندگانه بازخورد که صف اول، یک واحد زمانی، صف دوم و بعدی هر کدام دو برابر صف قبلی CPU تخصیص داده می شود. میانگین زمان کل چند است؟



زمان کل = T خروج - T ورود

$$ATT = (3 + 7 + 13 + 19 + 32 + 33) / 6 = 17.83$$

۱

۲

۳

۴

۵

۶

۷

۸

فصل چهارم

همگام سازی فرآیندها

فرایند ها دو نوع اند: مستقل و مشارکت کننده

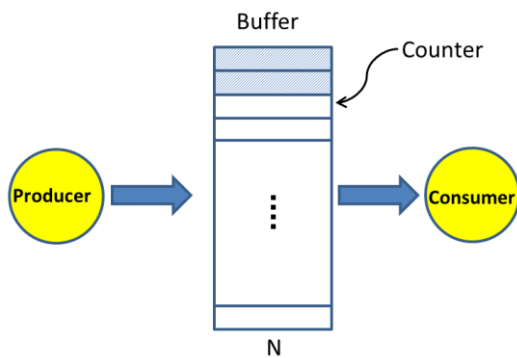
فرآیندهای مشارکت کننده (همکار) برای انجام همکاری نیاز به اشتراک و یا تبادل داده دارند.

مثال: کامپیوتر کاراکتر تولید می کند و پرینتر مصرف می کند.

دسترسی همروند به داده های مشترک ممکن است باعث ناسازگاری داده ای شود (فرآیندهای مستقل این مشکل را ندارند و به هر ترتیبی اجرا شوند خروجی نهایی یکسان است).

✓ **PRODUCER_CONSUMER PROBLEM** (مسأله تولید کننده/مصرف کننده)

اگر نرخ تولید و مصرف یکسان باشد به بافر احتیاج نیست اما چون در مسأله تولیدکننده/مصرف کننده ممکن است نرخ تولید و مصرف یکسان نباشد برای هماهنگ سازی بین تولید کننده و مصرف کننده از یک بافر استفاده می شود.



Producer Pseudo-code

Repeat

Produce an item;

while (Counter==N) do no_op;

Buffer[Counter]=item;

Counter=Counter+1;

Until False

Consumer Pseudo-code

Repeat

while (Counter==0) do no_op;

item=Buffer[Counter];

Counter=Counter-1;

Consume the item;

Until False

شبه کد تولید کننده

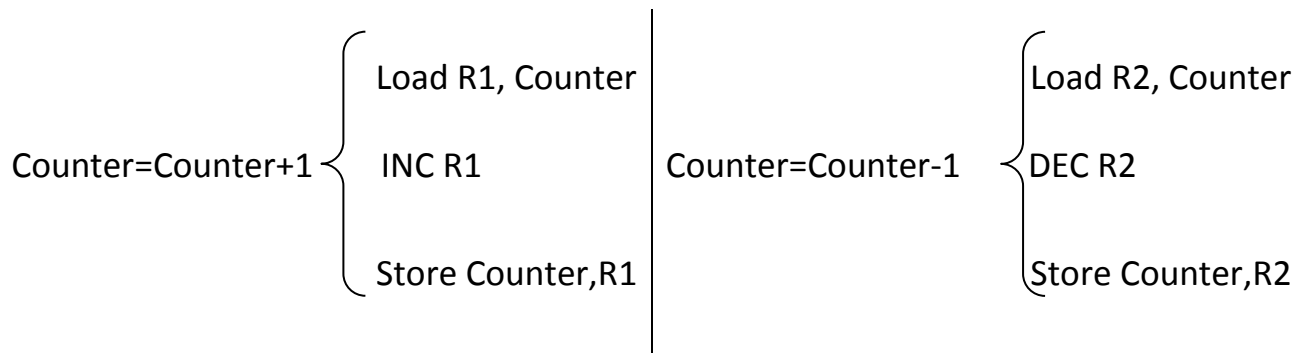
تولید کننده یک item تولید می کند. سپس بافر را بررسی می کند اگر بافر پر باشد هیچ کاری انجام نمی دهد در غیر این صورت item داخل $Buffer[Counter]$ گذاشته می شود و یک واحد به Counter اضافه می شود.

شبه کد مصرف کننده

اگر متغیر Counter برابر صفر باشد مصرف کننده هیچ کاری انجام نمی دهد. در غیر این صورت محتویات خانه Counter ام بافر داخل item گذاشته می شود. سپس یک واحد از Counter کم می شود. و item توسط مصرف کننده مصرف می شود.

فرض کنید مقدار فعلی $Counter=5$ باشد و تولید کننده دستور $Counter=Counter+1$ و مصرف کننده دستور $Counter=Counter-1$ را اجرا نماید بدیهی است که مقدار صحیح Counter مجدداً باید 5 باشد درحالی که با اجرای همروند این دو فرایند ممکن است مقدار این متغیر 5.4 یا 6 باشد.

یک دستور لزوماً یک خط نیست وقتی به زبان ماشین ترجمه می شود به چند خط تبدیل می شود.



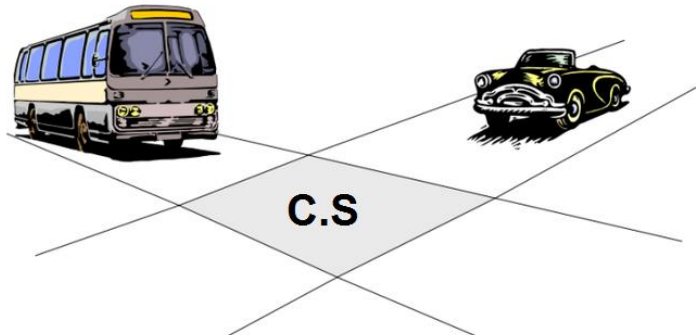
- T0: producer execute $R1 = \text{counter}$ { $R1 = 5$ }
T1: producer execute $R1 = R1 + 1$ { $R1 = 6$ }
T2: consumer execute $R2 = \text{counter}$ { $R2 = 5$ }
T3: consumer execute $R2 = R2 - 1$ { $R2 = 4$ }
T4: producer execute $\text{counter} = R1$ { $\text{counter} = 6$ }
T5: consumer execute $\text{counter} = R2$ { $\text{counter} = 4$ }

برای جلوگیری از این مشکل باید کاری کنیم که تنها یک فرآیند در هر زمان بتواند یک متغیر مشترک را دستکاری کند.

ناحیه بحرانی (Critical Section)

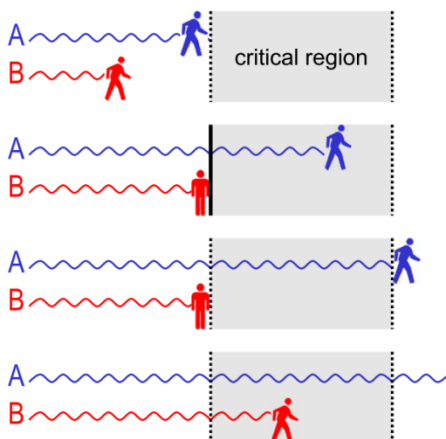
قسمتی از فرآیند است که دسترسی به منبع مشترک با فرآیند دیگر در آنجا صورت می‌گیرد. در این حالت فقط یک فرآیند در هر لحظه در ناحیه بحرانی خود قرار دارد.

```
while (true) {  
    entry section  
    critical section  
    exit section  
    remainder section  
}
```



الگوریتم‌هایی که به منظور هماهنگی و همگام سازی فرایندها پیشنهاد می‌شوند باید شروط زیر را رعایت کنند:

۱- شرط انحصار متقابل (Mutual exclusion): یعنی هنگامی که فرآیندی در ناحیه بحرانش اجرا می‌گردد، هیچ فرایند دیگری نباید در ناحیه بحرانی باشد.



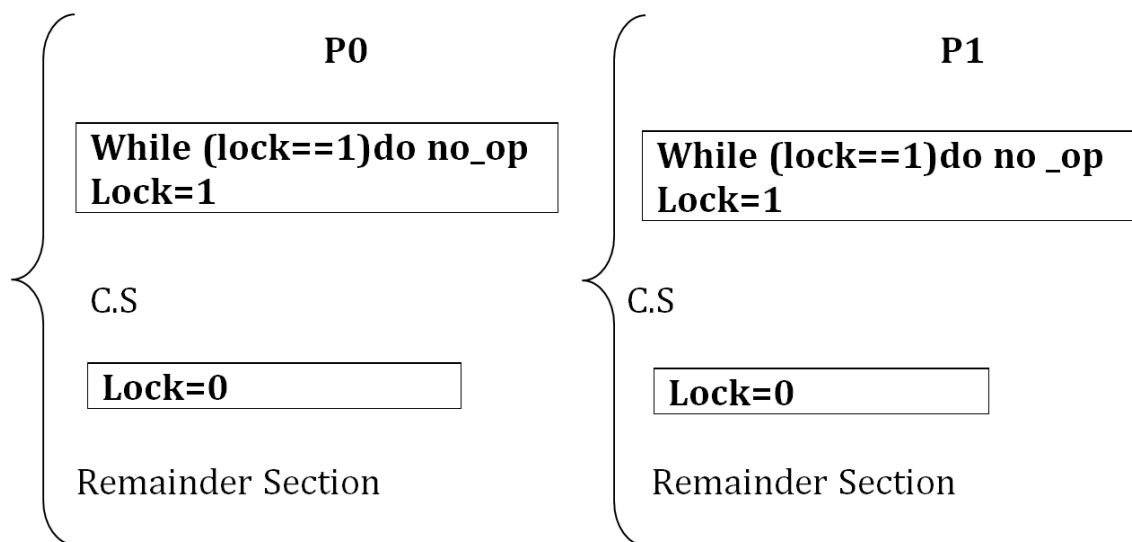
۲- شرط پیشرفت (Progress): اگر هیچ فرایندی در ناحیه بحرانی خود حضور ندارد و تقاضاهایی برای ورود به ناحیه بحرانی توسط سایر فرایندها وجود دارد، فقط فرایندهایی که هنوز به ناحیه بحرانی نرسیده‌اند در تصمیم‌گیری برای ورود دخالت می‌کنند. به عبارت دیگر فرایندی که نه در ناحیه بحرانی است و نه داوطلب ورود به ناحیه بحرانی، نباید مانع ورود فرایندهای دیگر به ناحیه بحرانشان شود.

۳- شرط انتظار محدود (**Bounded waiting**): یک فرآیند منتظر ورود به ناحیه بحرانی نباید به طور نامحدود در حالت انتظار باقی بماند (مدت استفاده از ناحیه بحرانی برای هر فرآیند محدود است).

ساده ترین راه حل: از کار انداختن وقفه قبل از C.S و فعال کردن مجدد آن بعد از C.S
این روش در سیستمهای چندپردازنده ای کاربرد ندارد. همچنین ممکن است برنامه کاربر دوباره وقفه را فعال نکند. پس این روش راه حل مناسبی نیست.

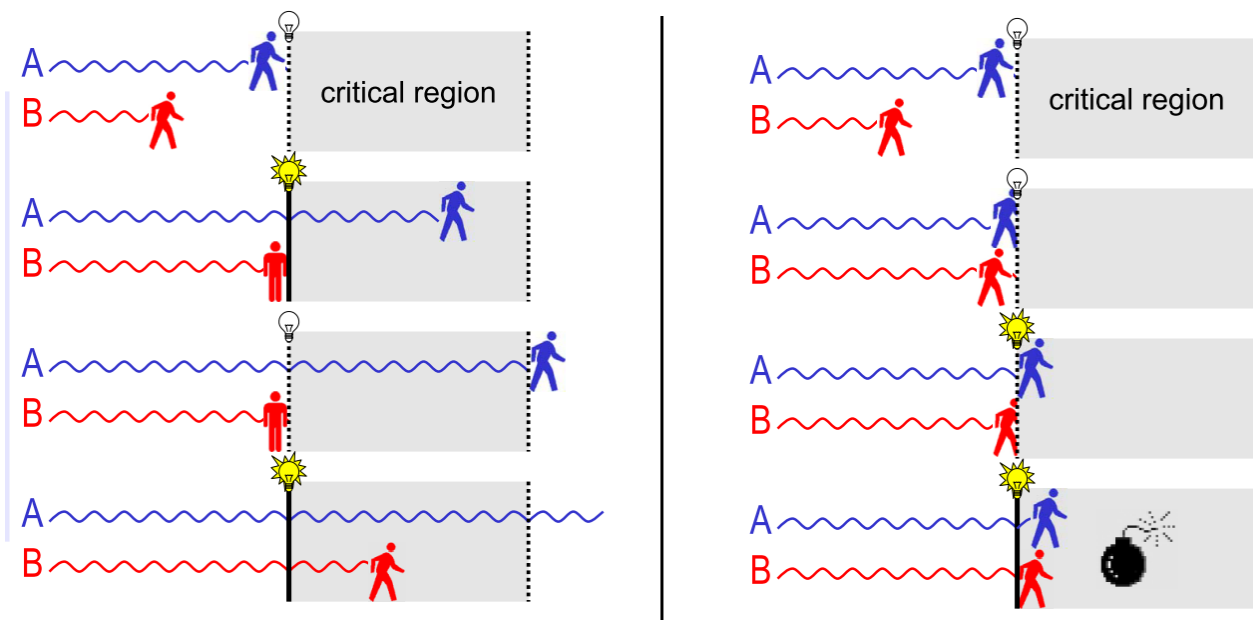
الگوریتمهای هماهنگ سازی فرآیندها

روش استفاده از متغیر قفل (متغیر lock): یک راه حل کاملاً نرم افزاری است که بر اساس متغیر مشترکی به نام lock کار می کند.

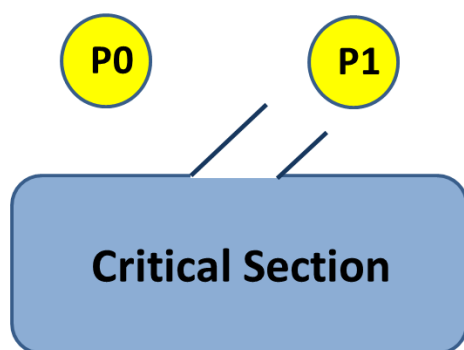


این روش شرط انحصار متقابل را برآورده نمی کند.

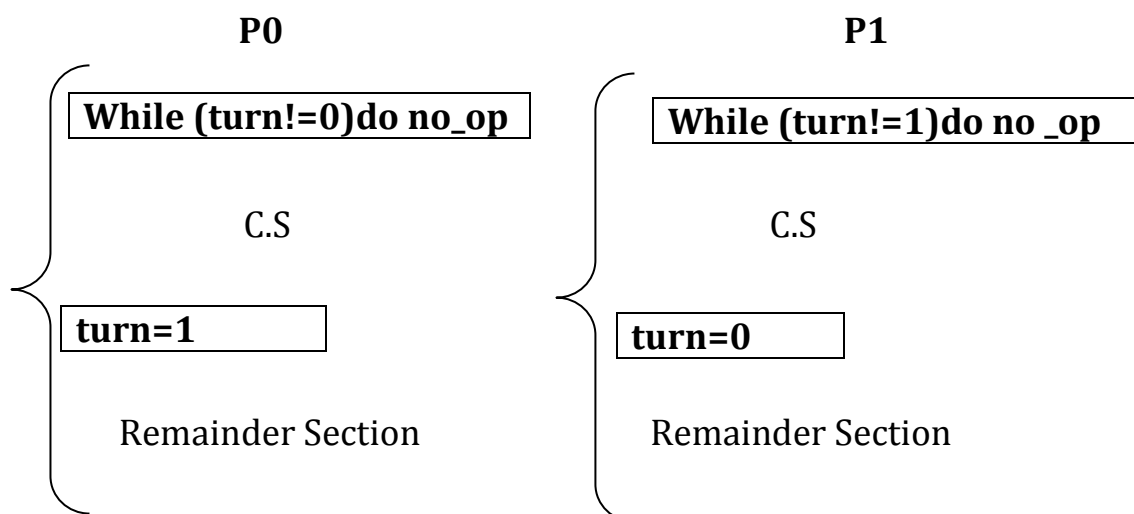
فرآیند P0 در حلقه ی While متغیر lock را چک می کند و چون برابر با صفر است به سراغ خط بعدی می رود تا lock را یک کند ولی قبل از آنکه این دستور را انجام دهد برش زمانی تمام شده و CPU به فرآیند P1 داده می شود. فرآیند P1 نیز متغیر lock را برابر صفر می بیند و از حلقه ی While خارج می شود و lock را برابر با یک کرده و وارد ناحیه بحرانی خود می شود. حال اگر دوباره پردازنده به P0 داده شود این فرآیند نیز عدد یک را در lock ریخته و وارد ناحیه بحرانی خود می شود.



روش تناوب قطعی (Strict Alternation) یا الگوریتم Dekker:



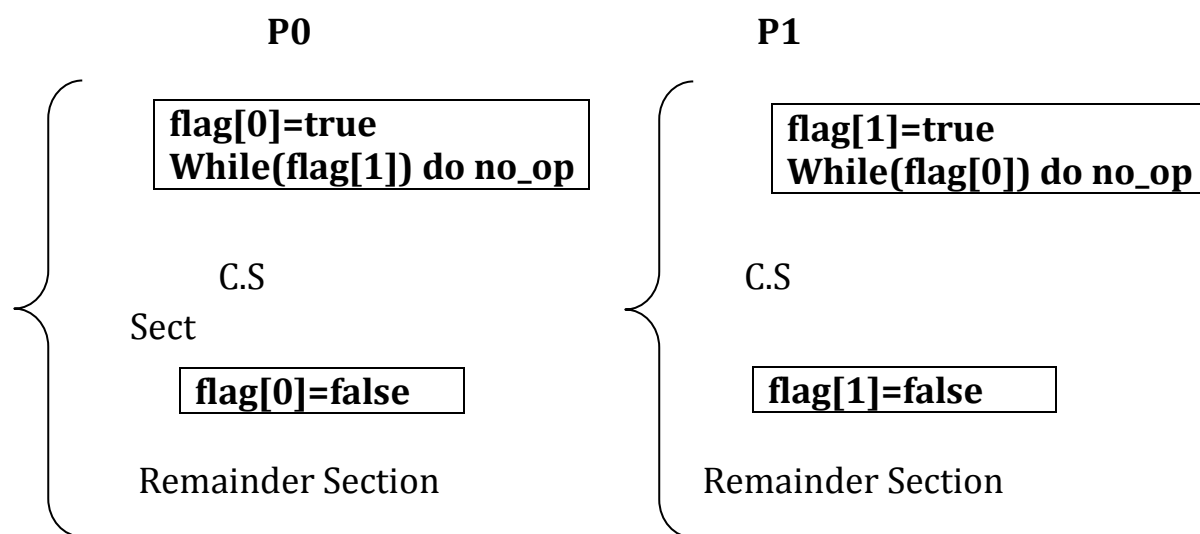
در این روش ورود به ناحیه بحرانی به صورت نوبتی امکان پذیر است. فرض میکنیم که ناحیه بحرانی دارای یک ورودی است و ورودی به سمت هر فرآیندی که باشد آن فرآیند می تواند وارد ناحیه بحرانی خود شود اما فرآیند موظف است پس از خروج از ناحیه بحرانی ورودی را به سمت فرآیند دیگر تنظیم نماید.



✓ این الگوریتم شرط پیشرفت را برآورده نمی کند زیرا اگر یک فرآیند نه در ناحیه بحرانی باشد و نه برای ورود به آن رقابت کند می تواند فرایند دیگر را بلوکه کند.

روش استفاده از **flag** ها:

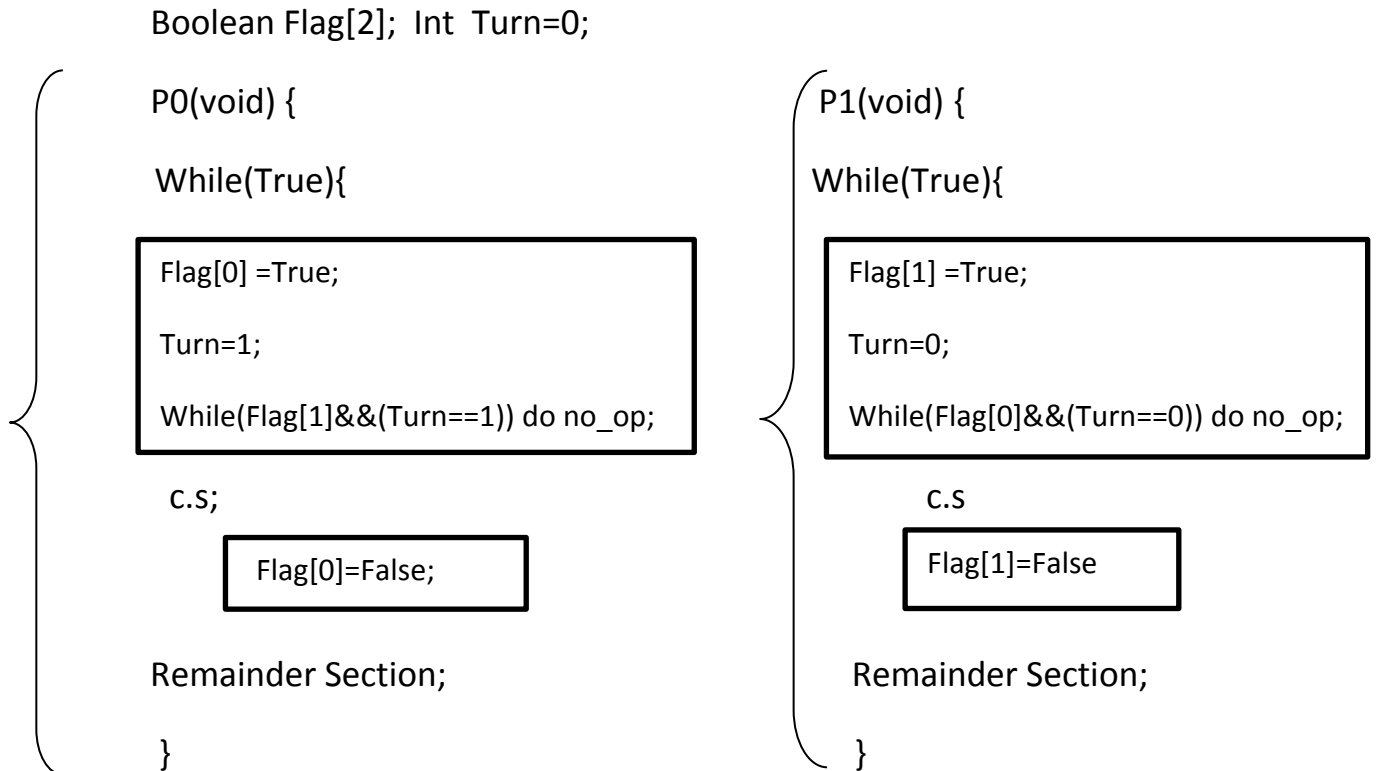
در این روش به جای استفاده از متغیر **turn** از یک آرایه دو خانه ای از متغیرهای **Boolean** استفاده می کنیم و مقدار اولیه این عناصر **false** است. $flag[0] = flag[1] = false$



در این الگوریتم شرط انحصار متقابل و شرط پیشرفت برقرار است. ولی شرط انتظار محدود را برآورده نمی کند. مثلاً اگر P0 بعد از اجرای دستور اول ($flag[0]=true$) پردازنده را از دست بدهد (بازه زمانی آن تمام شود) و پردازنده در اختیار P1 قرار گیرد، $flag[1]$ هم true شده و هر دو فرآیند در حلقه While گیر می کنند (در وضعیت no-op باقی خواهند ماند). هر چند CPU بین آنها جابجا می شود ولی وضعیت تغییر نمی کند.

الگوریتم پترسون (Peterson)

در این الگوریتم هم از متغیر **Turn** و هم از متغیر **Flag** استفاده می شود (تلفیقی از دو الگوریتم قبلی می باشد). متغیر **Turn** مبین این است که کدام فرآیند حق بیشتری برای ورود به ناحیه بحرانی را دارد.



```

Void main()
{
  Flag[0]=False;
  Flag[1]=False;
  Par begin(P0, P1);
}

```

مزایای الگوریتم پترسون:

این الگوریتم هر سه شرط را به صورت زیر برآورده می‌کند:

✓ ارضای شرط انحصار متقابل:

زمانی که هر دو فرآیند بخواهند وارد ناحیه بحرانی خود شوند (خط ۳)، متغیرهای Flag[0] و Flag[1] می‌توانند هر دو مقدار true داشته باشند، اما متغیر Turn نمی‌تواند هم صفر و هم یک باشد. لذا هر لحظه، فقط به یک فرآیند اجازه ورود به ناحیه بحرانی داده می‌شود و در نتیجه شرط انحصار متقابل در این الگوریتم برقرار است.

✓ ارضای شرط پیشرفت:

اگر فرآیند P1 در بخش باقیمانده خود باشد (بعد از خط ۴) و در این زمان فرآیند P0 از ناحیه بحرانی بیرون آمده و سپس مجدداً درخواست این ناحیه را داشته باشد، چون [1]Flag از قبل False شده (در خط ۴) پس باز هم وارد ناحیه بحرانی می شود و لذا شرط پیشرفت برقرار است.

✓ ارضای شرط انتظار محدود:

اگر یک فرآیند بخواهد مکرراً به ناحیه بحرانی دسترسی داشته باشد، فرآیند درخواست کننده دیگر در صورت نیاز به ناحیه بحرانی، با تنظیم مقدار Flag خود به true (در خط ۱)، اجازه دسترسی را از فرآیند رقیب (اول) می گیرد. سپس نوبت دهی تصادفی نبوده و گرسنگی وجود ندارد. مشکل بن بست نیز در این الگوریتم وجود ندارد چون پس از اجرای موازی یا همروند خطوط ۱ و ۲ با توجه به مقدار Turn، حتماً یکی از فرآیندها وارد ناحیه بحرانی می شود. پس شرط انتظار محدود همواره برقرار است.

عیب: انتظار مشغول رخ می دهد، زیرا تا زمانی که P0 در ناحیه بحرانی است P1 بی کار است و سیکل های پردازنده را به هدر می دهد.

مثال: اگر فرایندهای P0 و P1 به صورت زیر باشند کدامیک از خروجی ها قابل اجرا و کدامیک غیر قابل اجرا می باشند؟

P0 :

Print(A);

Print(B);

P1:

Print(C);

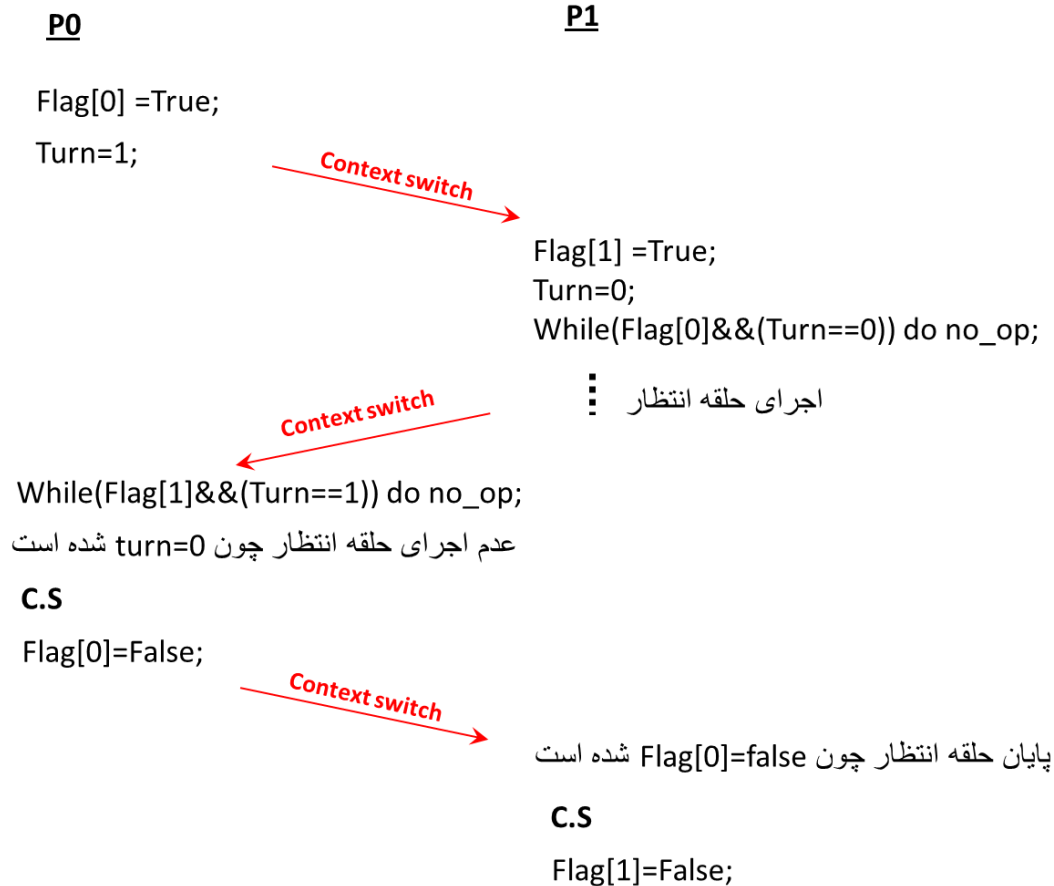
Print(D);

الف) ABCD: قابل اجرا می باشد زیرا ابتدا فرآیند P0 و سپس تعویض متن رخ داده و فرآیند P1 اجرا شده است.

ب) ACDB: قابل اجرا می باشد زیرا ابتدا دستور چاپ A و سپس تعویض متن رخ داده و دستور C چاپ می شود و سپس دستور D چاپ شده و مجدداً تعویض متن صورت می گیرد و دستور B چاپ می شود.

ج) ABCB: غیر قابل اجراست. دستور A چاپ شده و سپس برنامه ادامه داده می شود و دستور B چاپ و سپس تعویض متن صورت گرفته و دستور C چاپ می شود و در ادامه چون دستور B قبلاً یکبار اجرا شده نمی تواند مجدداً اجرا شود.

مثال برای تعویض متن در زمانهای مختلف در الگوریتم پترسون



همگام سازی با حمایت سخت افزار

اگر سخت افزار قابلیت اجرای دستورات مشخصی را به صورت وقفه ناپذیر (atomic) داشته باشد، با استفاده از متغیر مشترک می توان مسأله ناحیه بحرانی را حل کرد.

۱- دستور Test_and_set

```
Boolean Test_and_set (Boolean &target)
{
    Boolean rv=target;
    Target=True;
    Return rv;
}
```

در این دستور بار اول ورودی همان خروجی است ولی بررسی می کند که دفعات بعدی خروجی حتماً True باشد.

فرآیندهایی که می‌خواهند به صورت هماهنگ عمل کنند با استفاده از متغیر Lock با مقدار اولیه False مطابق زیر عمل می‌کند:

```
While( Test_and_set(Lock)) do nothing ;
```

C.S

```
Lock= False
```

Remainder Section;

۲- دستور **Swap** :

```
Void Swap(Boolean &A, &B)
```

```
{
```

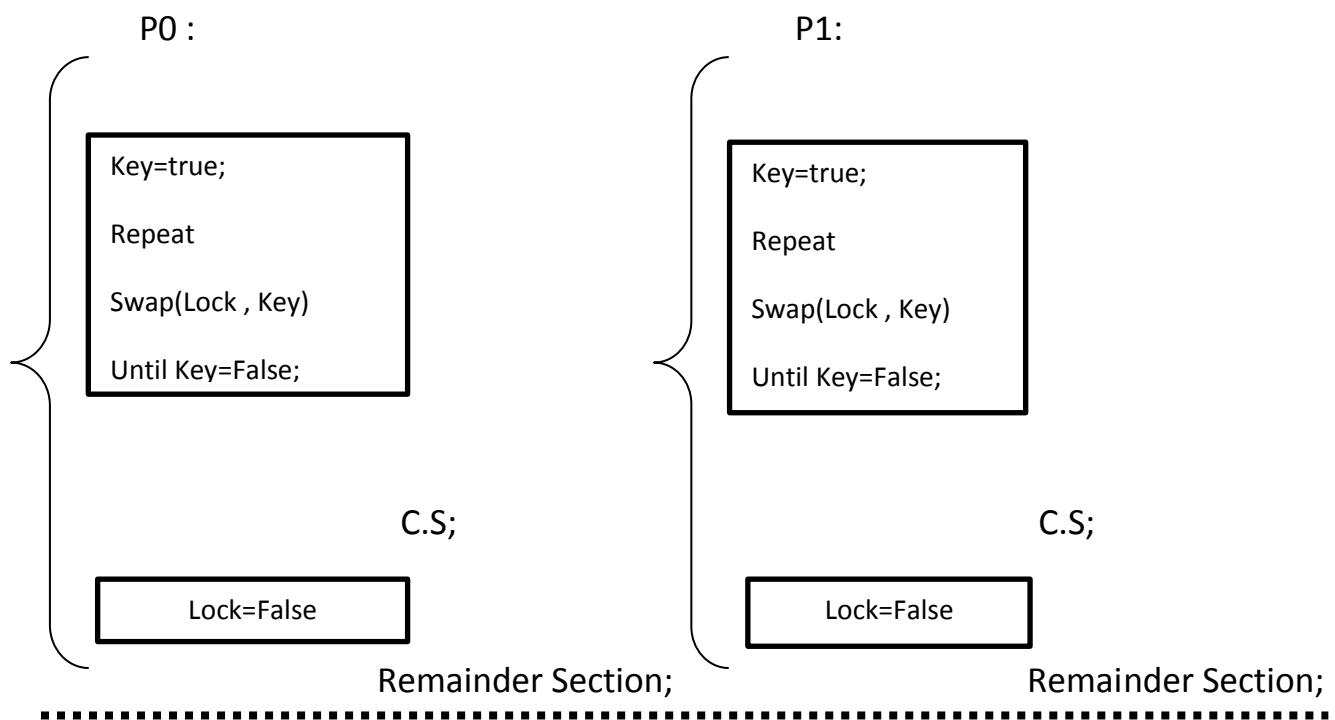
```
Boolean temp=A;
```

```
A=B;
```

```
B=temp;
```

```
}
```

فرآیندهایی که می‌خواهند از این دستور استفاده کنند باید یک متغیر محلی بنام Key با مقدار اولیه False و یک متغیر مشترک بنام Lock با مقدار اولیه False استفاده کنند. در این صورت:



هر کدام Lock را تغییر دهند در دیگری نیز تغییر می‌کند اما Key متغیر محلی است و هر کدام فقط می‌توانند Key مربوط به خود را تغییر دهند.

دستور Swap دستوری وقفه ناپذیر است.

تمامی روش‌هایی که تاکنون مطرح شد از مکانیزم انتظار مشغولی (Busy Waiting) استفاده می‌کنند. انتظار مشغولی بدین معنی است که فرآیند در حلقه گیر می‌کند و منتظر می‌شود تا اینکه اجرا شود و یا بازه زمانی آن تمام شود.



سمافور (Semaphore)

متغیر صحیحی (Int) است بنام S که صرف نظر از مقدار دهی اولیه‌اش فقط از طریق دو عمل وقفه ناپذیر Wait و Signal قابل دسترسی است.

این دو عمل به صورت زیر تعریف می‌شوند:

```
signal(S)
{
    S++;
}

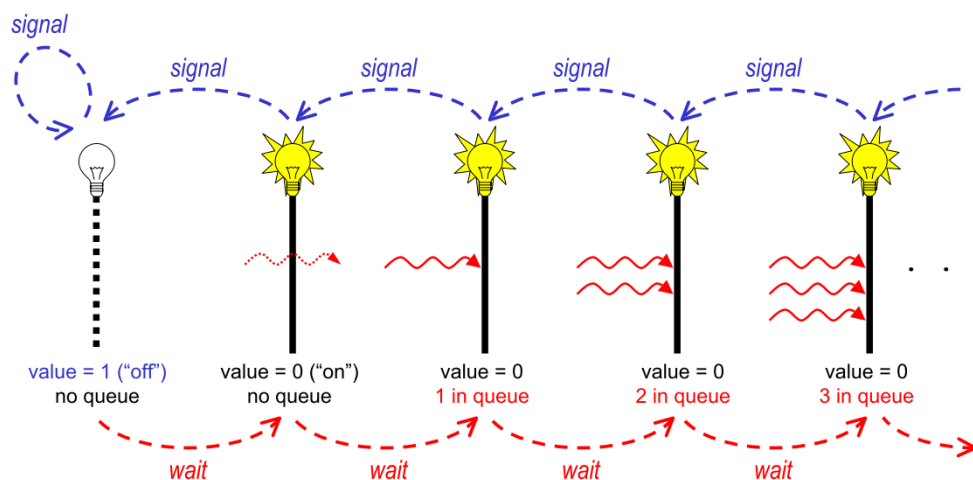
wait(Semaphore S)
{
    while S <= 0
        ; //no operation
    S--;
}
```

سمافورها به دو دسته تقسیم می‌شوند:

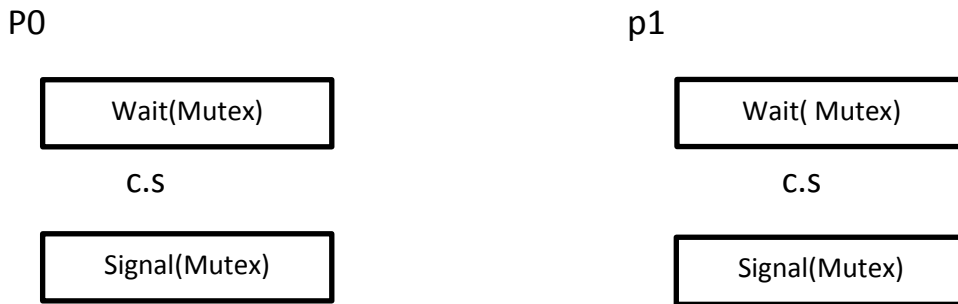
شمارشی: متغیر می‌تواند هر عدد صحیحی باشد.

باینری: متغیر می‌تواند فقط معادل ۰ یا ۱ داشته باشد.

Binary semaphore $\Leftrightarrow$ mutex



سمافور باینری به صورت زیر می‌باشد: (در صورتی که مقدار $Mutex=1$ باشد)



نکته:

Wait و Signal وقفه ناپذیر بوده پس در وسط این دو عمل وقفه‌ای صورت نمی‌گیرد.

همچنین می‌توان با استفاده از سمافور عمل هماهنگی فرآیندها را انجام داد. مثلاً اگر در دو فرآیند P0 و P1 لازم باشد دستوری در فرآیند P0 (مثلاً دستور MOV) حتماً قبل از دستوری در فرآیند P1 (دستور ADD) اجرا شود از سمافوری به نام Sync با مقدار اولیه صفر می‌توان به صورت زیر استفاده کرد.



نکته: مقداردهی اولیه به سمافور اهمیت زیادی دارد.

برای رفع انتظار مشغول (Busy waiting) می‌توان عملیات Wait و Signal را در سمافور به صورت زیر بهبود داد. برای این کار سمافور را به صورت یک رکورد تعریف می‌کنیم:

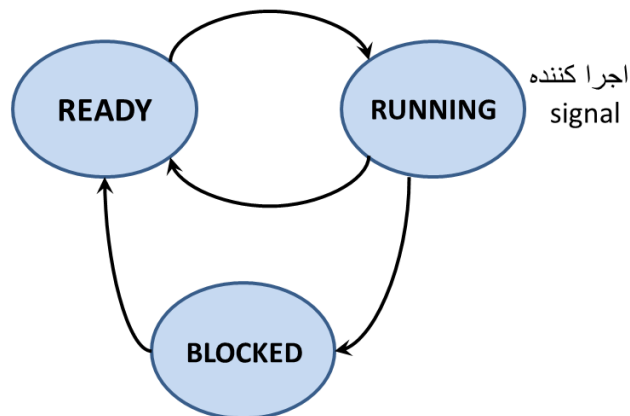
```
Typedef struct {
    Int Value;
    Stuct process *L;
} semaphore;
```

Wait(S)

```
S.Value --;  
  
If(S.Value<0)  
{  
  
Add this process to S.L;  
  
Block the process;  
  
}
```

Signal (s)

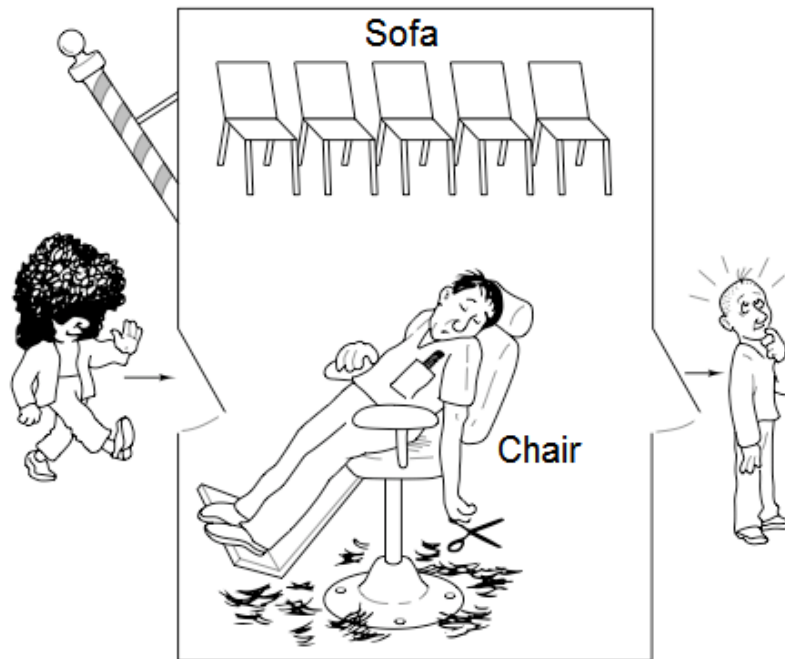
```
S.Value ++;  
  
If(S.Value≤0)  
{  
  
Remove a process P from S.L;  
  
Wake-up (P) (از حال انتظار به آماده)  
  
}
```



با اجرای signal یکی از فرآیندهایی که در این وضعیت بود به حالت آماده میرود. برای خود فرایندی که signal را اجرا کرده اتفاقی نمی افتد.

مسئله آرایشگر خواب آلود (Sleeping barber problem)

یک مغازه سلمانی دارای اتاق انتظار با n صندلی و یک صندلی برای کوتاه کردن مو است، اگر هیچ مشتری وجود نداشته باشد سلمانی به خواب می رود. اگر سلمانی مشغول به کار باشد و صندلی خالی موجود باشد، مشتری بر روی یک صندلی خالی می نشیند. اگر سلمانی خواب باشد مشتری او را بیدار می کند. این مسئله را به کمک سمافور حل کنید.



مقدار دهی اولیه:

Sofa :n

chair :1

ready :0

finished :0

leave-chair :0



(Costumer)

enter Barber-shop;

مشتری وارد مغازه می شود.

wait(sofa);

یکی از جاهای خالی sofa کم می شود.

Sit on sofa;

مشتری در صف sofa می نشیند .

wait(chair); chair=0 می شود.

Get up from sofa; مشتری از صف sofa بلند می شود .

Signal(sofa); یکی به جاهای خالی sofa اضافه می شود.

Sit in chair; مشتری بر روی صندلی chair می نشیند.

Signal (ready); مشتری آماده است.

Wait(finished); کار مشتری تمام شده است.

Leave chair; مشتری صندلی را ترک می کند.

Signal(leave-chair);



(Barber)

wait(ready); هیچ کس در حال حاضر آماده نیست.

cut hair; سلمانی موهای مشتری را کوتاه می کند.

signal(finished); کار مشتری تمام شده است.

wait(leave-chair); مشتری از مغازه خارج می شود.

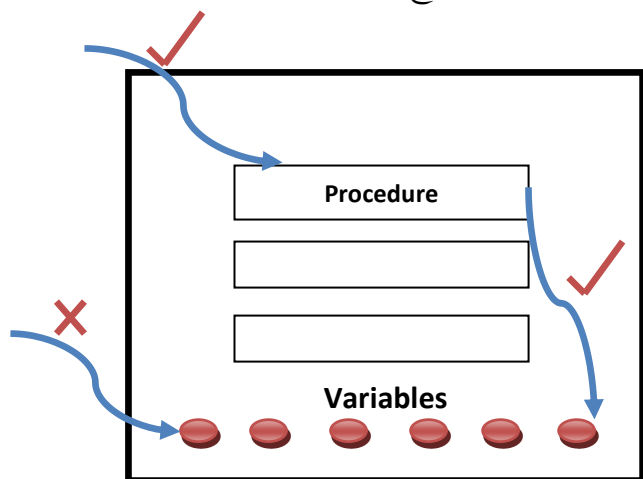
signal(chair); صندلی در حال حاضر خالی است و آماده پذیرش مشتری است.

تمرین: در مورد مساله فلاسفه و میز ناهار خوری تحقیق کرده، آن را به وسیله سمافور پیاده سازی کنید.

مانیتور

یکی دیگر از ابزارهای مورد استفاده برای هماهنگی فرآیندها، مانیتور می‌باشد. مانیتور یک ابزار نرم‌افزاری سطح بالاست که از مجموعه‌ای از رویه‌ها (procedure) و متغیرها تشکیل شده‌است.

این متغیرها صرفاً از طریق رویه‌های داخلی مانیتور قابل استفاده هستند و خارج از مانیتور فقط رویه‌ها قابل دسترسی‌اند.



۱ تعریف متغیرها

۲ تعریف رویه‌ها

۳ مقدار دهی کدها

ساختار مانیتور به شکل زیر است:

Monitor monitor-name

```
{
  ۱ shared variables and declarations;
  ۲ {
    procedure P1(...) { ... }
    procedure P2(...) { ... }
    ...
    procedure Pn (...) { ... }
  }
  ۳ Initialization code { ... }
}
```

Monitor Producer-Consumer

```
{  
  ۱ {  
    condition full,empty;  
    int count;  
  
    ۲ Procedure enter() {  
      if(count==N) wait(full);  
      put_item(widget);  
      count++;  
      if(count==1) signal(empty);  
  
      Procedure remove() {  
        if(count==0) wait (empty);  
        remove_item(widget)  
        count-- ;  
        if(count==N-1) signal(full)  
      }  
    }  
    ۳ count=0;  
  }  
}
```

همواره ناحیه بحرانی فرآیندهای مرتبط که دارای منابع مشترک هستند به صورت رویه‌های داخل مانیتور قرار می‌گیرد.

فرایند تولید کننده(Producer())

```
{  
while(true)  
{  
    make-item (widget)  
    Producer-Consumer.enter();  
}  
}
```

فرایند مصرف کننده(Consumer())

```
{  
while(true)  
{  
    Producer-Consumer.remove();  
    consume-item;  
}  
}
```

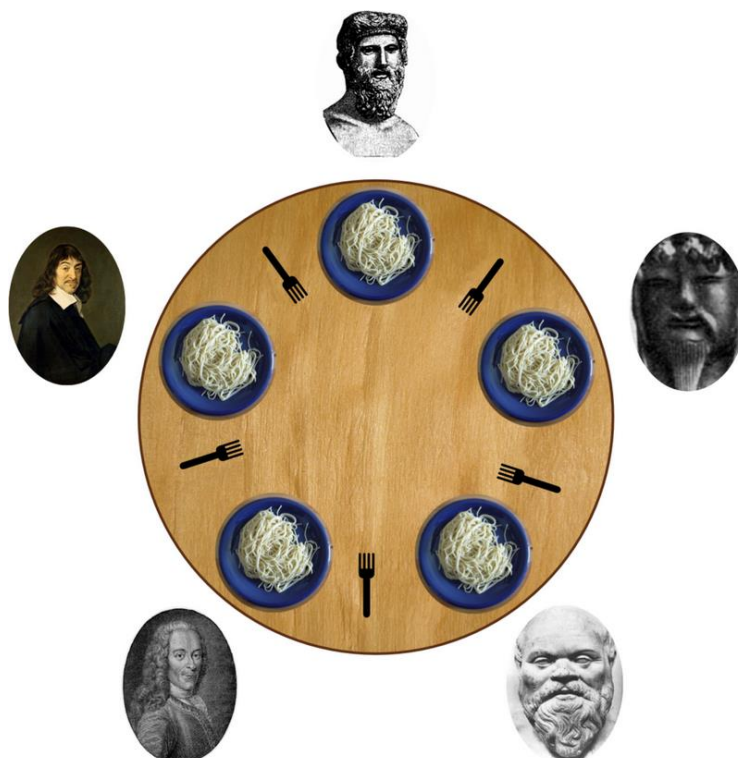
نکته:

- در هر لحظه فقط یک فرایند در مانیتور در حال اجراست.
- همواره ناحیه بحرانی فرایندهای مرتبط که دارای منابع مشترک هستند بصورت رویه‌هایی داخل مانیتور قرار می‌گیرند.
- full و empty متغیرهای شرطی هستند که بر روی آنها صرفاً دو عمل wait و signal قابل اجراست.

مساله فلاسفه و میز غذاخوری (Dining Philosophers)

پنج فیلسوف دور یک میز دایره‌ای نشسته‌اند و هر فیلسوف یک بشقاب ماکارونی دارد. در بین هر دو بشقاب، چنگالی قرار داده شده است. هر فیلسوف یا در حل فکر کردن است، و یا در حال غذا خوردن. با این حال، ماکارونی‌ها به قدری لغزنده هستند که هر فیلسوف باید با دو چنگال غذا بخورد. هر چنگال را فقط یک فیلسوف می‌تواند نگه دارد و یک فیلسوف تنها وقتی می‌تواند از چنگال‌ها استفاده کند که در حال حاضر آن چنگال در اختیار فیلسوف دیگری قرار نداشته باشد. بعد از اینکه فیلسوف مدتی غذا خورد، باید چنگال‌ها را بر روی میز قرار دهد تا بقیه فیلسوف‌ها بتوانند از آن استفاده کنند. یک فیلسوف تنها می‌تواند چنگال‌هایی که در سمت چپ و سمت راستش قرار دارند را بردارد و تا وقتی که هر دو آنها را بر نداشته، نمی‌تواند غذا خوردن را شروع کند. در این مسئله فرض می‌شود که بشقاب‌های ماکارونی هیچ‌گاه تمام نمی‌شوند و نامحدود هستند. مسئله اینجاست که چگونه می‌توان الگوریتم همروندی طراحی کرد که هیچ فیلسوفی گرسنه نماند و بتواند برای همیشه یکی در میان

به غذا خوردن و فکر کردن پردازد، با این فرض که فیلسوف‌ها نمی‌دانند که بقیه می‌خواهند غذا بخورند یا فکر کنند و این امکان وجود دارد که دو فیلسوف همزمان بخواهند یک چنگال را بردارند.



تمرین: روشی برای حل این مساله با استفاده از سمافور و مانیتور پیشنهاد دهید.

۱

۲

۳

۴

۵

۶

۷

۸

فصل پنجم

بن بست

در یک سیستم چند برنامه‌گی امکان رقابت فرایندها برای در اختیار گرفتن منابع وجود دارد. در موقعیتی که تعدادی فرآیند منتظر در سیستم وجود دارد که هر کدام منابعی را در اختیار دارند و دیگران منتظر همان منابع هستند و با پیشرفت زمان تغییری در حالت فرایندها پیش نمی‌آید، در سیستم بن بست رخ داده است.

انواع منابع از نظر نوع استفاده :

► منابع قابل استفاده مجدد (Reusable Resources)

◦ منابعی که بارها توسط فرایندها می‌توانند استفاده شوند.

• پردازنده، حافظه اصلی و جانبی - I/O

► منابع غیر قابل استفاده مجدد (Non-reusable Resources)

◦ منابعی که با یک بار استفاده، دیگر قابل استفاده نیستند.

• وقفه‌ها، پیام‌ها، علائم

هرگاه فرآیندی بخواهد منبعی را بدست آورد باید مراحل زیر طی شود:

۱- در خواست منبع و به دست آوردن آن: اگر منبع آزاد باشد آن را به دست می‌آورد در غیر این صورت به وضعیت بلوکه می‌رود.

۲- استفاده از منبع: فرآیند می‌تواند روی منبعی که در اختیار دارد کار کند.

۳- آزاد کردن منبع: با اتمام کار، فرایند منبع را آزاد می‌کند و این منبع به مجموع منابع آزاد سیستم اضافه می‌شود.

*** در خواست منبع و آزاد کردن آن از طریق system call انجام می‌گیرد.

✓ برای رخداد بن بست باید همه شرایط زیر فراهم باشد:

◦ انحصار متقابل (mutual Exclusion)

یعنی منبعی وجود داشته باشد که در هر زمان فقط یک فرآیند بتواند از آن استفاده کند.

◦ گرفتن منابع و انتظار برای منابع جدید (Hold & Wait)

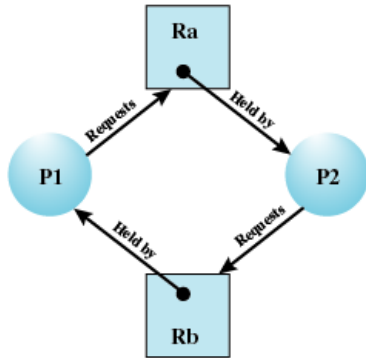
یک فرآیند بتواند منبعی را در اختیار بگیرد و تا زمانی که دیگر منابع مورد نیاز وی (فرآیند مورد

نظر) که در اختیار فرایندهای دیگر است آزاد نشود، منتظر بماند.

- عدم قابلیت پس گرفتن منابع (انقطاع ناپذیری)
منابع را نتوان پیش از اتمام کار فرآیند و آزاد شدن آنها از یک فرآیند پس گرفت.

- انتظار حلقوی (Circular Waiting) (شرط اصلی)

چرخه‌ای از فرآیندهای منتظر وجود داشته باشد و هر کدام منابع مورد نیاز فرآیندهای دیگری را در اختیار دارد.



(c) Circular wait

گراف تخصیص منابع (resource allocation graph): روشی برای نمایش بهتر مسائل بن بست است.

فرآیندها را با یک دایره نشان می‌دهیم. 

منابع را با مستطیل نشان می‌دهیم و نقطه‌های درون آن تعداد منابعی را که از آن موجود است را نشان می‌دهد.

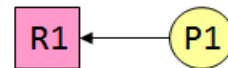


R_4

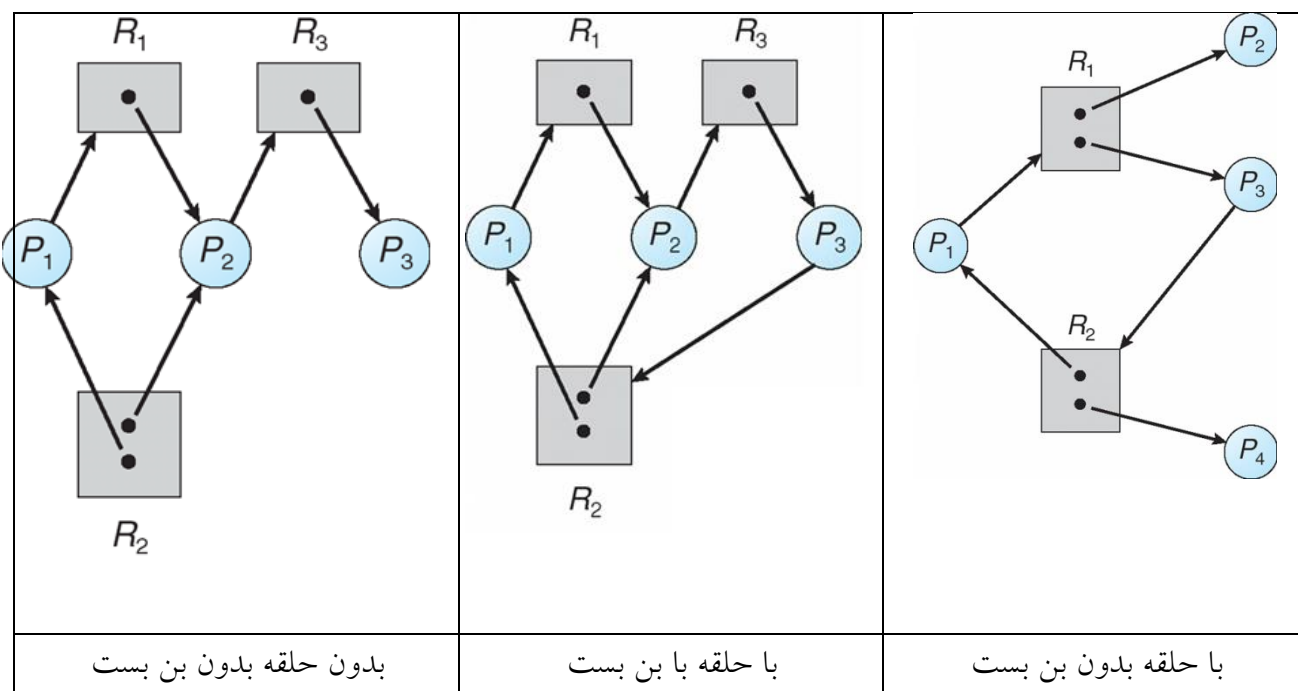
جهت فلش مشخص کننده درخواست و یا در اختیار داشتن منبع است.



فرآیند P1 منبع R1 را در اختیار دارد



فرآیند P1 درخواست منبع R1 را دارد



معیار اولیه‌ی تشخیص بن بست از روی گراف

گراف حلقه ندارد: $\leftarrow$ به طور قطع بن بست نیست

گراف حلقه دارد:

الف- از هر نوع منبع درگیر در حلقه فقط یک نمونه داریم $\leftarrow$ بن بست رخ داده است.

ب- از هر نوع منبع درگیر در حلقه چند نمونه داریم $\leftarrow$ ممکن است بن بست داشته باشیم.

روش‌های برخورد با بن بست



۱- روش اول: استراتژی شترمرغ (ostrich strategy). در این استراتژی بیخیال بن بست می‌شود. به عبارت دیگر، مشکل را نمی‌بینیم و امیدوارم مشکل هم ما را نبیند 😊. اگر وقوع بن بست خیلی نادر باشد در صورت وقوع آن را نادیده می‌گیریم و سیستم را restart می‌کنیم (مانند سیستم عامل windows). نفوس بد نزن. انشاءالله اتفاقی نمی‌افته ...

۲- روش دوم: پیشگیری از بن بست (Deadlock Prevention)

در این روش تلاش می‌شود حداقل یکی از شرایط چهارگانه نقض شود.

الف- نقض شرط انحصار متقابل

برای برخی از منابع نظیر چاپگر نمی‌توان از این روش استفاده کرد زیرا آن را نمی‌توان به صورت اشتراکی استفاده نمود. اما برای یک فایل فقط خواندنی (read only) می‌توان از این روش استفاده کرد.

ب- نقض شرط نگهداری و انتظار (Hold & Wait)

باید مانع از ایجاد موقعیتی شد که در آن یک فرایند منبعی را در اختیار بگیرد و تقاضای منبع دیگری نماید. فرایند دو راه برای درخواست منبع دارد:

- تمام منابع مورد نیاز را در ابتدای اجرا در خواست کند.
 - همه منابع در اختیار را آزاد کرده و سپس تقاضای منبع جدید کند.
- کارایی این روش پایین بوده و احتمال قحطی (گرسنگی) وجود دارد.

ج- نقض شرط انقطاع ناپذیری:

سیستم عامل می‌تواند بررسی کند که اگر فرآیندی درخواست یک منبع کرده که امکان تخصیص آن وجود ندارد و آن فرآیند منبعی را در اختیار دارد که مورد درخواست سایر فرآیندها است، منابع را از یک فرآیند پس گرفته و به فرآیند دیگری تخصیص می‌دهد.

د- نقض شرط انتظار دوره‌ای

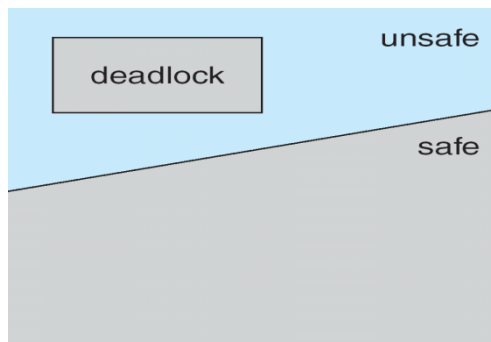
برای نقض این شرط می‌توان به فرآیندها اجازه نداد که منابع را به هر ترتیبی درخواست کنند. منابع را می‌توان شماره‌گذاری کرد و به فرآیندها اجازه می‌دهیم فقط به ترتیب افزایش شماره منابع، آن‌ها را درخواست کنند. به عنوان مثال، اگر به فرآیند P1 منابع R5 تخصیص داده شود، اکنون دفعه بعد اگر P1 درخواست R4 یا R3 کند، چنین درخواستی قبول نخواهد شد و فقط درخواست برای منابع بیشتر از R5 اعطا خواهد شد.

۳- روش سوم: روش اجتناب کردن از بن بست (Deadlock Avoidance)

در این روش سیستم تلاش می‌کند تا با استفاده از یک "دانش قبلی" درباره رفتار فرآیند (نوع-ترتیب) از وقوع بن بست جلوگیری کند. باید فرآیندها را ملزم کرد که قبل از شروع به اجرا، نوع و حداکثر تعداد منبع مورد نیاز خود را اعلام کند.

الگوریتمی که با داشتن حداکثر نیازهای کلی فرآیندهای موجود به انواع منابع بتواند راه حلی برای استفاده از آنها بدون رفتن به حالت بن‌بست پیشنهاد کند، یک الگوریتم اجتناب از بن‌بست نامیده می‌شود.

تعریف حالت امن: یک حالت تخصیص منابع حالت امن است اگر برطبق آن حالت بتوان ترتیبی را جهت تخصیص حداکثر منابع مورد نیاز فرآیندها پیدا کرد. چنین ترتیبی را نیز یک "ترتیب امن" می‌گوییم.



ترتیب $\langle P_1, P_2, \dots, P_n \rangle$ یک ترتیب امن است در صورتی که برای هر P_i ، منابعی که P_i می‌تواند تقاضا کند زیرمجموعه‌ای از منابع موجود و منابع در اختیار P_j ها باشد ($\forall j < i$).

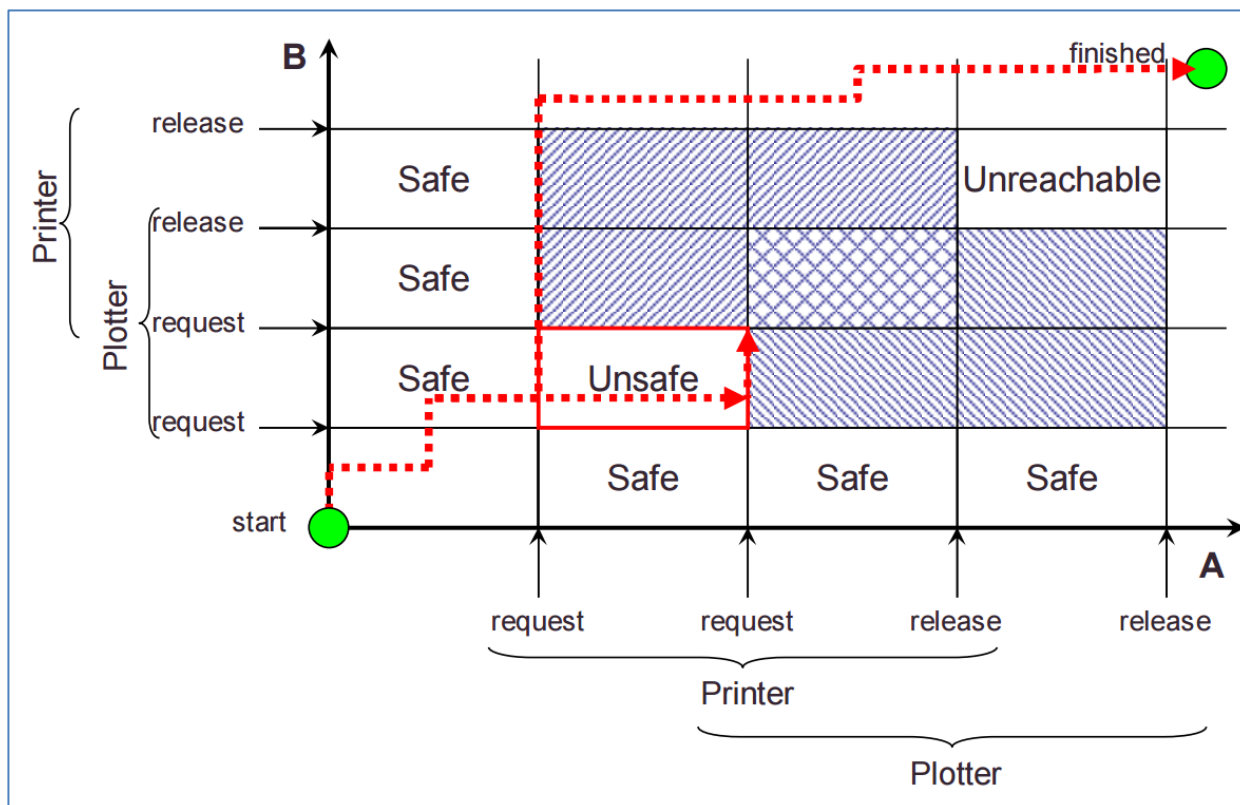
در مثال زیر اگر سه نمونه از منبع درخواستی آزاد باشد، آیا می‌توان یک ترتیب امن برای اجرای هر چهار فرایند پیدا کرد؟

فرایند	منابع درخواستی	منابع در اختیار
P1	9	4
P2	3	3
P3	6	3
P4	12	1

$P2 \rightarrow P3 \rightarrow P1 \rightarrow P4$

✓ اگر سیستم در حالت امن باشد بن‌بست وجود ندارد.

✓ اگر سیستم در حالت نا امن باشد ممکن است بن‌بست رخ دهد.



الگوریتم‌های اجتناب از بن‌بست تضمین می‌کنند که سیستم هیچگاه وارد حالت ناامن نشود.

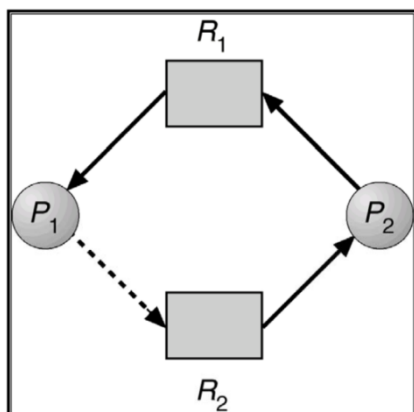
- هنگامی که از هر منبع تنها یک نمونه موجود است از الگوریتم گراف تخصیص منابع استفاده می‌کنیم.
- هنگامی که بیش از یک نمونه از منبع موجود است از الگوریتم بانکداران استفاده می‌کنیم.

الگوریتم گراف تخصیص منابع

اجزای الگوریتم:

- وقتی ممکن است فرایند P_i منبع R_j را تقاضا کند، یک *یال خواسته* (claim edge) به صورت خط چین بین رئوس مربوطه می‌کشیم.
- وقتی فرایند منبع را درخواست می‌کند، یال خواسته به یال تقاضای منبع تبدیل می‌شود.
- وقتی فرایند منبع را آزاد می‌کند، یال تخصیص به یال خواسته تبدیل می‌شود.
- منابع مورد نیاز باید از قبل از سیستم خواسته شوند.

حالت ناامن در گراف تخصیص منابع



فرض کنیم فرآیند P_i منبع R_j را درخواست کند.
درخواست تنها هنگامی پاسخ داده می شود که تبدیل یال تقاضا به
یال تخصیص باعث ایجاد یک حلقه در گراف نشود.

الگوریتم بانکداران

در حالتی که از هر منبع بیش از یک نمونه داشته باشیم، الگوریتم گراف تخصیص منابع جوابگو نیست و باید از الگوریتم بانکداران (Bankers) که توسط Dijkstra مطرح شد، استفاده کرد.

در این روش هرفرآیند باید از قبل حداکثر تعداد منابع مورد نیاز خود را اعلام کند. در این الگوریتم که در روش اجتناب مطرح شده است هرگاه فرآیندی منبعی را تقاضا می کند سیستم عامل ابتدا فرض می کند که منبع را اختصاص داده است. با این فرض، شرایط جدید سیستم شبیه سازی می شود. چنانچه در این شرایط بتوان ترتیبی از پاسخگویی به فرآیندها، با منابع آزاد موجود یافت حالت وضعیت امن بوده و منبع را واقعاً به فرآیند اختصاص می دهد. در غیر اینصورت از اختصاص منبع خودداری می کند.
فرض کنید n تعداد فرآیندها و m تعداد نوع منابع باشد.

`int available [m];`

$available[j] = k$ یعنی k نمونه از منبع R_j موجود هستند.

`int max [n][m];`

$max[i][j] = k$ یعنی فرآیند P_i حداکثر ممکن است k نمونه از منبع R_j را درخواست کند.

`int allocation [n][m];`

$allocation[i][j] = k$ یعنی فرآیند P_i ، k نمونه از منبع R_j را در اختیار دارد.

`int need [n][m];`

$need[i][j] = k$ یعنی فرآیند P_i به k نمونه دیگر از منبع R_j نیاز دارد تا کارش را تمام کند.

$Need[i,j] = Max[i,j] - Allocation[i,j]$

تشخیص حالت امن و ناامن در الگوریتم بانکداران

۱. فرض کن Work و Finish دو بردار به طول m و n باشند. این بردارها را به این صورت مقداردهی اولیه کن:

Finish = {False}; Work = Available;

۲. اندیس i را به گونه ای پیدا کن که:

Need $_i$ ≤ Work; Finish [i] = False;

اگر چنین i پیدا نکردی به گام ۴ برو.

۳. Finish [i] = True; Work = Work + Allocation $_i$;

به گام ۲ برو.

۴. اگر $\forall i: \text{Finish}[i] == \text{True}$ آنگاه سیستم در یک حالت امن است.

الگوریتم درخواست منبع برای فرایند P_i

بردار Request_i را به عنوان بردار نیاز فرایند P_i تعریف می‌کنیم:

۱- اگر $\text{Request}_i \leq \text{Need}_i$ برو به گام ۲، وگرنه اعلام خطا.

۲- اگر $\text{Request}_i \leq \text{Available}$ برو به گام ۳، وگرنه P_i باید منتظر بماند تا منابع مورد نیاز آن آزاد شود.

یعنی درخواست باید کمتر از سقف و موجودی باشد.

۳- فرض میکنیم منابع مورد نیاز P_i را اختصاص داده‌ایم. حالت تخصیص منابع را به صورت زیر به روز می‌کنیم:

Available = Available - Request $_i$

Allocation $_i$ = Allocation $_i$ + Request $_i$

Need $_i$ = Need $_i$ - Request $_i$

• اگر حالت سیستم امن بود $\Leftarrow$ منابع به P_i اختصاص یافته اند.

• اگر حالت سیستم ناامن بود $\Leftarrow P_i$ باید منتظر بماند، حالت قبلی سیستم را بازیابی کن.

مثالی از الگوریتم بانکدارها

$n = 5$ P0, P1, P2, P3, P4

$m = 3$

A, B, C {
A: نمونه ۱۰
B: نمونه ۵
C: نمونه ۷

MAX - Allocation =

Process	Allocation	MAX	Available	Need
	ABC	ABC	ABC	ABC
p0	0 1 0	7 5 3	3 3 2	7 4 3
p1	2 0 0	3 2 2	$= [10, 5, 7] - [7, 2, 5]$	1 2 2
p2	3 0 2	9 0 2		6 0 0
p3	2 1 1	2 2 2		0 1 1
p4	0 0 2	4 3 3		4 3 1

Total Alloc:	7 2 5			

جستجو برای یک ترتیب امن:

می‌خواهیم با استفاده از الگوریتم تشخیص حالت امن، بدانیم که آیا درحالت امن قرار داریم یا نه؟

Work = Available = $[3, 3, 2]$

Finish = { F, F, F, F, F }

تمام شدن
P1 $\Rightarrow$ Work = $[5, 3, 2]$ Finish = { F, T, F, F, F }

تمام شدن
P3 $\Rightarrow$ Work = $[7, 4, 3]$ Finish = { F, T, F, T, F }

تمام شدن
 $P_4 \rightarrow \text{Work} = [7, 4, 5] \quad \text{Finish} = \{F, T, F, T, T\}$

تمام شدن
 $P_0 \rightarrow \text{Work} = [7, 5, 5] \quad \text{Finish} = \{T, T, F, T, T\}$

تمام شدن
 $P_2 \rightarrow \text{Work} = [10, 5, 7] \quad \text{Finish} = \{T, T, T, T, T\}$

طبق این الگوریتم چون یک ترتیب امن داشتیم $(P_1, P_3, P_4, P_0, P_2)$ ، پس در حالت امن قرار داریم.

Alternate safe sequence: $\langle P_1, P_3, P_4, P_2, P_0 \rangle$ Solution is not unique:

حال فرض کنیم فرآیند P_1 درخواست یک نمونه دیگر از منبع A و دو نمونه از منبع C را داشته باشد. آیا با این درخواست موافقت کنیم یا خیر؟

$\text{Request}_1 = [1, 0, 2] \quad \text{Available} = [2, 3, 0]$

1) $\text{Request}_1 \leq \text{Need}_1$ ☒ بیشتر از نیاز تقاضا نکرده

2) $\text{Request}_1 \leq \text{Available}$ ☒ بیشتر از منابع موجود تقاضا نکرده

3) فرض می‌کنیم منابع مورد نیاز به این فرآیند اختصاص داده می‌شود، ماتریس‌های Allocation و Need و بردار Available باید دستکاری شوند.

	A	B	C
P_0	0	1	0
P_1	3	0	2
$\text{Allocation} = P_2$	3	0	2
P_3	2	1	1
P_4	0	0	2

	A	B	C
P ₀	7	4	3
P ₁	0	2	0
Need = P ₂	6	0	0
P ₃	0	1	1
P ₄	4	3	1

P₁ → Work = [5 , 3 , 2]

P₃ → Work = [7 , 4 , 3]

P₄ → Work = [7 , 4 , 5]

P₀ → Work = [7 , 5 , 5]

P₂ → Work = [10 , 5 , 7]

با قبول این درخواست بازهم می توان ترتیب امنی پیدا کرد، پس هنوز حالت امن خواهیم داشت.

بنابراین درخواست P₁ قبول شده و منابع اختصاص داده می شوند.

حال فرض کنیم P₄ درخواستی را به صورت مقابل معرفی می کند:

Request₄ = [3 , 3 , 0]

بیشتر از منابع موجود درخواست کرده (نقض گام ۲)، بنابراین درخواست آن را رد می کنیم.

Request₄ > Available

۱

۲

۳

۴

۵

۶

۷

۸

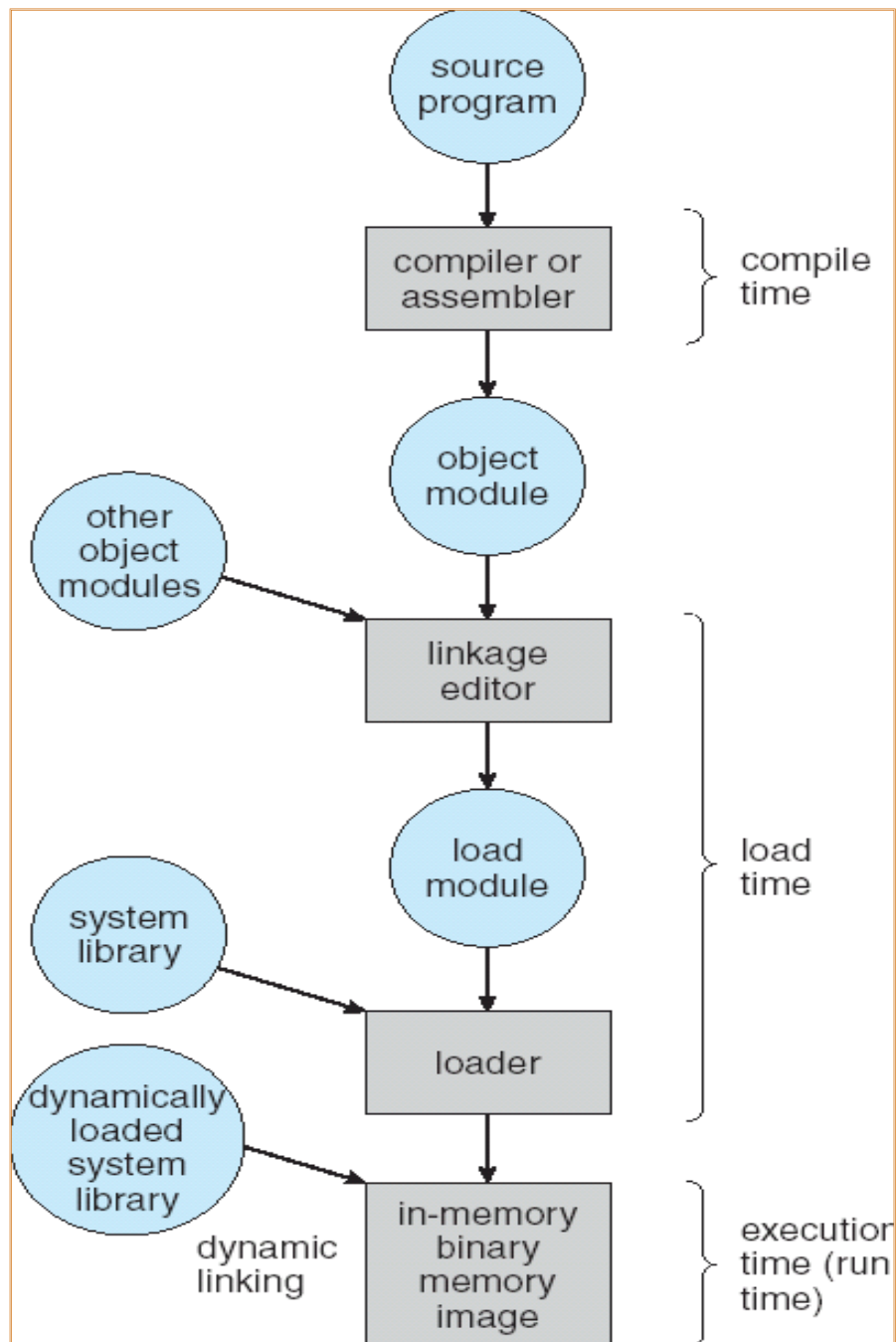
فصل ششم

مدیریت حافظه

هدف مدیریت از حافظه، تخصیص، باز پس گیری، حفاظت، استفاده اشتراکی، سازماندهی فیزیکی و منطقی و ... می باشد. سیستم عامل باید بتواند چندین فریند را همزمان در حافظه قرار دهد و مدیریت کند.

منظور مدیریت حافظه، حافظه اولیه یا RAM می باشد. در مورد حافظه ثانویه (معمولا دیسک مغناطیسی) در فصل دیگری بحث خواهد شد.

پردازش چند مرحله ای برنامه کاربر



نگاشت آدرس دستورالعمل ها و داده ها به آدرس حافظه در سه مرحله امکان پذیر است:

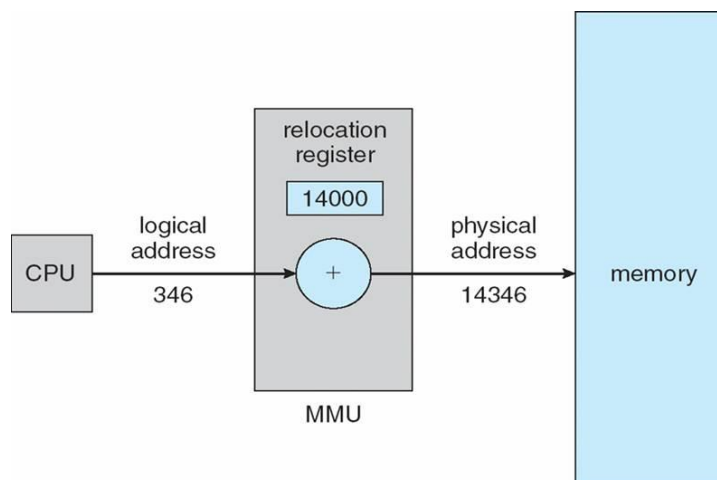
۱- **زمان کامپایل:** اگر فضای حافظه برنامه از قبل مشخص باشد می توان در زمان کامپایل آدرس ها را مشخص کرد که در این صورت اگر آدرس شروع حافظه برنامه تغییر کند، باید برنامه مجدداً کامپایل شود.

۲- **زمان بارگذاری:** در صورتی که در زمان کامپایل آدرس حافظه برنامه مشخص نباشد، باید برای آن برنامه کد قابل جابه جایی (Relocatable Code) تعریف شود (کدی که آدرس ها نسبت به آدرس ابتدای برنامه تعریف می شوند). یعنی اگر در آن امکان مشخص شده آدرس ۵۰ وجود دارد و برنامه هنگام بار شدن از آدرس ۶۰۰ در حافظه قرار گیرد، آدرس ۵۰ تبدیل به آدرس ۶۵۰ می شود.

۳- **زمان اجرا:** نگاشت زمان اجرا، همچنین به عنوان نگاشت پویا شناخته می شود، یک تکنیک مدیریت حافظه است که در آن نگاشت نهایی تا زمانی که برنامه واقعاً اجرا شود به تعویق می افتد. این امکان انعطاف پذیری و کارایی بیشتر در مدیریت حافظه را فراهم می کند. این نوع نگاشت در تکنیک حافظه مجازی استفاده می شود و نیاز به حمایت سخت افزاری با استفاده از واحد مدیریت حافظه (MMU) دارد.

مفاهیم آدرس فیزیکی و آدرس منطقی :

آدرس هایی که توسط CPU تولید می شود آدرس منطقی و آدرس هایی که به حافظه ارائه می شود، آدرس فیزیکی گفته می شود.



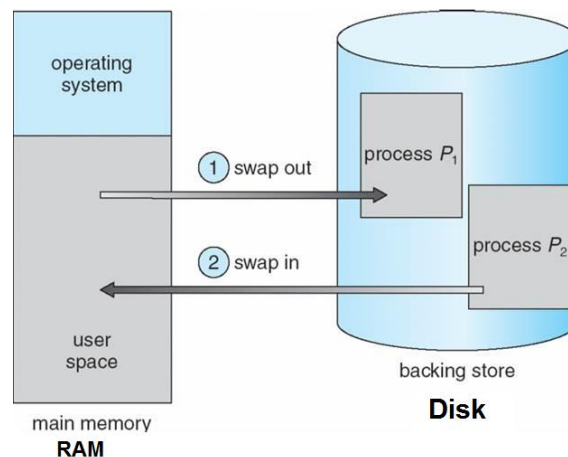
Memory Management Unit (MMU): سخت افزاری است که وظیفه نگاشت آدرس منطقی به فیزیکی را بر عهده دارد. این واحد می تواند در داخل بسته بندی پردازنده و یا بر روی مادربرد باشد.

آدرس های منطقی و فیزیکی در روش های نگاشت فضای حافظه زمان کامپایل و زمان بارگذاری یکی هستند اما در روش زمان اجرا با یکدیگر متفاوتند.

• بارگذاری پویا (Dynamic Loading)

در این روش یک روال (Procedure) تا زمانی که فراخوانی نشده به درون حافظه اصلی منتقل نمی شود.

• مبادله (Swapping): به معنای جابجایی بین RAM و Disk است.

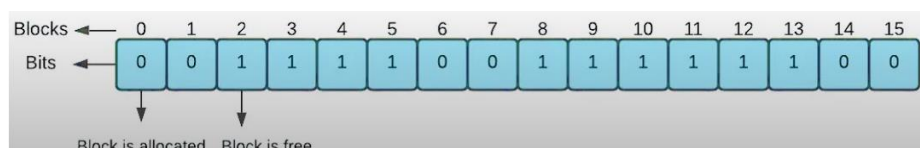


بر اثر عملیات مبادله ممکن است حفره های متعددی در حافظه به وجود آید که البته برای رفع این مشکل می توان در زمانهایی معین حافظه را فشرده سازی کنیم (فضاهای خالی را به هم بچسبانیم)

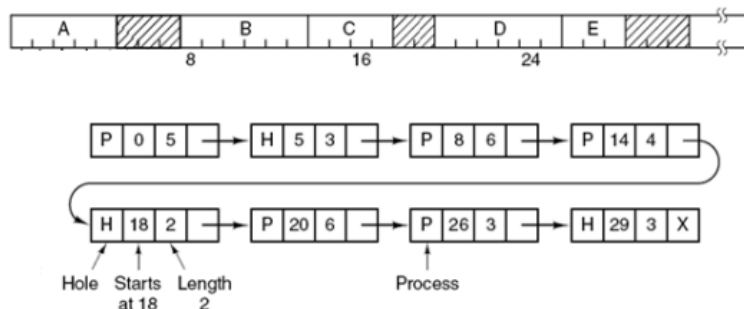
مدیریت فضای آزاد:

همانطور که در روش های ذکر شده بالا مشاهده گردید، مدیر حافظه در هر لحظه باید بداند کدام قسمت حافظه آزاد است و کدام قسمت استفاده شده است. برای این منظور می توان از دو روش نگاشت بیتی (bit maps) و روش لیست پیوندی (linked list) استفاده کرد.

الف) روش نگاشت بیتی: در این روش حافظه به چندین واحد تخصیص (Allocation unit) تقسیم می شود. اندازه این واحدها می تواند به کوچکی چندین کلمه یا به بزرگی چندین کیلو بایت باشد. متناظر با هر واحد تخصیص یک بیت در نگاشت بیتی وجود دارد. اگر واحدی استفاده شده باشد بیت متناظر آن ۱ شده و اگر آزاد باشد بیت متناظر آن ۰ می شود.



ب) روش لیست پیوندی: در این روش یک لیست پیوندی از قطعه‌های آزاد یا تخصیص یافته حافظه تشکیل می‌دهیم. هر گره یا یک پروسس است یا یک حفره که حرف H در اول گره نمایانگر حفره (Hole) و حرف P نمایانگر پروسس (Process) می‌باشد. هر عنصر لیست شامل آدرس شروع و طول هر بخش می‌باشد.



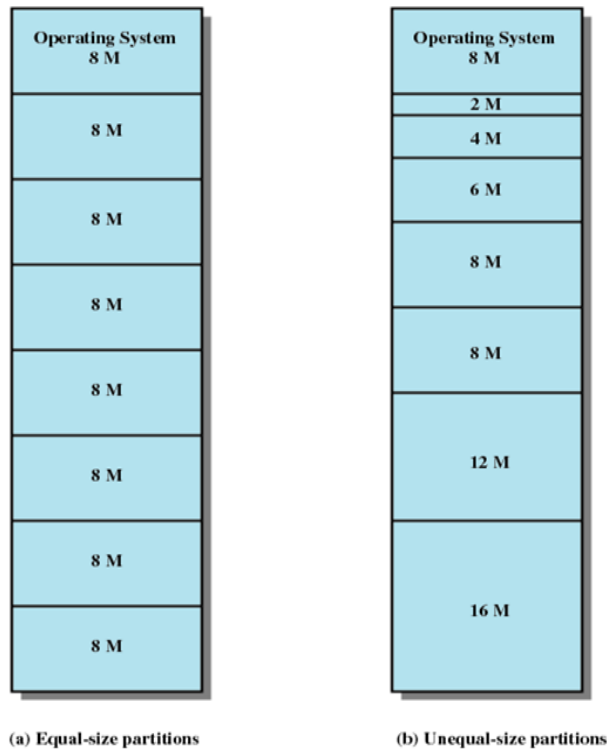
بخش بندی حافظه

حافظه باید برای استفاده همزمان توسط فرآیندهای همروند، بخش‌بندی شود. حافظه باید طوری بخش‌بندی شود که حداکثر تعداد فرآیند بتوانند از حافظه استفاده نمایند. بخش‌بندی می‌تواند به شیوه‌های مختلفی صورت گیرد.

• بخش بندی ایستا

a. با بخش‌های مساوی: در این روش حافظه به بخش‌های هم اندازه تقسیم می‌شود. هر فرآیندی می‌تواند در هر کدام از بخش‌های حافظه باشد. برنامه نویس باید محدودیت حافظه را هنگام برنامه نویسی در نظر بگیرد. اگر مجموعه حافظه فرآیندها از تمام حافظه بیشتر باشد باید برخی از آن‌ها را به دیسک منتقل کرد (swap out) مهم‌ترین مشکل این روش مشکل تکه تکه شدن داخلی است (Internal Fragmentation). اندازه فرآیندها باید کوچکتر یا مساوی از قسمت‌های حافظه باشد.

b. با بخش‌های نامساوی: در این روش حافظه به بخش‌های نامساوی تقسیم می‌شود و هر فرآیند می‌تواند در هر کدام از بخش‌ها قرار گیرد به شرطی که اندازه آن کوچکتر از آن بخش باشد. در این صورت انتخاب‌های بیشتری برای جابجایی فرآیندها در حافظه وجود دارد. در هر دو حالت فرآیند باید کامل و یکپارچه بارگذاری شود.



Example of Fixed Partitioning of a 64-Mbyte Memory

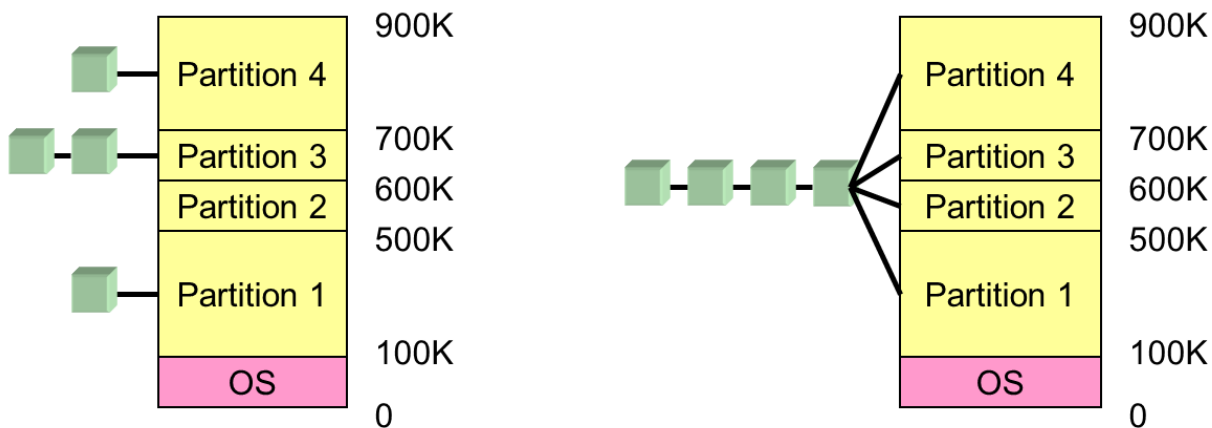
منظور از ایستا بودن ثابت بودن مرز بین بخشهای حافظه است.

الگوریتم‌های جایابی در بخش‌بندی ایستا

برای حالت بخش‌های مساوی (Equal-size partitions) به دلیل یکسان بودن اندازه همه بخش‌ها، فرقی نمی‌کند که کدام بخش برای هر فرآیند مورد استفاده قرار گیرد.

برای حالت بخش‌های نامساوی (Unequal-size partition) هر فرآیند را می‌توان به کوچکترین بخشی که بتواند فرآیند را در خود جای دهد، جایابی نمود. هدف کاهش مقدار تکه‌تکه شدن داخلی است.

ممکن است کل بخش‌ها یک صف داشته باشند یا هر دسته از بخش‌های هم اندازه، یک صف داشته باشند.



• بخش بندی پویا

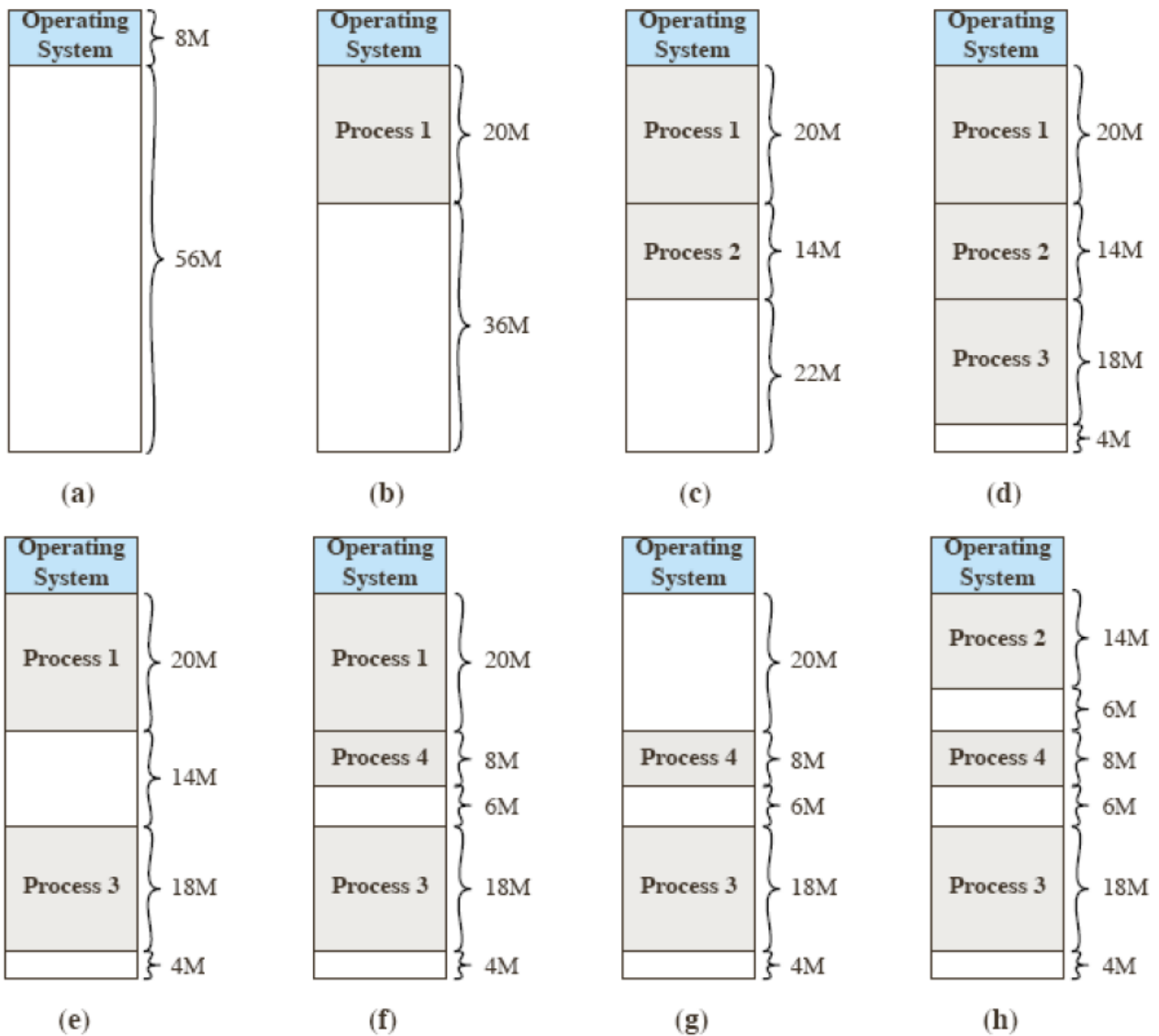
در ابتدای کار سیستم عامل، حافظه بخش بندی نشده است و در حین اجرای فرآیندها، به هر فرآیند هر میزان که حافظه بخواهد اختصاص می یابد. این روش مشکل تکه تکه شدن داخلی ندارد، اما مشکل تکه تکه شدن خارجی (External Fragmentation) دارد. برای رفع این مشکل می توان با فشردن بخش های خالی، حفره های ایجاد شده را حذف نمود که این کار بسیار زمان بر است.

به بلاک های آزاد حافظه، حفره (hole) می گویند. حفره ها در سراسر حافظه با اندازه های متفاوت پراکنده شده اند. وقتی یک فرآیند وارد می شود، فضای حافظه مورد نیاز آن از یکی از حفره های موجود که به اندازه کافی بزرگ است اختصاص می یابد. سیستم عامل اطلاعات مربوط به قطعه های اختصاص یافته و قطعه های آزاد (حفره ها) را نگهداری می کند.

جدول زیر دو نوع بخش بندی ایستا و پویا را مقایسه می کند.

ویژگی	بخش بندی پویا	بخش بندی ایستا
زمان تعیین اندازه و تعداد بخش ها	زمان اجرا	کامپایل یا بارگذاری
سادگی پیاده سازی	پیچیده	ساده
انعطاف پذیری	زیاد	کم
استفاده از حافظه	کارآمدتر	ممکن است ناکارآمد باشد
سربار سیستم عامل	زیاد	کم

تاثیرات بخش بندی پویا با یک مثال در شکل زیر نشان داده شده است.



"یعنی در اثر بارگذاری و خروج فرآیندهای با سایزهای مختلف امکان دارد حفره‌هایی در حافظه ایجاد شود که اگر چه ممکن است اندازه مجموع کل حفره‌ها از یک فرآیند بیشتر شود ولی نتوان آن فرآیند را بارگذاری کرد زیرا هیچ حفره‌ای با سایز مناسب پیدا نشود زیرا فرآیند باید بصورت یک پارچه بارگذاری شود."

✓ تخصیص همجوار: تمام بخش‌های یک فرآیند در مکان‌های مجاور هم در حافظه قرار می‌گیرند.

✓ تخصیص غیرهمجوار: یک فرآیند می‌تواند به صورت پراکنده در بخش‌های مختلف حافظه قرار گیرد.

- کل فرآیند لازم است در حافظه باشد (تکنیک های صفحه‌بندی و قطعه‌بندی)

- لازم نیست کل فرآیند در حافظه باشد (حافظه‌مجازی)

الگوریتم‌های جایگذاری برای بخش بندی پویا

۱- Best_Fit:

انتخاب حفره‌ای که پس از اختصاص آن کمترین فضای خالی بماند یا کوچکترین حفره‌ای که می‌توان اختصاص داد. در این حالت فضای خالی باقی مانده به احتمال زیاد به درد هیچ فرآیند دیگری نمی‌خورد.

۲- Worst_Fit:

حفره‌ای که بیشترین فضای خالی پس از اختصاص فرآیند باقی بماند انتخاب می‌شود. به امید این که فضای باقی مانده را بتوان برای سایر فرآیندها استفاده نمود.

۳- First_Fit:

فرآیند را در اولین حفره مناسب قرار بده (هر بار از بالا به پایین در حفره‌ها جستجو کن)

۴- Next_Fit:

مانند روش سوم است اما در دفعه‌های بعد از بالا به پایین جستجو انجام نمی‌شود بلکه از آخرین محل اختصاص یافته به بعد جستجو انجام می‌شود.

مثال:

► اگر بلوک‌های خالی حافظه بترتیب فوق باشند

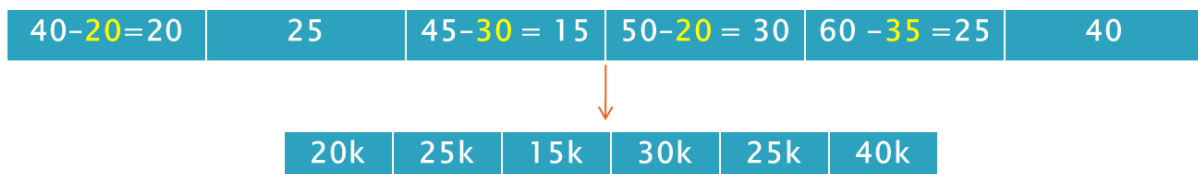
شروع حافظه →

40k	25k	45k	50k	60k	40k
-----	-----	-----	-----	-----	-----

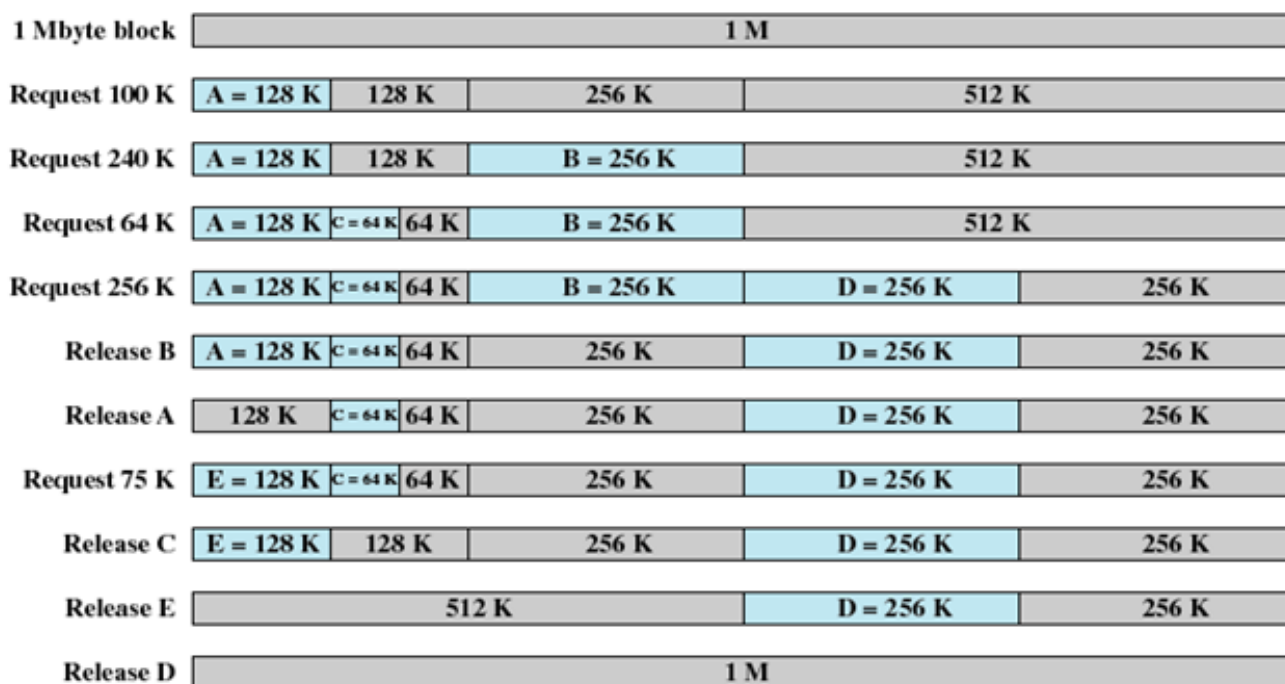
و درخواست‌های جدیدی برای چهار بلوک به اندازه 20k، 30k، 20k و 35k بترتیب از راست به چپ داده شود و از روش Next Fit استفاده گردد و در ابتدا تخصیص از اول حافظه شروع شود، وضعیت حافظه بعد از این تخصیص‌ها کدام گزینه است.

۱) 20k, 25k, 15k, 15k, 60k, 40k ۲) 5k, 25k, 25k, 20k, 40k, 40k

۳) 20k, 25k, 15k, 30k, 25k, 40k ۴) 10k, 5k, 15k, 50k, 60k, 5k



در این روش که ترکیبی از دو حالت قبلی است، حافظه به بخش‌هایی که سایز آن‌ها توانی از ۲ باشد تقسیم می‌شود. اگر بلوکی با اندازه مورد درخواست موجود باشد استفاده می‌گردد. و اگر بلوکی با اندازه مورد درخواست موجود نباشد بلوک‌های بزرگ‌تر با نصف شدن خرد می‌شوند تا سایز مورد نظر ایجاد شود. اگر دو بخش خالی کنار هم باشند در صورتی که هر دو از نصف شدن یک بلوک بزرگ‌تر ایجاد شده باشند با هم تلفیق می‌شوند.



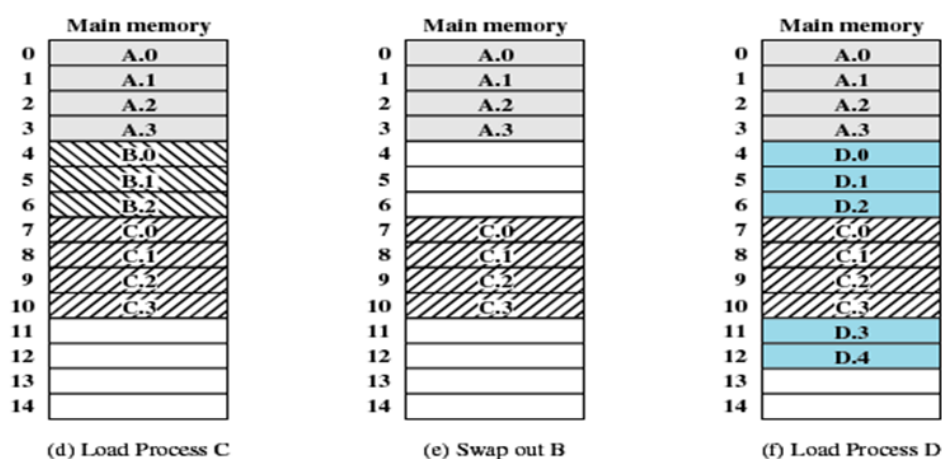
می‌توان فرآیندها را به صورت غیر همجوار نیز بارگذاری کرد که در این حالت از تکنیک‌های صفحه بندی و قطعه بندی استفاده می‌شود.

تکنیک صفحه بندی (Paging)

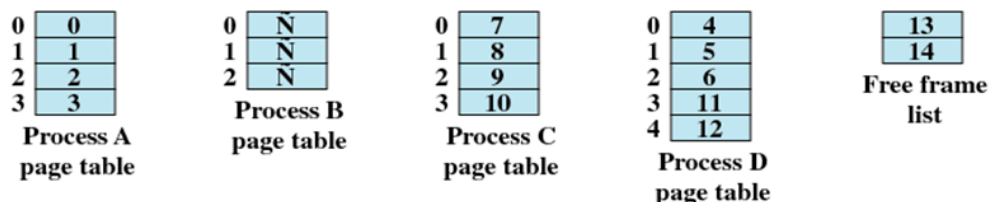
فضای اختصاص یافته به یک فرایند ممکن است یکپارچه نباشد. هر وقت فضای فیزیکی موجود بود به فرایند اختصاص داده می‌شود. در این روش حافظه فیزیکی به بلاک‌هایی با اندازه ثابت که قاب (frame) نامیده می‌شوند تقسیم می‌شود. اندازه هر قاب توانی از ۲ بین ۵۱۲ تا ۸۱۹۲ بایت است. حافظه منطقی به بلاک‌هایی با همان اندازه که صفحه (page) نامیده می‌شوند تقسیم می‌شود.

در این روش کل حافظه به بخش‌های کوچک با اندازه مساوی تقسیم بندی می‌شود و اندازه بخش‌ها به قدری کوچک است که هیچ فرآیندی در آن جا نمی‌شود و به هر فرآیند هر تعداد بخش که لازم باشد اختصاص داده می‌شود.

- مشخصات تمام قاب‌های آزاد نگه داری می‌شود.
- برای اجرای برنامه‌ای با اندازه n صفحه باید n قاب آزاد پیدا کرد و سپس برنامه را بارگذاری کرد. (فعلاً باید کل فرآیند در حافظه بار شود. بعداً در مبحث حافظه مجازی خواهیم دید که این کار لازم نیست.)
- از یک جدول صفحه (page table) برای تبدیل آدرس منطقی به فیزیکی استفاده می‌شود.
- مشکل قطعه قطعه‌گی داخلی وجود دارد ولی فقط در آخرین Page.



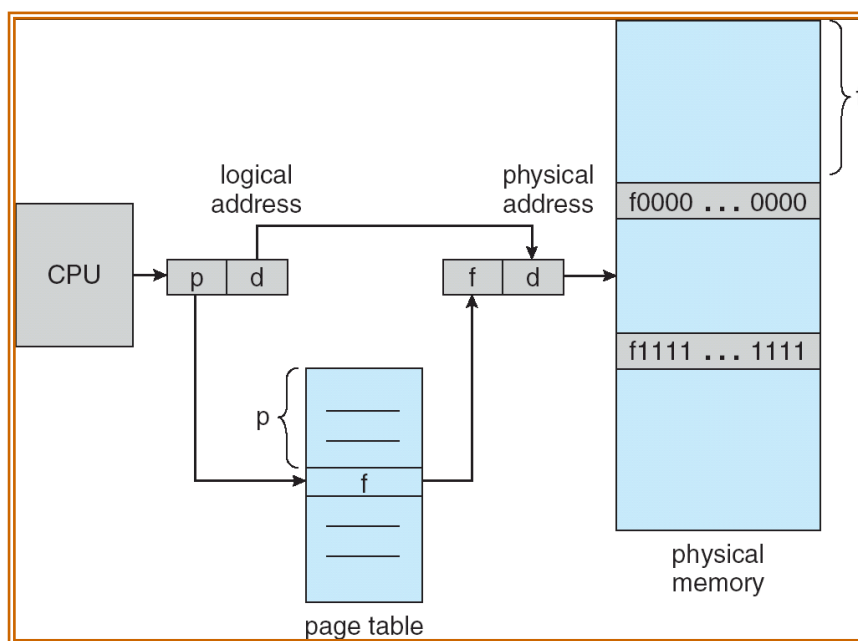
Assignment of Process Pages to Free Frames



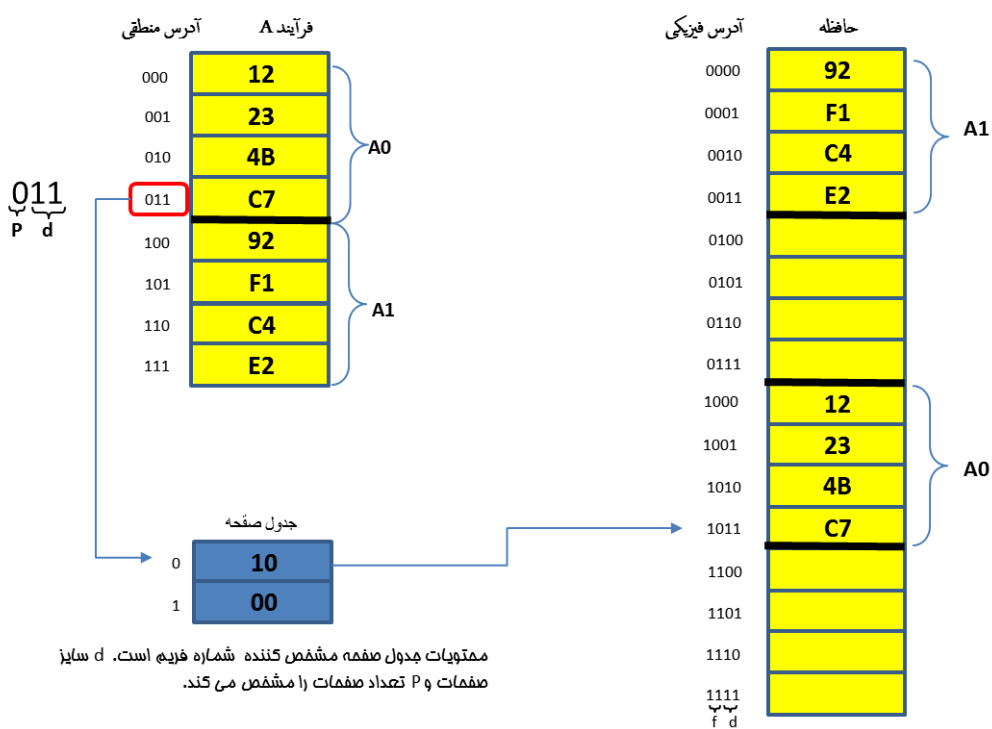
قطعات مربوط به یک فرآیند لزوماً به ترتیب یا همجوار در کنار هم قرار نمی‌گیرند. به ازای هر فرآیند یک جدول صفحه وجود دارد که مشخص می‌کند هر Page در کجا قرار دارد.

روش ترجمه آدرس

آدرس تولیدشده توسط پردازنده به دو قسمت شماره صفحه و آفست صفحه (displacement) تقسیم می شود. شماره صفحه به عنوان اندیسی استفاده می شود که به خانه ای از جدول صفحه اشاره می کند. جدول صفحه، آدرس شروع هر صفحه در فضای حافظه فیزیکی را دارد. آفست صفحه با آدرس شروع صفحه ترکیب می شود تا آدرس فیزیکی را که به حافظه اصلی ارسال می شود تشکیل دهد.



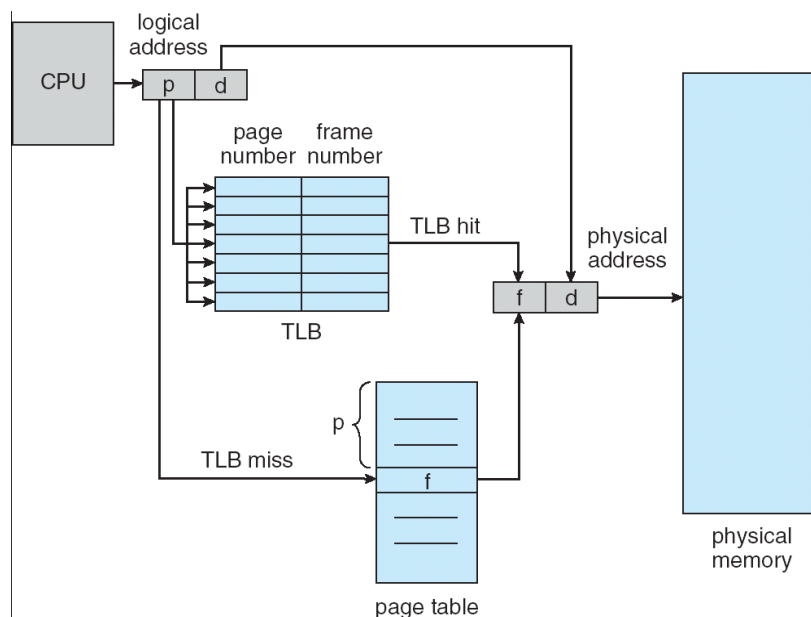
مثال:



یک فضای آدرس منطقی صفحه‌بندی، متشکل از ۳۲ صفحه ۲ کیلوبایتی را که به یک فضای آدرس یک مگابایتی نگاشت شده است. هر مدخل جدول صفحه باید چند بیتی باشد؟

الف) ۱۱ ب) ۹ ج) ۵ د) ۴

هر فرآیند جدول صفحه مربوطه به خود را دارد که باید در حافظه نگهداری شود. بنابراین هر آدرس مستلزم دوبار دسترسی به حافظه است. یک‌بار برای مراجعه به صفحه و بار دیگر برای مراجعه به حافظه مورد نظر و طبیعتاً این کار باعث کندی مراجعه می‌شود. به همین خاطر معمولاً از یک حافظه سریع ویژه به نام TLB (Translation Lookaside Buffer) استفاده می‌شود.

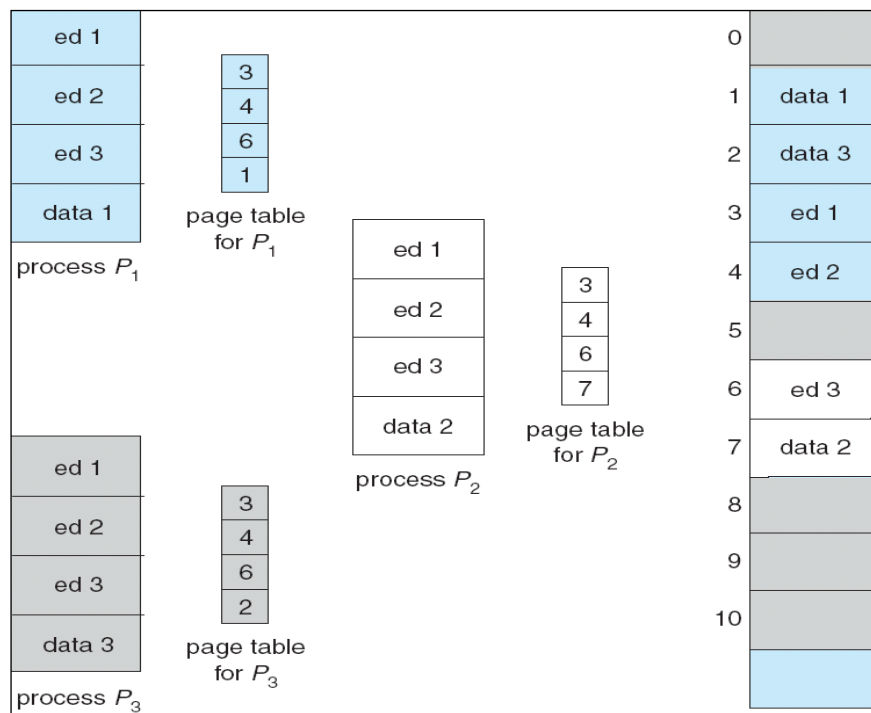


TLB از جنس حافظه Cache می باشد و hitRate آن معمولاً بالای 95% است. شماره صفحه مورد نظر همزمان هم به TLB و هم به جدول صفحه ارائه می‌شود. در صورتی‌که در TLB موجود باشد، شماره فریم را می‌توان سریع به دست آورد.

- نکته: اگر اندازه صفحات خیلی بزرگ باشد احتمال تکه تکه شدن داخلی (در صفحه آخر) بیشتر می‌شود. ولی در عوض اندازه جدول صفحه کوچک‌تر می‌شود. اگر اندازه صفحات کوچک باشند، Fragmentation داخلی کمتر ولی اندازه جدول صفحه بزرگ‌تر می‌شود.

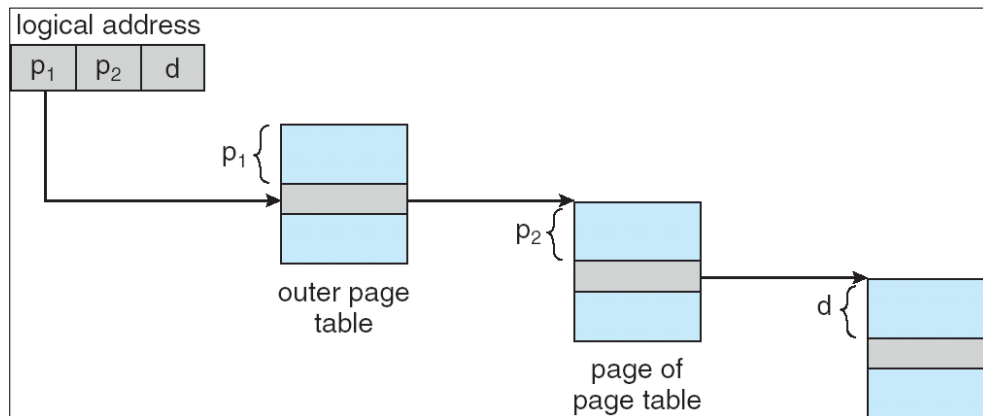
جدول صفحه می‌تواند علاوه بر شماره فریم شامل بیتی بنام **v** (valid) باشد. یک بیت اعتبار به هر ورودی در جدول صفحه مرتبط شده است. معتبر نشان دهنده این است که صفحه مرتبط در فضای آدرس منطقی فرآیند است. نا معتبر نشان دهنده این است که صفحه در فضای آدرس منطقی فرآیند نیست.

یک صفحه می‌تواند بین چند فرآیند مشترک باشد. مثلاً یک کپی از کد مشترک فقط خواندنی در میان فرآیندها (مثل پردازنده های متن، کامپایلرها و ...). کد مشترک باید در فضای آدرس منطقی تمام فرآیندها، در مکان یکسان باشد.



صفحه بندی سلسله مراتبی

در برخی موارد ممکن است سائز جدول صفحه بسیار بزرگ باشد. مثلاً در یک ماشین ۳۲ بیتی با اندازه صفحه 4KB دارای ۲^{۲۰} ورودی است، که اگر هر ورودی 4Byte باشد، اندازه جدول صفحه برای هر فرآیند 4MB خواهد بود و هر فرآیند احتیاج به جدول صفحه مربوط به خود دارد که در این صورت حجم زیادی از حافظه صرف جداول صفحات می‌شود. برای رفع این مشکل از صفحه بندی سلسله مراتبی استفاده می‌شود:



page number		page offset
p_1	p_2	d
10	10	12

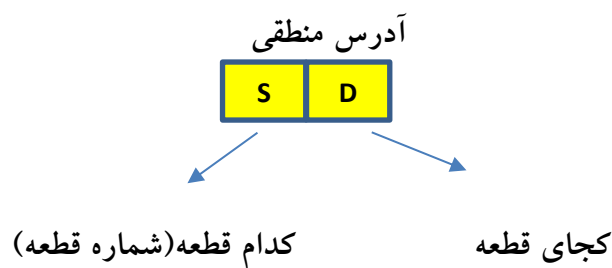
- فضای مورد نیاز در حالت تک سطحی: جدول صفحه دارای 2^{20} ورودی است.
- فضای مورد نیاز برای بارگذاری جدول صفحه ها هنگام استفاده از جدول صفحه سلسله مراتبی:

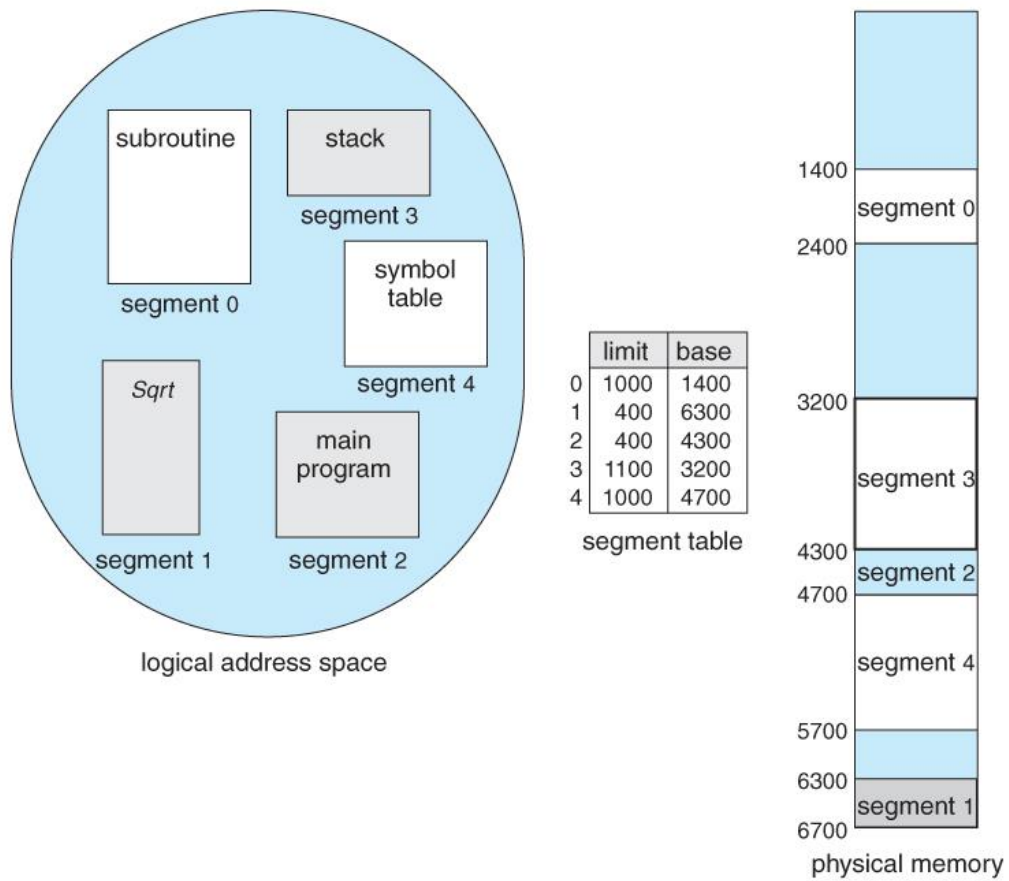
$$2048 = 2^{10} + 2^{10}$$

قطعه بندی (Segmentation)

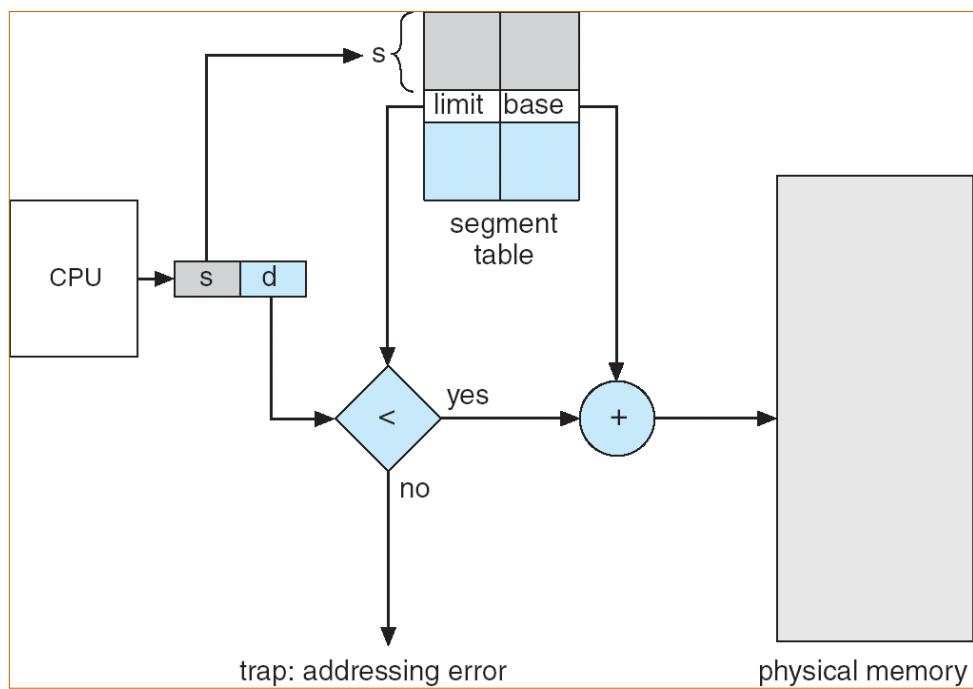
قطعه بندی طرحی برای مدیریت حافظه است که از دید کاربر در مورد حافظه پیروی می کند. یک برنامه مجموعه ای از قطعه ها است. یک قطعه یک واحد منطقی است مثل: برنامه اصلی - رویه - متد - متغیرهای محلی - استک

مشابه روش صفحه بندی در این حالت نیز احتیاج به جدولی موسوم به جدول قطعه داریم هر ورودی جدول قطعه شامل دو عنصر پایه (base) و حد (limit) است. (چون طول آن متغیر است پس از این دو عنصر استفاده می کنیم). آدرس منطقی نیز شامل دو قسمت شماره قطعه و آفست می باشد.





نحوه ترجمه آدرس منطقی به آدرس فیزیکی:



قطعه‌بندی صفحه‌بندی شده (Paged Segmentation)

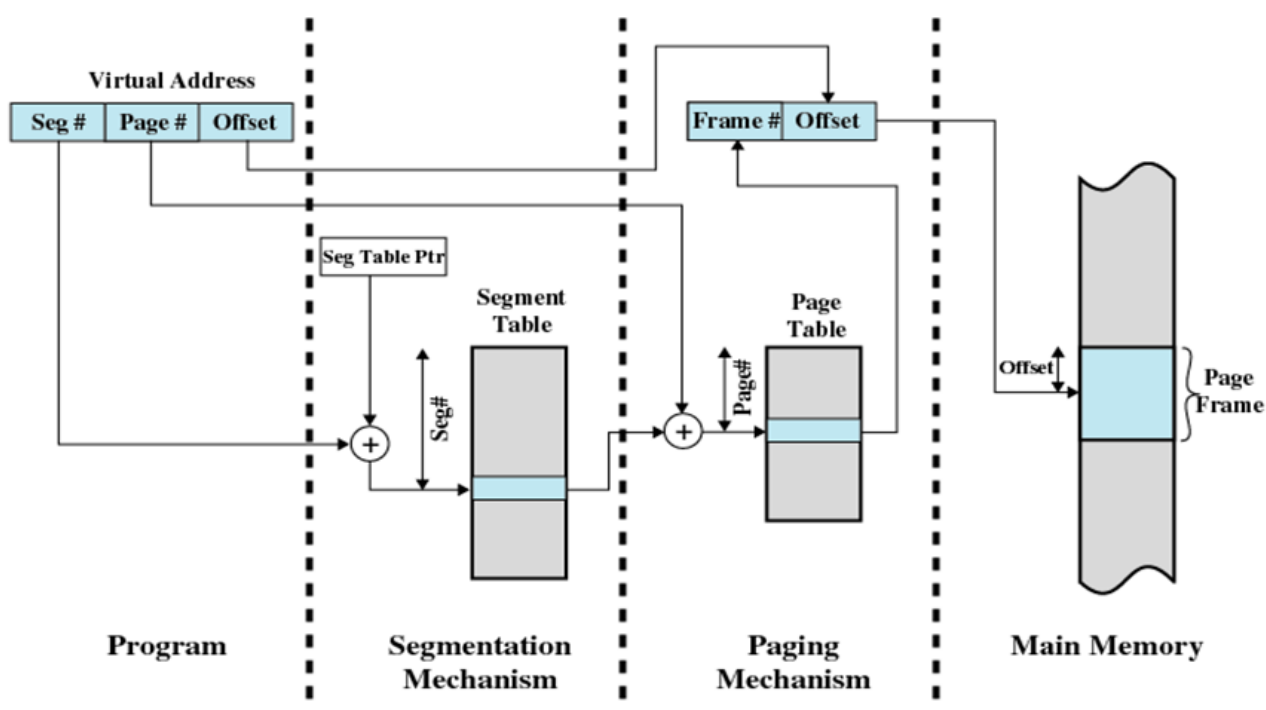
در این حالت هر قطعه بطور جداگانه صفحه‌بندی می‌شود. (قطعات به صفحات مساوی تقسیم بندی می‌شوند). برای هر قطعه در داخل جدول قطعه باید مشخص کنیم که جدول صفحه آن کجاست. در این حالت



آدرس منطقی شامل سه قسمت است:

D: Displacement P: شماره صفحه S: شماره قطعه

مکانیسم ترجمه آدرس در سیستم قطعه‌بندی صفحه‌بندی شده در شکل زیر نشان داده شده است.



Address Translation in a Segmentation/Paging System

Segment Table Pointer آدرس شروع جدول قطعه یک فرآیند را مشخص می‌کند.

محتویات جدول قطعه آدرس شروع جدول صفحه است.

هر قطعه جدول صفحه مخصوص به خود را دارد.

جدول زیر مزایا و معایب روشهای مختلف مدیریت حافظه را نشان می دهد:

معایب	مزایا	شرح	
استفاده غیر مؤثر از حافظه به دلیل تکه تکه شدن داخلی؛ تعداد ثابت فرایندهای فعال	سادگی پیاده سازی؛ کمی سربار سیستم عامل	حافظه اصلی به تعدادی بخشهای ایستا، در زمان ایجاد سیستم تقسیم می شود. یک فرایند می تواند به داخل بخشی با اندازه مساوی یا بزرگ تر بار شود.	روش بخش بندی ایستا
استفاده غیر مؤثر از پردازنده به دلیل نیاز به فشردگی برای جهت مقابله با تکه تکه شدن خارجی	بدون تکه تکه شدن داخلی؛ استفاده مؤثرتر از حافظه اصلی	بخشها به صورت پویا ایجاد می شوند، به طوری که هر فرایند به داخل یک بخش هم اندازه خودش بار می شود.	بخش بندی پویا
مقدار کمی تکه تکه شدن داخلی	بدون تکه تکه شدن خارجی	حافظه اصلی به تعداد قابهای هم اندازه تقسیم می شود. هر فرایند به تعدادی صفحه های هم اندازه با قابها تقسیم می گردد. یک فرایند از طریق بار کردن تمام صفحه های آن به داخل قابهای موجود که لزوماً پیوسته نیستند بار می شود.	صفحه بندی ساده
گسترش به کارگیری حافظه و کاهش سربار نسبت به بخش بندی پویا	بدون تکه تکه شدن داخلی	هر فرایند به تعدادی قطعه تقسیم می گردد. یک فرایند از طریق بار کردن تمام قطعه های در داخل بخشهایی که به صورت پویا به وجود آمده اند و لزوماً پیوسته نیستند به حافظه بار می شود.	قطعه بندی ساده
سربار پیچیدگی مدیریت حافظه	بدون تکه تکه شدن خارجی؛ درجه چند برنامه گی بالاتر؛ فضای آدرس مجازی بزرگ تر	مانند صفحه بندی ساده است با این تفاوت که نیازی نیست تمام صفحه های یک فرایند بار شوند. صفحه های غیرمقیم در زمانی که به آنها نیاز باشد به طور خودکار به حافظه آورده می شوند.	صفحه بندی حافظه مجازی
سربار پیچیدگی مدیریت حافظه	بدون تکه تکه شدن داخلی؛ درجه چند برنامه گی بالاتر؛ فضای آدرس مجازی بزرگ؛ حمایت از اشتراک و حفاظت	مانند قطعه بندی ساده است با این تفاوت که نیازی به بار کردن تمام قطعه های یک فرایند نیست. قطعه های غیرمقیم در زمانی که به آنها نیاز باشد به طور خودکار به داخل آورده می شوند.	قطعه بندی حافظه مجازی

۱

۲

۳

۴

۵

۶

۷

۸

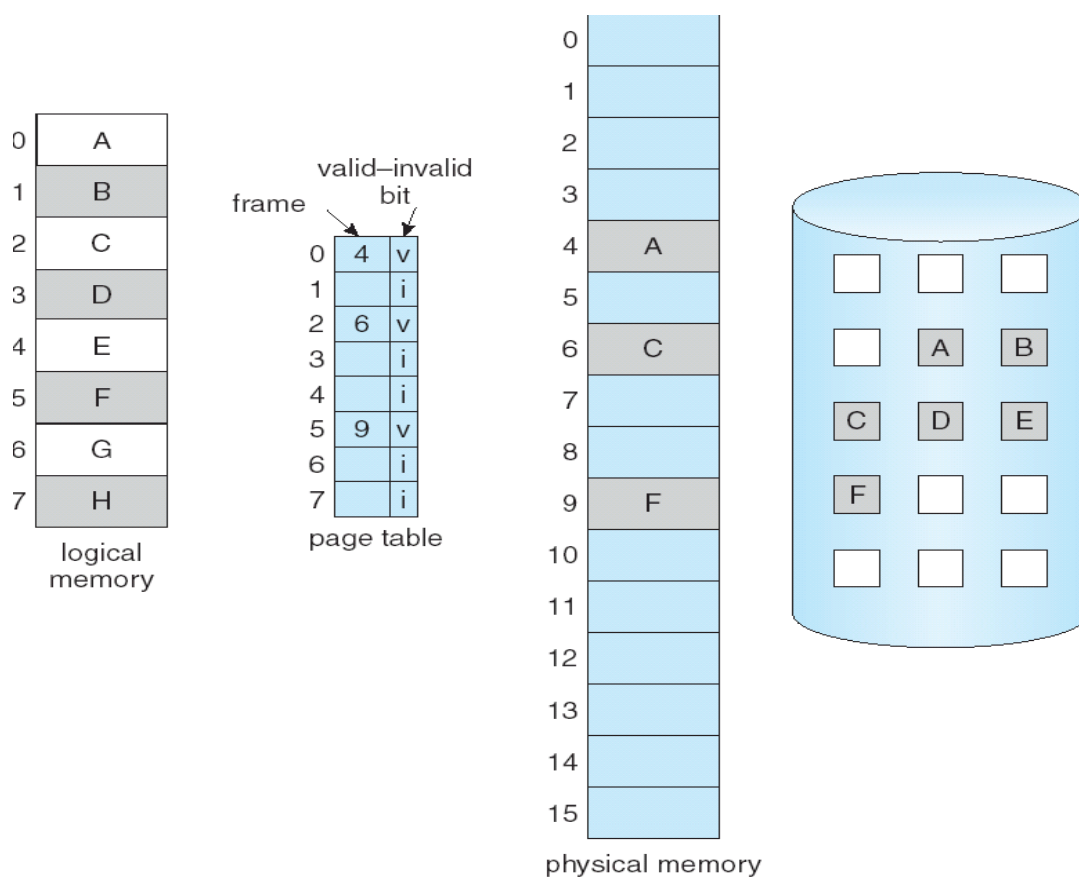
فصل هفتم

حافظه مجازی

حافظه مجازی یکی از مفاهیم مهم در سیستم عامل است. تا کنون فرض کردیم برای اجرای هر فرآیند باید تمام آن داخل حافظه باشد (فرض محدودکننده). مفهوم حافظه‌ی مجازی این فرض را نقض می‌کند و می‌گوید لازم نیست کل فرآیند در حافظه باشد. مزیت مهم این روش این است که اجازه می‌دهد سائز یک برنامه بزرگ‌تر از حافظه اصلی باشد. همچنین در این روش فرآیندهای بیشتری می‌توانند در حال اجرا باشند.

حافظه‌ی مجازی معمولاً با تکنیک صفحه بندی بر حسب تقاضا یا صفحه‌بندی نیازی (Demand paging) پیاده سازی می‌شود. این روش تلفیقی از تکنیک‌های صفحه بندی و مبادله می‌باشد؛ به این معنی که فقط صفحاتی که مورد نیاز باشند به حافظه آورده می‌شوند.

به هر مدخل جدول صفحه یک بیت اعتبار اختصاص داده می‌شود. $1 \leq$ در حافظه، $0 \leq$ بیرون از حافظه مقداردهی اولیه بیت اعتبار برای همه مدخل ها صفر است.



چه وقت صفحه‌ای به حافظه آورده می‌شود؟ زمانی یک صفحه به حافظه منتقل می‌شود که مورد نیاز باشد.

✓ نیاز به عملیات ورودی/خروجی کم‌تر

✓ نیاز به حافظه کم‌تر

✓ پاسخ سریع‌تر

✓ کاربران بیشتر

وقتی صفحه‌ای مورد نیاز باشد به آن ارجاع می‌شود.

✓ ارجاع به آدرس نادرست $\Leftarrow$ توقف

✓ فقدان صفحه در حافظه $\Leftarrow$ آوردن صفحه به حافظه (نقص صفحه)

نقص صفحه (page fault)

وقتی در حین اجرا آدرسی را لازم داشته باشیم که در حافظه نباشد (بیت اعتبار آن صفر باشد)، نقص صفحه رخ می‌دهد.

در هنگام نقص صفحه مراحل زیر انجام می‌شود:

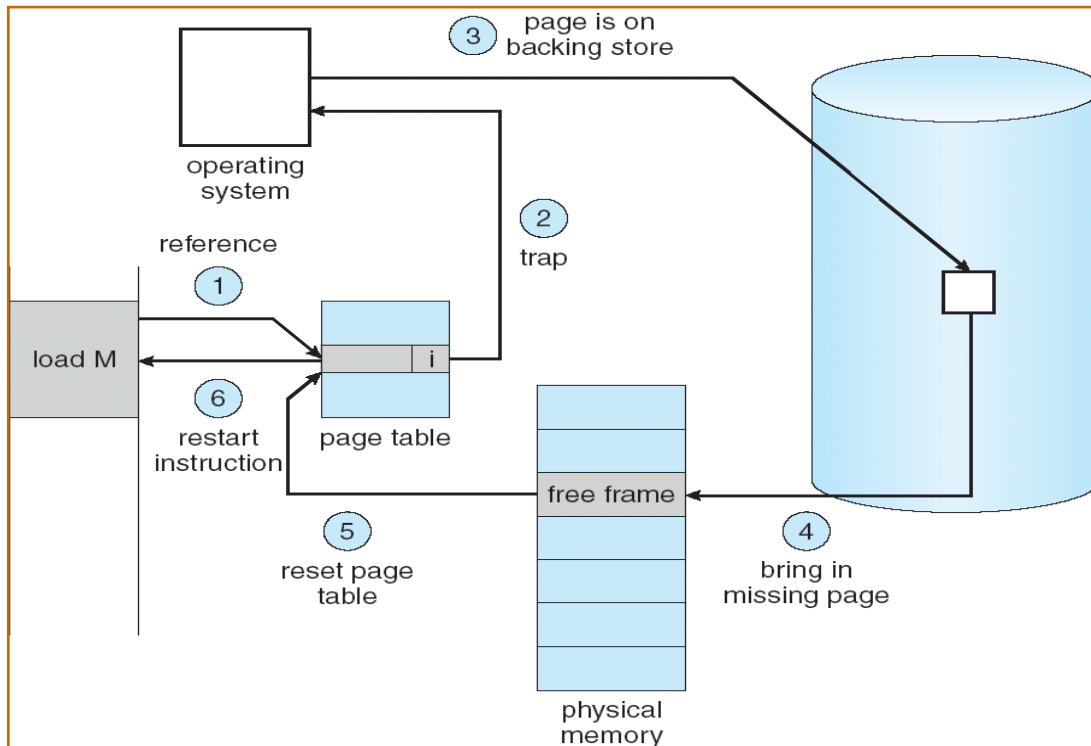
۱- سیستم عامل بررسی می‌کند که آیا این مراجعه در محدوده آدرس مجاز است یا خیر اگر آدرس مورد نظر خارج از محدوده مجاز باشد به فرآیند خاتمه می‌دهد و در غیر این صورت باید اقدام به آوردن صفحه مورد نظر نماید.

۲- یک فریم خالی در حافظه پیدا می‌کند.

۳- صفحه مورد نظر را از دیسک به حافظه منتقل می‌کند.

۴- جدول صفحه را تغییر می‌دهد تا نشان دهد صفحه مذکور در حافظه قرار دارد. (بیت اعتبار)

۵- اجرای دستوری که منجر به نقص صفحه شده بود را از سر می‌گیرد.



زمان دسترسی مؤثر به صفحه :

$$EAT = (1 - p) \times \text{memory access} + p \{ \text{page fault overhead} + [\text{swap page out}] + \text{swap page in} + \text{restart overhead} \}$$

سربار نقص صفحه : پیدا کردن فریم خالی و آوردن صفحه‌ای از دیسک یا خارج کردن یک صفحه از حافظه و وارد کردن صفحه‌ی مورد نیاز.

مثالی از روش تقاضای صفحه:

سربار زمان نقص صفحه: 8 milliseconds

زمان دسترسی به حافظه: 200 ns

متوسط زمان سرویس خطای صفحه

$$EAT = (1-p) \times 200 + p \times (8\text{milliseconds})$$

$$= (1-p) \times 200 + p \times 8,000,000 \text{ ns} = 200 + p \times 7,999,800$$

اگر در هر ۱۰۰۰ بار دسترسی به حافظه، یک خطای صفحه رخ دهد،

$$EAT = 8.2 \text{ microseconds}$$

✓ ۴۰ برابر کند شده است!

اندازه صفحه

▶ اگر اندازه هر صفحه خیلی بزرگ باشد:

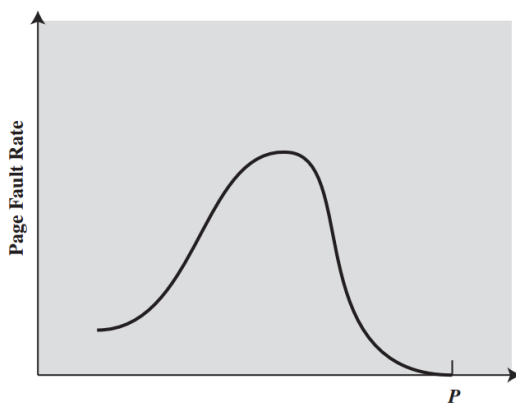
- تعداد صفحات کم شده و جدول صفحه کوچک می شود.
- اتلاف حافظه داخلی زیاد می شود.

▶ اگر اندازه هر صفحه خیلی کوچک باشد:

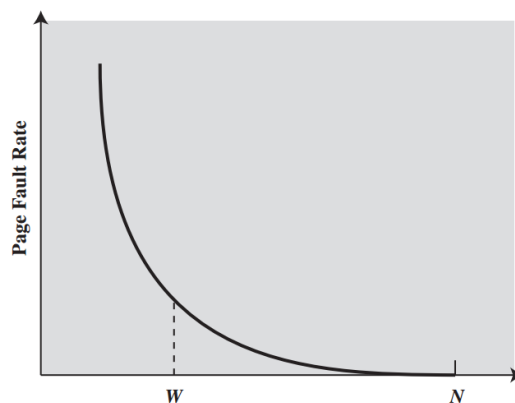
- تعداد صفحات زیاد شده و جدول صفحه بزرگ می شود.
- اتلاف حافظه داخلی کم می شود.

▶ یک مقدار میانی بهینه وجود دارد. با افزایش اندازه صفحه، اصل محلی بودن تضعیف شده و نرخ خطای صفحه شروع به افزایش می کند. اما اگر اندازه صفحه به اندازه کل فرآیند نزدیک شود، نرخ خطای صفحه کم می شود. برای یک اندازه ثابت صفحه، نرخ خطای صفحه با رشد تعداد صفحه های تخصیص یافته، کاهش می یابد.

اگر سایز صفحه کوچک باشد، تعداد زیادی از صفحات در حافظه اصلی هستند. در حین اجرا، صفحاتی که در حافظه قرار دارند شامل قسمتهایی از فرآیند هستند که اخیراً مورد ارجاع قرار گرفته اند. لذا خطاهای صفحه کم می شوند. اگر سایز صفحه بزرگ باشد، هر صفحه حاوی محلهایی است که از ارجاعات فعلی دور هستند. لذا خطاهای صفحه افزایش می یابند.



(a) Page Size



(b) Number of Page Frames Allocated

P = size of entire process
 W = working set size
 N = total number of pages in process

اندازه بهینه صفحه

▶ اگر اندازه میانگین برنامه‌ها، S بایت و اندازه یک صفحه P بایت و هر سطر جدول صفحه e بایت فضا لازم داشته باشد، تعداد سطرهای لازم برای یک برنامه در جدول صفحه برابر $\frac{S}{P}$ است و در نتیجه اندازه جدول صفحه $\frac{Se}{P}$

▶ برای یک برنامه نصف صفحه اتلاف وجود دارد $(\frac{P}{2})$

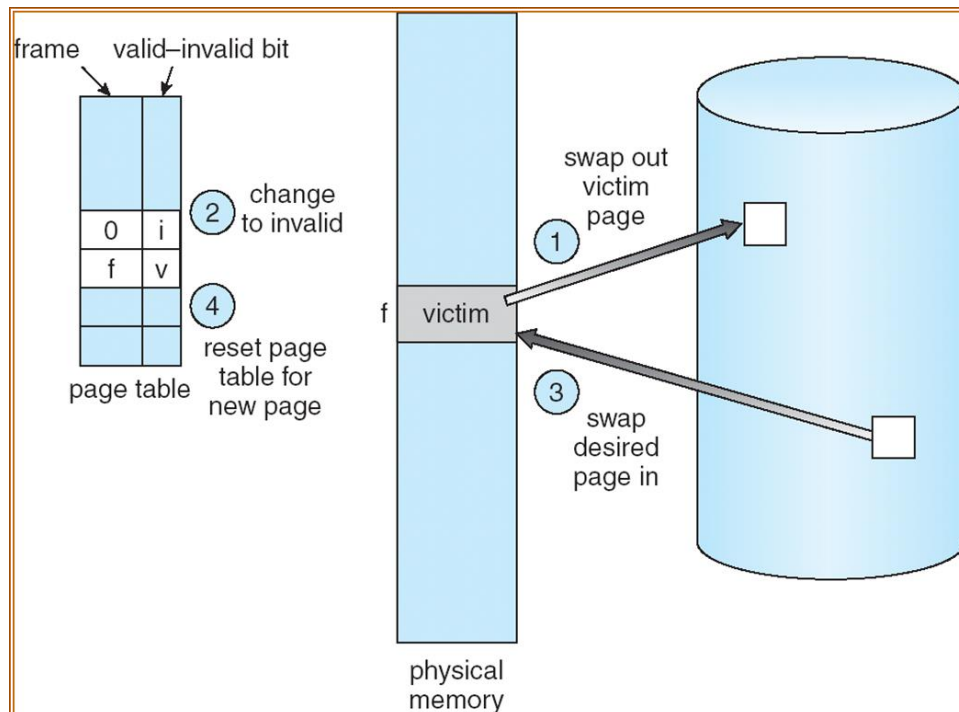
▶ فضای سربار هر برنامه $= \frac{Se}{P} + \frac{P}{2}$

▶ برای حداقل کردن این اتلاف باید مقدار آن نسبت به P مینیمم گردد. بنابراین با محاسبه مشتق این تابع و یافتن ریشه آن، P محاسبه می‌شود:

$$-\frac{Se}{P^2} + \frac{1}{2} = 0 \Rightarrow P = \sqrt{2Se}$$

اگر بخواهیم صفحه‌ای را به حافظه بیاوریم و قاب خالی وجود نداشته باشد چکار باید کرد؟

جابجایی صفحه انجام می‌دهیم یعنی صفحه‌ای را که در حافظه اصلی قرار دارد اما از آن استفاده نمی‌شود (اصطلاحاً صفحه "قربانی") یافته و آن را با صفحه مورد نظر جابجا می‌کنیم. برای این کار نیاز به الگوریتمی است که به کمترین تعداد خطای صفحه منجر شود. ممکن است یک صفحه چندین بار به درون حافظه منتقل شود.



الگوریتم‌های جایگزینی صفحات

ارزیابی با اجرای الگوریتم روی یک رشته ارجاع به حافظه و شمارش خطاهای صفحه رخ داده انجام می‌شود.
یک رشته ارجاع مثلا می‌تواند به صورت زیر باشد:

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

در ادامه تعدادی از الگوریتمهای جایگزینی صفحه توضیح داده شده‌اند.

✓ بهینه یا OPT (Optimal)

✓ "اخیرا کمترین استفاده شده" یا LRU (Least Recently Used)

✓ خروج به ترتیب ورود یا FIFO

✓ ساعت یا Clock

الگوریتم بهینه یا OPT (Optimal):

✓ برای جایگذاری، صفحه‌ای انتخاب می‌شود که در آینده کمتر مورد مراجعه قرار خواهد گرفت.

✓ نیاز به جستجو کل حافظه دارد.

✓ این روش در عمل قابل پیاده‌سازی نیست زیرا نیاز به پیشگویی آینده دارد و فقط به عنوان معیاری جهت مقایسه سایر الگوریتمها استفاده می‌شود.

مثال: اگر رشته ارجاع فرایندی به صورت زیر باشد و تعداد سه قاب برای این فرآیند اختصاص داده شده باشد، تعداد نقص صفحه‌ها را حساب کنید.

مجموعاً ۶ خطای صفحه ۳ تا در ابتدا و ۳ تا در حین کار

Page address
stream

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

OPT

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

الگوریتم «اخیرا کمترین استفاده شده» یا LRU:

- ✓ برای جایگذاری، صفحه‌ای انتخاب می‌شود که در گذشته دورتری مورد مراجعه قرار گرفته است.
- ✓ براساس اصل محلی بودن، اگر صفحه در گذشته کمتر مورد استفاده بوده است در آینده هم کمتر مورد استفاده خواهد شد.

مثال: برای مثال قبل تعداد نقص صفحه را به روش جایگزینی LRU بدست آورید.

Page address stream	2	3	2	1	5	2	4	5	3	2	5	2																																				
LRU	<table><tr><td>2</td></tr><tr><td></td></tr><tr><td></td></tr></table>	2			<table><tr><td>2</td></tr><tr><td>3</td></tr><tr><td></td></tr></table>	2	3		<table><tr><td>2</td></tr><tr><td>3</td></tr><tr><td></td></tr></table>	2	3		<table><tr><td>2</td></tr><tr><td>3</td></tr><tr><td>1</td></tr></table>	2	3	1	<table><tr><td>2</td></tr><tr><td>5</td></tr><tr><td>1</td></tr></table> F	2	5	1	<table><tr><td>2</td></tr><tr><td>5</td></tr><tr><td>1</td></tr></table>	2	5	1	<table><tr><td>2</td></tr><tr><td>5</td></tr><tr><td>4</td></tr></table> F	2	5	4	<table><tr><td>2</td></tr><tr><td>5</td></tr><tr><td>4</td></tr></table>	2	5	4	<table><tr><td>3</td></tr><tr><td>5</td></tr><tr><td>4</td></tr></table> F	3	5	4	<table><tr><td>3</td></tr><tr><td>5</td></tr><tr><td>2</td></tr></table> F	3	5	2	<table><tr><td>3</td></tr><tr><td>5</td></tr><tr><td>2</td></tr></table>	3	5	2	<table><tr><td>3</td></tr><tr><td>5</td></tr><tr><td>2</td></tr></table>	3	5	2
2																																																
2																																																
3																																																
2																																																
3																																																
2																																																
3																																																
1																																																
2																																																
5																																																
1																																																
2																																																
5																																																
1																																																
2																																																
5																																																
4																																																
2																																																
5																																																
4																																																
3																																																
5																																																
4																																																
3																																																
5																																																
2																																																
3																																																
5																																																
2																																																
3																																																
5																																																
2																																																

مجموعاً ۷ نقص صفحه. ۳ تا در ابتدا و ۴ تا در حین کار.

الگوریتم خروج به ترتیب ورود یا FIFO :

- ✓ برای جایگذاری، صفحه‌ای انتخاب می‌شود که زودتر وارد حافظه شده است.
- ✓ ساده است.
- ✓ اشکال این روش این است که تنها عمر صفحات مورد توجه قرار می‌گیرد نه میزان مراجعه به آنها.

Page address stream	2	3	2	1	5	2	4	5	3	2	5	2																																				
FIFO	<table><tr><td>2</td></tr><tr><td></td></tr><tr><td></td></tr></table>	2			<table><tr><td>2</td></tr><tr><td>3</td></tr><tr><td></td></tr></table>	2	3		<table><tr><td>2</td></tr><tr><td>3</td></tr><tr><td></td></tr></table>	2	3		<table><tr><td>2</td></tr><tr><td>3</td></tr><tr><td>1</td></tr></table>	2	3	1	<table><tr><td>5</td></tr><tr><td>3</td></tr><tr><td>1</td></tr></table> F	5	3	1	<table><tr><td>5</td></tr><tr><td>2</td></tr><tr><td>1</td></tr></table> F	5	2	1	<table><tr><td>5</td></tr><tr><td>2</td></tr><tr><td>4</td></tr></table> F	5	2	4	<table><tr><td>5</td></tr><tr><td>2</td></tr><tr><td>4</td></tr></table>	5	2	4	<table><tr><td>3</td></tr><tr><td>2</td></tr><tr><td>4</td></tr></table> F	3	2	4	<table><tr><td>3</td></tr><tr><td>2</td></tr><tr><td>4</td></tr></table>	3	2	4	<table><tr><td>3</td></tr><tr><td>5</td></tr><tr><td>4</td></tr></table> F	3	5	4	<table><tr><td>3</td></tr><tr><td>5</td></tr><tr><td>2</td></tr></table> F	3	5	2
2																																																
2																																																
3																																																
2																																																
3																																																
2																																																
3																																																
1																																																
5																																																
3																																																
1																																																
5																																																
2																																																
1																																																
5																																																
2																																																
4																																																
5																																																
2																																																
4																																																
3																																																
2																																																
4																																																
3																																																
2																																																
4																																																
3																																																
5																																																
4																																																
3																																																
5																																																
2																																																

از نظر منطقی باید با افزایش تعداد قابها تعداد نقص صفحات کاهش یابد. حال به مثال زیر توجه کنید.

مثال:

در یک سیستم حافظه مجازی صفحه بندی شده، فرآیندی به صفحات مربوط به خود طبق رشته زیر مراجعه می‌کند. اگر در حافظه سه قاب (Frame) خالی برای این فرایند موجود باشد و از الگوریتم جایگذاری صف یا FCFS استفاده شود، تعداد نقص صفحه ها (Page fault) چقدر خواهد بود؟ اگر چهار قاب خالی موجود باشد، تعداد نقص صفحه ها چه تغییری خواهد کرد؟

3, 2, 1, 0, 3, 2, 4, 3, 2, 1, 0, 4

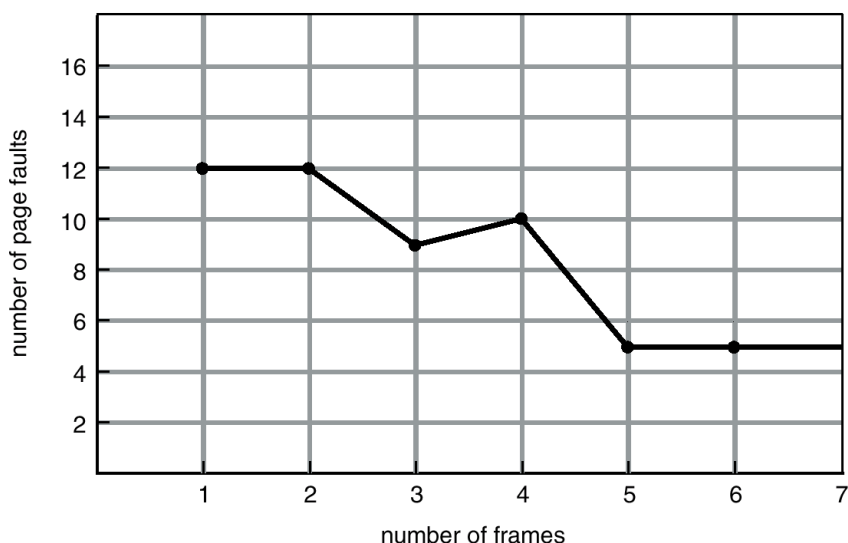
برای حالت ۳ قاب خالی: تعداد نقص صفحه‌ها = 9

3	2	1	0	3	2	4	3	2	1	0	4
3	3	3	0	0	0	4	4	4	4	4	4
	2	2	2	3	3	3	3	3	1	1	1
		1	1	1	2	2	2	2	2	0	0

برای حالت ۴ قاب خالی: تعداد نقص صفحه‌ها = 10

3	2	1	0	3	2	4	3	2	1	0	4
3	3	3	3	3	3	4	4	4	4	0	0
	2	2	2	2	2	2	3	3	3	3	4
		1	1	1	1	1	1	2	2	2	2
			0	0	0	0	0	0	1	1	1

همانطور که دیدید برای رشته ارجاعات فوق، با اضافه شدن تعداد قابها تعداد نقص صفحه بیشتر شد. این حالت فقط در الگوریتم FIFO رخ می دهد که به آن بی نظمی belady (belady's anomaly) می گویند.



الگوریتم جایگزینی صفحه دومین شانس:

با ایجاد یک تغییر کوچک در الگوریتم FIFO می تواند باعث جلوگیری از خروج صفحاتی شود که زیاد مورد استفاده قرار می گیرند. بیت ارجاع R قدیمیترین صفحه را بازرسی می نماییم و در صورتیکه این بیت صفر باشد صفحه هم قدیمی است و هم بلا استفاده. بنابراین با صفحه جدید جایگزین می شود. ولی اگر بیت R مربوطه یک باشد، آنرا صفر کرده و صفحه را به انتهای لیست منتقل می کنیم. سپس جستجو ادامه می یابد. مشکل اصلی این الگوریتم این است که صفحات در لیست باید جایجا شوند که این عمل دارای سربار زیادی است.

الگوریتم جایگزینی صفحه ساعت (The Clock Page Replacement Algorithm)

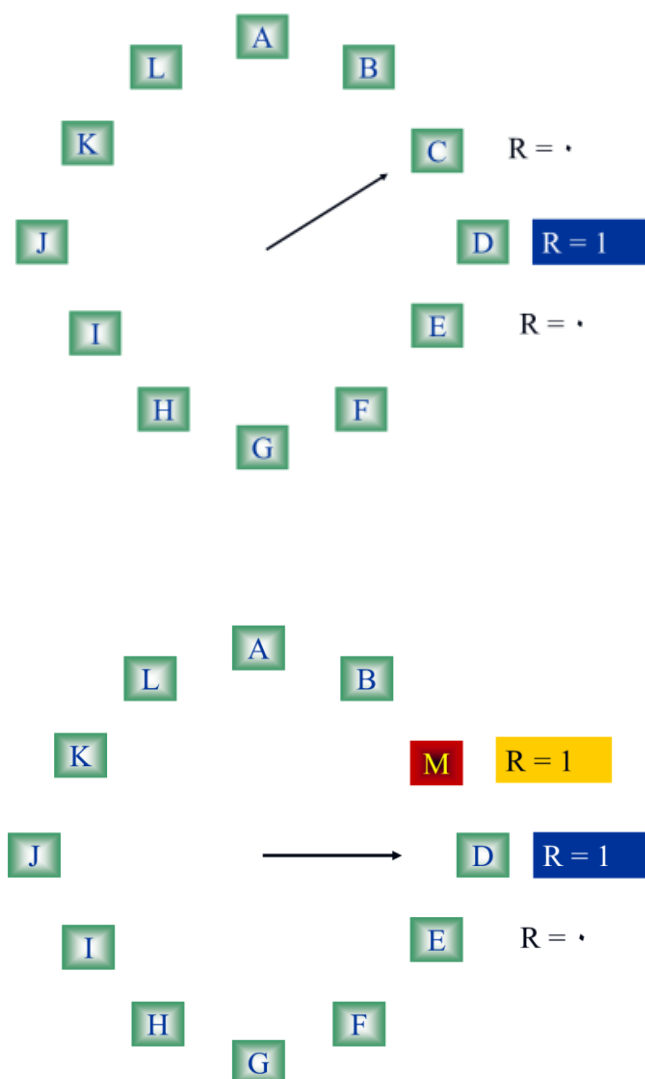
همان الگوریتم دومین شانس است با تفاوت در پیاده سازی (زیرا جابجایی صفحات در لیست پیوندی در دومین شانس زمان گیر بود) فقط به جای لیست پیوندی خطی از لیست پیوندی حلقوی استفاده می شود. در این الگوریتم به هر صفحه یک بیت اختصاص داده می شود. مطابق شرح ذیل:

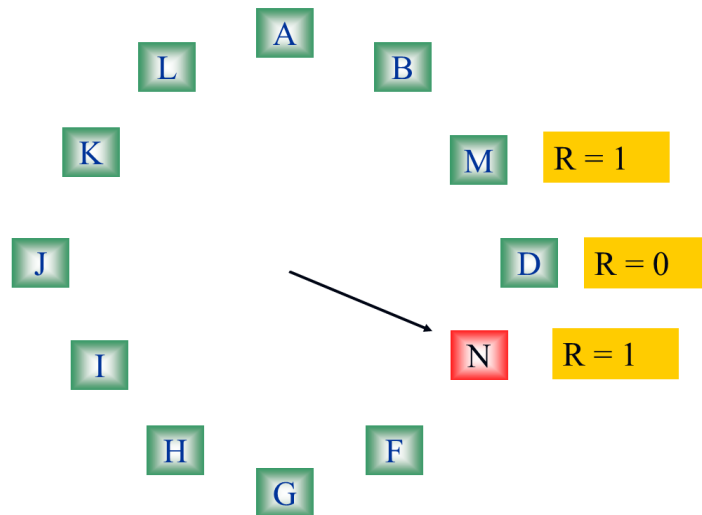
صفحات قرار گرفته در **page frame** ها (که کاندید اخراج هستند) در یک لیست حلقوی قرار می گیرند. عقربه به ابتدای لیست (قدیمی ترین صفحه) اشاره می کند.

وقتی یک نقص صفحه رخ می دهد، صفحه‌ای که عقربه به آن اشاره می کند مورد بررسی قرار می گیرد. اگر بیت $R=0$ بود، صفحه مورد نظر خارج شده و صفحه جدید در همان مکان جایگزین می شود. عقربه نیز یک قدم به جلو می رود (لذا صفحه بعدی در ابتدای لیست و صفحه جدید در انتهای لیست خواهد بود. این تغییرات فقط با تغییر یک اشاره گر (عقربه) بوجود آمده است.

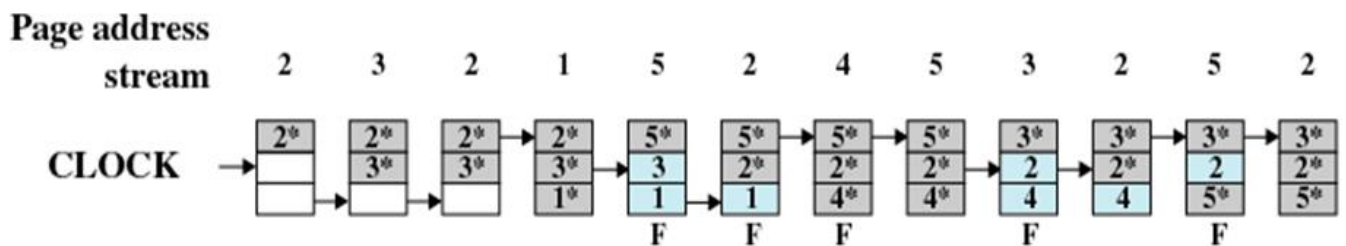
اگر $R=1$ بود آنگاه $R=0$ شده و عقربه یک قدم جلو می رود (یعنی آن صفحه به انتهای لیست ارسال می شود و صفحه بعدی در ابتدای لیست خواهد بود). این عمل آنقدر ادامه می یابد تا یک صفحه با بیت $R=0$ پیدا شود. تفاوت این الگوریتم با دومین شانس فقط در پیاده سازی آن است. فرضا صفحه جدید که قرار است به حافظه آورده شود M نام دارد:

شکلهای زیر مراحل کار این الگوریتم را نشان می دهند.





تعداد نقص صفحه ها را در مراجعه به رشته صفحات زیر با الگوریتم جایگذاری ساعت بدست آورید.



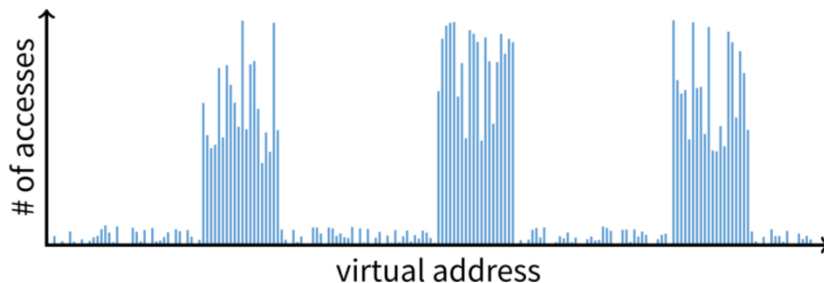
مجموعاً ۸ نقص صفحه. ۳ تا در ابتدا و ۵ تا در حین کار.

الگوریتم های شمارشی

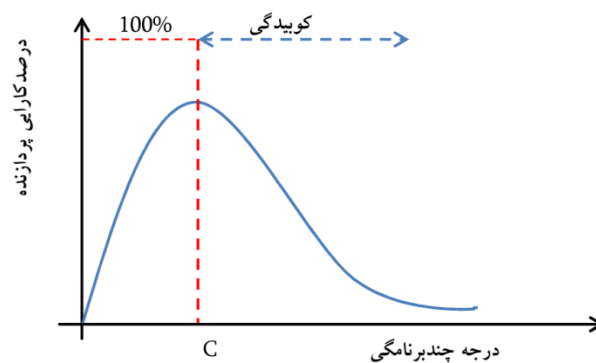
➤ الگوریتم **LFU (Least Frequently Used)**: صفحه ای را که کمترین شماره ارجاع را دارد جابجا می کند.

➤ الگوریتم **MFU (Most Frequently Used)**: با این استدلال که صفحه ای که کوچکترین شماره ارجاع را دارد احتمالاً به تازگی به حافظه منتقل شده است و هنوز استفاده خواهد شد، صفحه ای را که بیشترین شماره ارجاع را دارد جابجا می کند.

مجموعه کاری (working set)، به مجموعه صفحاتی می‌گویند که فرایند در حال حاضر از آنها استفاده می‌کند. به عبارتی دیگر، مقدار حافظه‌ای که یک فرایند در یک بازه زمانی خاص احتیاج دارد را مجموعه کاری آن فرایند می‌گویند. اگر تمام مجموعه کاری یک فرایند در حافظه موجود باشد، فرایند مورد نظر بدون ایجاد تعداد زیادی نقص صفحه اجرا خواهد شد.



در صورتیکه حافظه اصلی کوچک باشد و مجموعه کاری فرایند بزرگ باشد و نتوان بیکباره آن را در حافظه جای داد، فرایند به یکباره تعداد زیادی نقص صفحه ایجاد می‌کند. به طوری که فرایند بیشتر وقت خود را صرف تولید کردن نقص صفحه می‌کند تا اجرا شدن. در این حالت، کارایی پردازنده و کارایی کلی سیستم به شدت کاهش می‌یابد. به شرایطی که فرایند در هر چند دستورالعمل یک نقص صفحه ایجاد می‌کند، کوفتگی یا کوبیدگی (thrashing) می‌گویند که کارایی کلی سیستم را به شکل چشمگیری کاهش می‌دهد. در بسیاری از سیستم‌های صفحه‌بندی سعی بر این است که مجموعه کاری یک فرایند، از پیش شناخته شده و در حافظه بارگذاری شود تا فرایند مدام نقص صفحه ایجاد نکند. بدین ترتیب قبل از اختصاص دادن پردازنده به فرایند، مجموعه کاری اش در حافظه قرار دارد. اگر مجموعه کاری فرایند قبل از اختصاص پردازنده به آن در حافظه وجود نداشته باشد، فرایند مورد نظر آن قدر نقص صفحه ایجاد می‌کند تا بالاخره تمام مجموعه کاری اش در حافظه قرار بگیرد. (به شرطی که حافظه به میزان کافی فضا داشته باشد) مجموعه کاری یک فرایند در هر لحظه متفاوت است و با گذشت زمان تغییر می‌کند.



اثر درجه چندبرنامگی بر کارایی CPU، نقطه C بهترین سطح چندبرنامگی

۱

۲

۳

۴

۵

۶

۷

۸

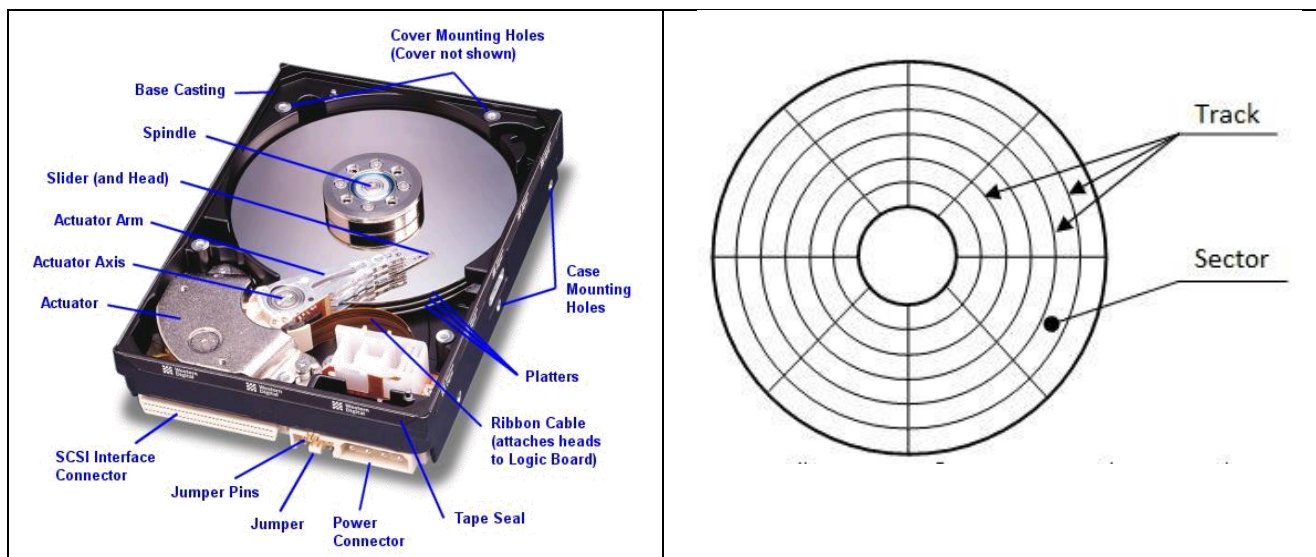
فصل هشتم

مدیرت دیسک



حافظه اصلی کامپیوتر فرار بوده و محتویات آن با قطع برق پاک می-شود. هارددیسک ابزاری برای نگهداری ثابت داده‌ها و برنامه‌ها (از جمله سیستم عامل) می‌باشد. سالها پیش دیسکهای سخت بسیار بزرگ و سنگین بودند و فقط برای محیطهای حفاظت شده مانند مرکز اطلاعات یا اداره‌های بزرگ مناسب بودند. شکل روبرو یک دیسک سخت با ظرفیت ۵ مگابایت را نشان می‌دهد.

هر دیسک از تعدادی صفحه تشکیل می‌شود که این صفحات به شکل مدور و مسطح می‌باشد و بر روی هر صفحه یک هد خواندن و نوشتن قرار دارد. سطح هر صفحه به طور منطقی به دوائر مرکزی به نام شیار تقسیم شده است. هر شیار خود به سکتورهای تقسیم می‌شود.



زمان دسترسی به یک دیسک

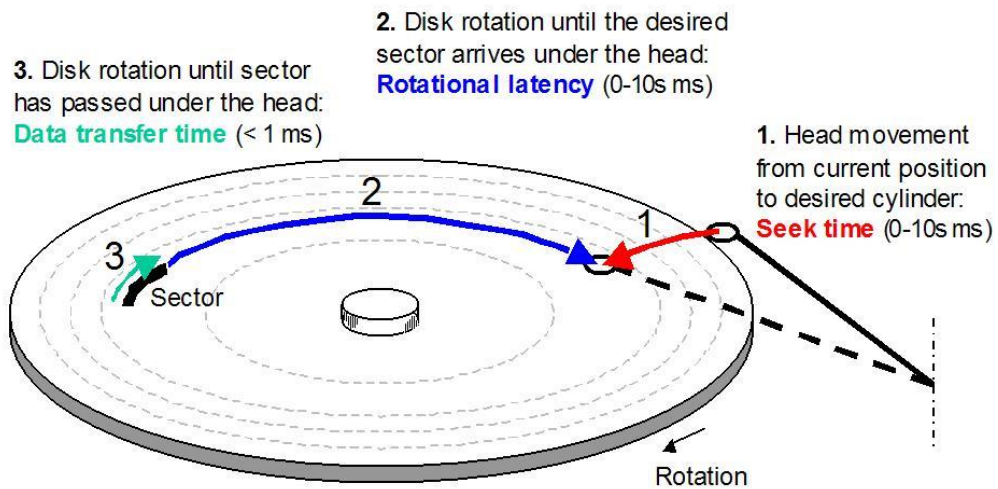
✓ زمان جستجو (seek time): همانطور که در شکل نشان داده است زمان لازم برای انتقال بازوی دیسک

(هد) به شیار (track) مورد نظر که الگوریتم‌های زمان بندی دیسک مربوط به این قسمت می‌باشد.

✓ زمان چرخش (Rotation Time): مدت زمانی که دیسک می‌چرخد تا هد روی سکتور مورد نظر قرار گیرد.

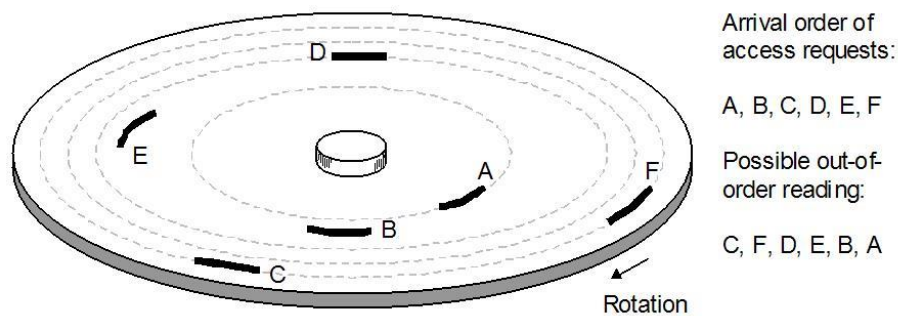
✓ زمان انتقال (Transfer Time): مدت زمانی که کل اطلاعات از دیسک خوانده می‌شود و جابجایی

اطلاعات صورت می‌گیرد.



الگوریتم های زمان بندی دیسک

سیستم عامل سعی می کند با توجه به ترتیب درخواست اطلاعات بر روی شیارها روشی را جهت بهینه کردن زمان دسترسی ارائه کند.

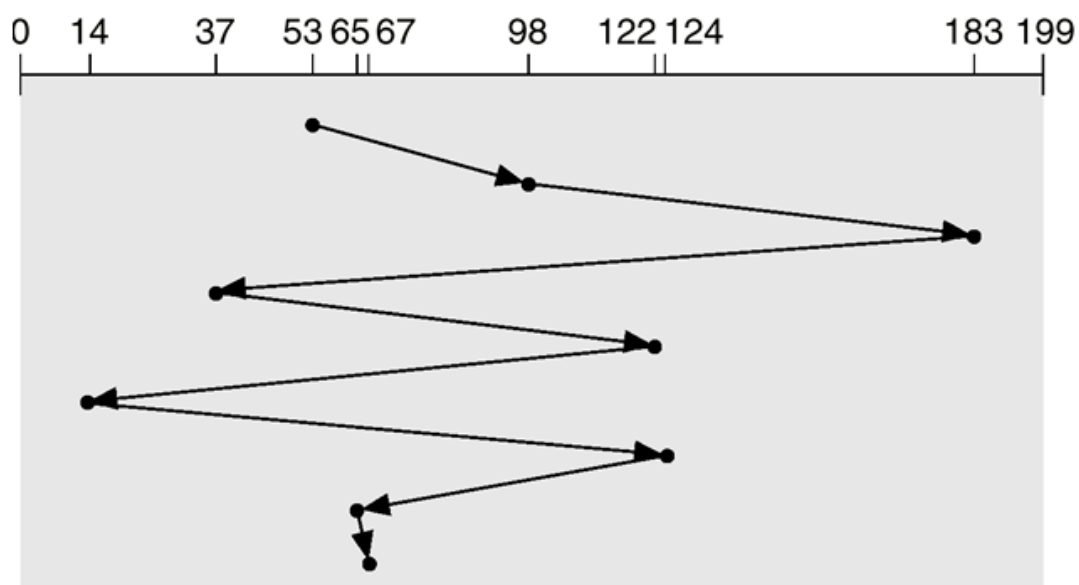


FIFO (الگوریتم خروج به ترتیب ورود):

در این روش درخواست ها با توجه به زمان ورودشان به صف، سرویس می گیرند. لذا نمی توان کاری برای بهینه کردن زمان دسترسی انجام داد.

مثال) فرض کنید شماره شیارهای درخواستی موجود در صف یک دیسک به صورت زیر باشد و در شروع کار هد بر روی شیار ۵۳ باشد. تعداد کل حرکت های هد را بدست آورید.

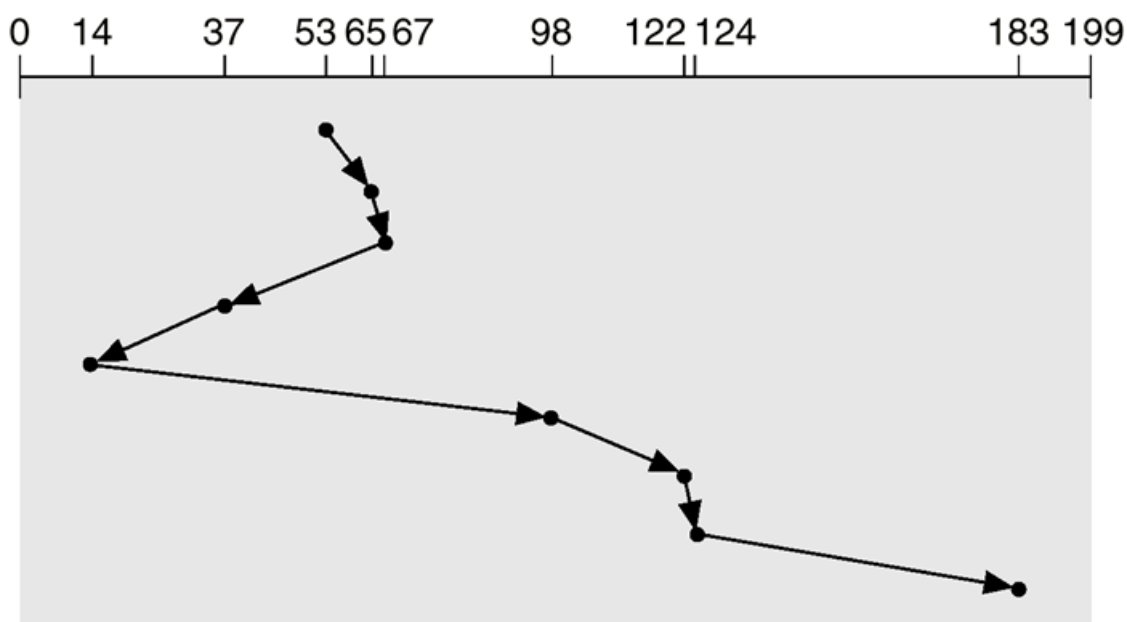
Queue: 98, 183, 37, 122, 14, 124, 65, 67



Total head movements= $45+85+146+85+108+110+59+2=640$

SSTF (الگوریتم ابتدا کوتاه ترین زمان جستجو)

در این روش به درخواست‌های موجود در صف بر اساس مکان قرار گیری پاسخ داده می‌شود. به گونه ای که در هر حرکت به درخواستی سرویس داده می‌شود که به هد نزدیک‌تر باشد.

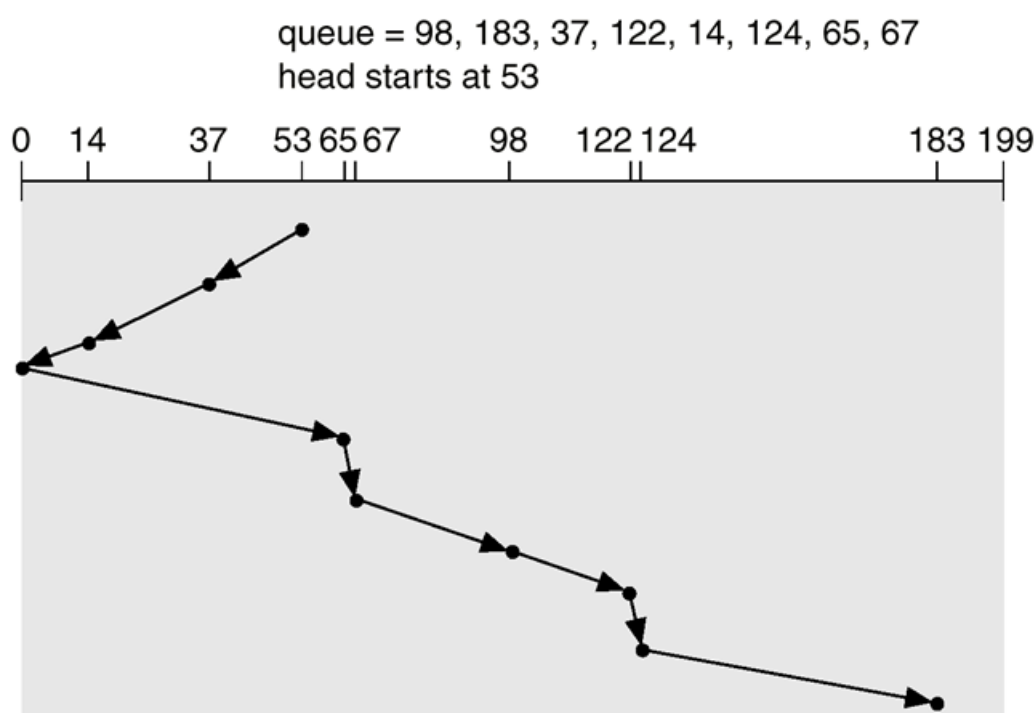


Total head movements= $12+2+30+23+84+24+2+59=236$

الگوریتم زمانبندی دیسک **SCAN** :

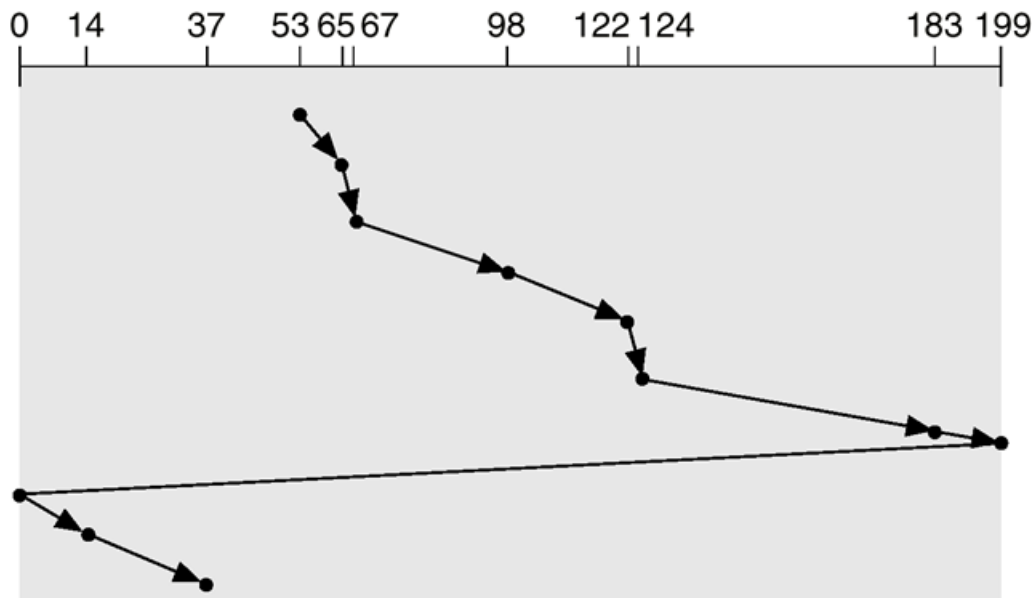
در روش **SCAN** (پویش) هد دیسک مرتباً از یک انتهای دیسک به سمت انتهای دیگر حرکت می کند و هر بار به سیلندری برسد که نیاز به سرویس دهی دارد، به آن سرویس می دهد.

در شروع این روش علاوه بر دانستن مکان جاری هد، باید جهت شروع حرکت هد را نیز بدانیم. الگوریتم **SCAN** به الگوریتم آسانسور نیز معروف است. چرا که مانند آسانسور یک ساختمان عمل می کند. در این روش اگر درخواستی به صف برسد که جلوی هد باشد، این درخواست سریعاً سرویس داده می شود ولی اگر در خواست پشت سر هد باشد، بایستی صبر کند تا هد به انتهای دیسک رفته، تغییر جهت داده و برگردد. بنابراین مشکل این روش آن است که تقاضایی که بلافاصله پس از عبور هد از یک سیلندر برای آن سیلندر دریافت می شود به تعویق می افتد.



الگوریتم زمانبندی **C-SCAN** :

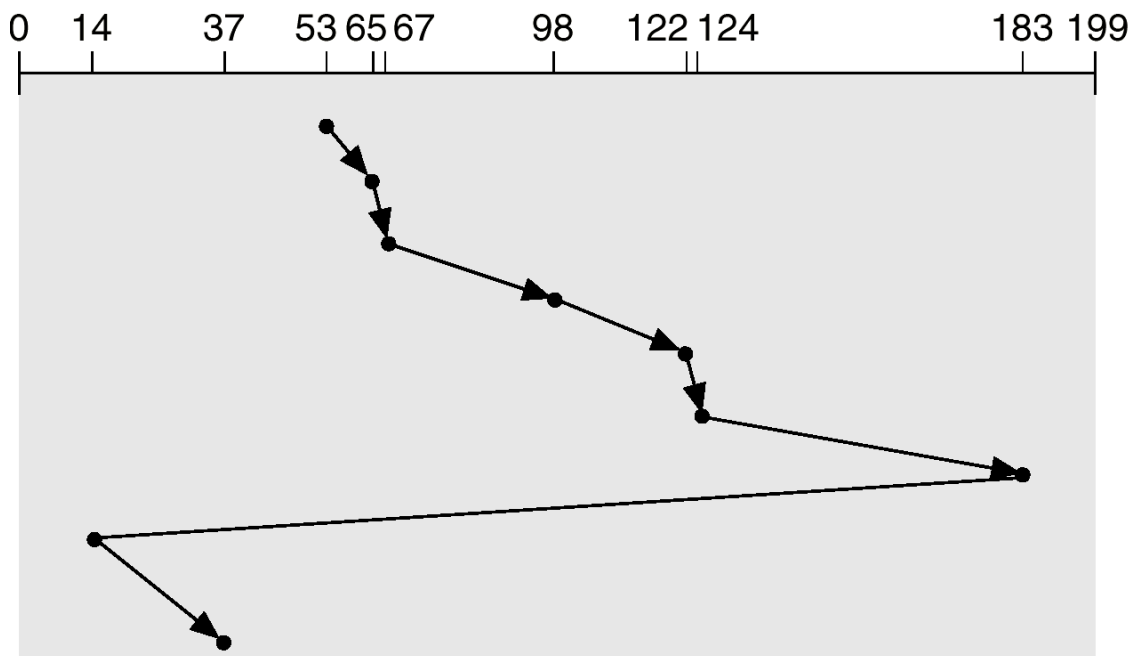
C-SCAN مخفف **Circular Scan** یا پویش چرخشی است. این روش نسبت به روش قبلی زمان یکنواخت کمتری را پدید می آورد. در روش **C-SCAN** مانند **SCAN** هد در یک جهت مثلاً از داخل به خارج حرکت کرده و در مسیر خود به تمام درخواست ها سرویس می دهد. ولی هنگامیکه به انتهای دیسک رسید سریعاً به اول دیسک بر می گردد و در این حرکت برگشتی سریع، هیچ سرویس دهی انجام نمی دهد.



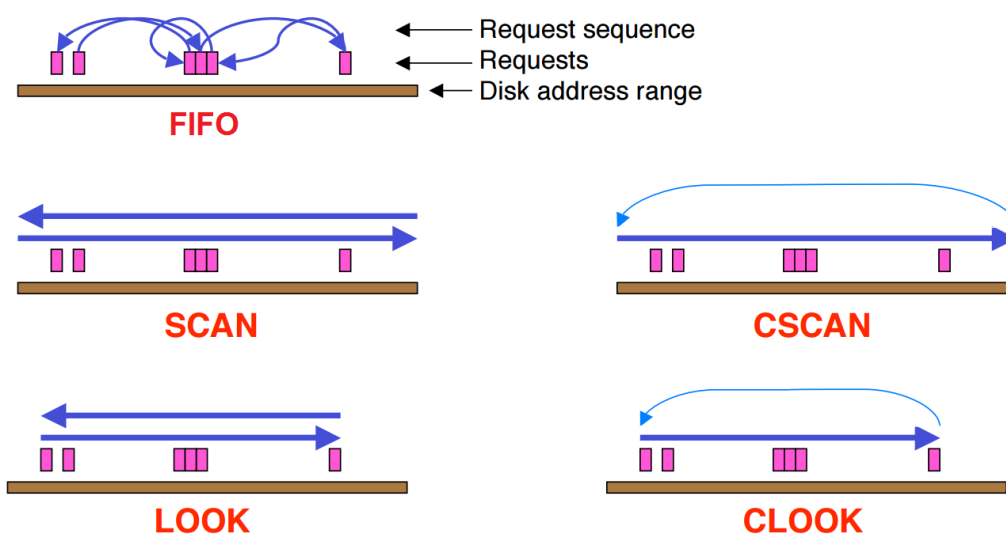
الگوریتم زمانبندی دیسک C-LOOK:

این روش اصلاح شده‌ی روش‌های SCAN و C-SCAN می‌باشد که در آنها الزاما حرکت از ابتدای دیسک شروع نمی‌شود و تا آخرین سیلندر نیز ادامه نمی‌یابد. بلکه از اولین درخواست شروع شده و به آخرین درخواست ختم می‌شود.

queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53



شکل زیرشمای کلی الگوریتمهای زمانبندی دیسک را بصورت دیگری نشان می دهد.



مثال:

فرض کنید یک دیسک ۲۰۰ شیار داشته و صف درخواست دیسک درخواست های نامنظمی را در خود دارد. شیارهای درخواست شده به ترتیب دریافت عبارتند از:

۵۵-۵۸-۳۹-۱۸-۹۰-۱۶۰-۱۵۰-۳۸-۱۸۴

در هریک از حالات زیر:

اگر زمان حرکت از یک شیار به شیار دیگر ۴ میلی ثانیه طول بکشد و بازوی دیسک در ابتدا بر روی شیار ۱۰۰ قرار داشته باشد، ترتیب سرویس دهی به درخواست ها و طول متوسط چقدر است؟

۱. FIFO
۲. SSTF
۳. LOOK
۴. C-LOOK

فرض کنید در روش LOOK, C-LOOK جهت اولیه حرکت به سمت افزایش شماره شیار باشد.

۱. FIFO

تعداد شیارهای های پیموده شده هنگام پاسخ دادن:

$$0+45+3+19+21+72+70+10+112+146=498$$

$$\text{Average Seek length} = 498/9=55.3$$

زمان جستجوی کل :

$$498*4(\text{ms})=1992\text{msec}$$

۲. SSTF

تعداد تراک های پیموده شده	شماره تراک مورد نظر	ترتیب سرویس
	۱۰۰	۰
۱۰	۹۰	۱
۳۲	۵۸	۲
۳	۵۵	۳
۱۶	۳۹	۴
۱	۳۸	۵
۲۰	۱۸	۶
۱۳۲	۱۵۰	۷
۱۰	۱۶۰	۸
۲۴	۱۸۴	۹

تعداد تراک های پیموده شده:

$$10+32+3+16+1+20+132+10+24=248$$

$$\text{Average seek time} = 248/9=27.5$$

زمان جستجوی کل:

$$248*4(\text{ms})=992\text{msec}$$

۳. LOOK

تعداد تراک های پیموده شده	شماره تراک مورد نظر	ترتیب سرویس
	۱۰۰	۰
۵۰	۱۵۰	۱
۱۰	۱۶۰	۲
۲۴	۱۸۴	۳
۹۴	۹۰	۴
۳۲	۵۸	۵
۳	۵۵	۶
۱۶	۳۹	۷
۱	۳۸	۸
۲۰	۱۸	۹

تعداد تراک های پیموده شده:

$$50+10+24+94+32+3+16+1+20=250$$

$$\text{Average seek length}=250/9=27.8$$

زمان جستجوی کل:

$$250*4(\text{ms})=1000\text{msec}$$

۴. C-LOOK

تعداد تراک های پیموده شده	شماره تراک مورد نظر	ترتیب سرویس
	۱۰۰	۰
۵۰	۱۵۰	۱
۱۰	۱۶۰	۲
۲۴	۱۸۴	۳
۱۶۶	۱۸	۴
۲۰	۳۸	۵
۱	۳۹	۶
۱۶	۵۵	۷
۳	۵۸	۸
۳۲	۹۰	۹

تعداد تراک های پیموده شده:

$$50+10+24+166+20+1+16+3+32=322$$

$$\text{Average seek time} = 322/9=35.8$$

زمان جستجوی کل:

$$322*4(\text{ms})=1288\text{msec}$$

RAID چیست ؟

کوتاه شده عبارت **Redundant Array of Independent Disks** و به معنای آرایه پشتیبان (افزونه) از دیسک‌های مستقل می‌باشد که هدف آن ایجاد یک واحد از مجموع چند هارد دیسک می‌باشد. در واقع با قرار دادن چند هارد دیسک در کنار هم و پیاده سازی RAID همه هارد دیسک‌های ما به یک واحد تبدیل می‌شوند و سیستم عامل همه آنها را فقط به عنوان یک منبع واحد (یک دیسک بزرگ) می‌بیند که بسته به اینکه چه سطحی از RAID پیاده سازی شده باشد می‌تواند باعث افزایش کارایی و یا امنیت اطلاعات و یا تلفیقی از این دو شود.

دیسک‌های سخت در سیستم‌های کامپیوتری سرعت پایینی دارند و به عنوان گلوگاه عمل می‌کنند. مخصوصاً در سرورها (مانند وب سرور) که باید دائماً سراغ دیسک برویم. هدف RAID افزایش کارایی، سرعت، قابلیت اطمینان و حجم دیسک است. این کار با استفاده از تکنیک‌هایی مانند **mirroring**، **striping**، اضافه کردن افزونگی برای تشخیص و تصحیح خطا و موازی سازی انجام می‌گردد.

استفاده از دیسک‌های کوچکتر به صورت موازی می‌تواند باعث افزایش حجم ذخیره‌سازی و سرعت دسترسی به اطلاعات شود اما نمی‌تواند احتمال خطا را کاهش دهد بلکه حتی احتمال آن بیشتر می‌گردد.

تمام دیسک‌های متصل به هم توسط یک کنترلر Raid مدیریت می‌شوند و فقط خود کنترلر می‌تواند آن‌ها را به - عنوان دیسک‌های مجزا ببیند اما سرور استفاده کننده فقط یک دیسک واحد را می‌بیند.

تکنولوژی Raid را می‌توان به دو شکل نرم افزاری و سخت افزاری پیاده سازی کرد.

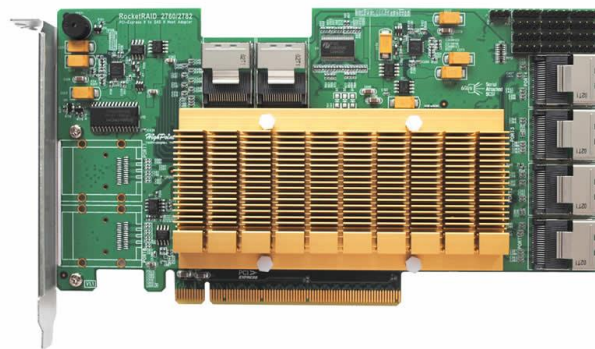
RAID سخت افزاری: بیشتر برای سرورهای سازمانی، تجاری و هنگامی که میزان تحمل خطا و بهینه شدن کارایی

سیستم بسیار مورد اهمیت است، مورد استفاده قرار می‌گیرد و نیاز به سخت افزار RAID Controller دارد. با RAID

سخت افزاری به دلیل اینکه عملیات پردازش بر عهده یک کنترلر RAID قرار می‌گیرد، بار اضافی به سرور تحمیل

نخواهد شد. RAID سخت افزاری طبیعتاً گران قیمت تر و پرهزینه تر از RAID نرم افزاری می‌باشد. در شکل زیر

یک بورد کنترل کننده RAID نشان داده شده است.



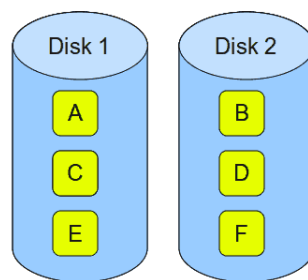
RAID نرم افزاری:

RAID نرم افزاری از طریق سیستم عامل تنظیم می‌شود و به طور ذاتی کارایی کمتری نسبت به کنترلرهای سخت افزاری RAID دارد. علت آن فقدان سخت افزار اختصاصی برای مدیریت آرایه‌های RAID است. RAID نرم-افزاری معمولاً برای مصارف خانگی و استفاده در محیط‌های کوچکی که حجم درخواست‌ها و تعداد کاربران کمتری دارند، استفاده می‌شود. مزیت اصلی Raid نرم افزاری نسبت به Raid سخت‌افزاری هزینه پایین آن می-باشد. یک مشکل دیگر Raid نرم افزاری سرعت کم آن نسبت به Raid سخت افزاری است.

سطوح مختلف RAID :

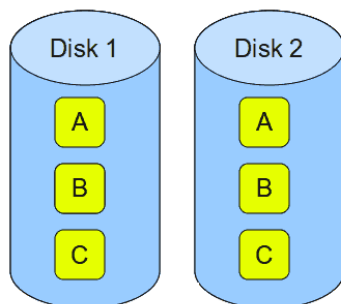
پیاده سازی RAID در سطوح مختلفی امکان پذیر است که در زیر شرح داده شده‌اند.

RAID0 (block by block striping): داده‌هایی که سرور می‌خواهد روی هارد دیسک بنویسد را به صورت بلوک به بلوک روی هارد دیسک‌های فیزیکی به ترتیب می‌نویسد. به عنوان مثال اگر A ، B ، C ، D ، E و ... بلوک‌های مورد نظر ما برای نوشتن روی دیسک باشد کنترلر ابتدا بلوک A را روی اولین دیسک فیزیکی می‌نویسد و سپس بلوک B را روی دیسک فیزیکی دوم می‌نویسد تا نوبت به آخرین دیسک می‌رسد و پس از آن دوباره کنترلر سراغ دیسک فیزیکی اول می‌رود و این روال تکرار می‌شود تا تمام بلوک‌ها در حافظه نوشته شوند. در این نوع Raid مبادله داده‌ها با کنترلر بسیار سریع‌تر از عملیات خواندن/نوشتن روی دیسک چرخان می‌باشد. به عنوان مثال زمانی که کنترلر مشغول نوشتن بلوک A روی اولین دیسک است، همزمان بلوک B را به دیسک سخت دوم ارسال می‌کند. Raid0 باعث افزایش عملکرد می‌شود اما تحمل خرابی پایینی دارد چون اگر یک دیسک سخت از بین برود همه داده‌های روی دیسک مجازی از بین می‌رود.



• **RAID 1 (block by block mirroring):** هدف آن افزایش تحمل در برابر خرابی می باشد. شیوه کار

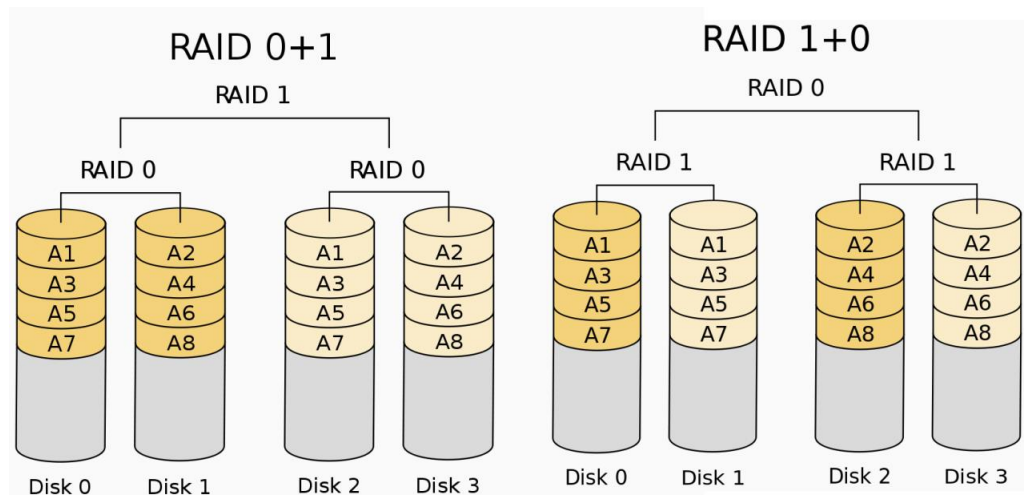
این تکنولوژی به این صورت است که حداقل در حالت پایه نیاز به دو دیسک سخت در قالب یک دیسک مجازی دارد. هنگامی که سرور یک بلوک روی دیسک مجازی می نویسد کنترلر این بلوک را روی هر دو هارد دیسک می نویسد. به هر یک از این کپی های یکسان **mirror** گفته می شود و در بعضی از موارد هم سه نسخه از هر بلوک نگهداری می شود.



RAID 0+1 / RAID 10: ترکیبی از تکنیک های Striping و Mirroring

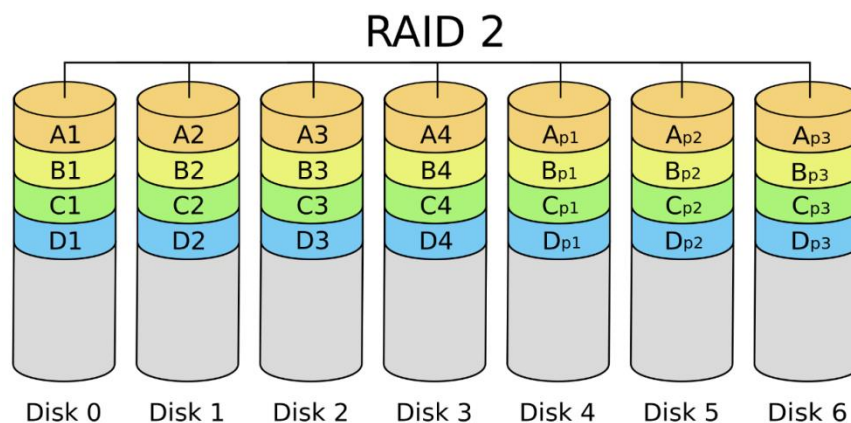
Raid 0 و Raid 1 یکی از موارد تحمل پذیری در برابر خطا و یا کارایی را افزایش می دهد اما اگر بتوان این دو را در کنار هم داشت به وضعیت مطلوب تری می رسیدیم. Raid 0+1 و Raid 10 این کار را برای ما انجام می دهند به این صورت که ایده های Raid 0 و Raid 1 را با هم ترکیب می کنند. Raid 0+1 و Raid 10 هر دو سلسله مراتب مجازی دو مرحله ای را انجام می دهند.

در Raid 0+1 کنترلر Raid در ابتدا به وسیله Raid 0 تمام دیسک ها را در قالب دو دیسک سخت مجازی که تنها برای کنترلر دیسک قالب مشاهده است ترکیب می کند (عمل Striping). در سطح دوم این دو دیسک سخت مجازی در قالب یک دیسک سخت مجازی واحد با استفاده از تکنیک Mirroring در Raid 1 با هم ادغام می گردند و تنها این دیسک مجازی واحد برای سرور قابل مشاهده است. در Raid 10 مراحل اجرایی Raid 0 و Raid 1 نسبت به Raid 0+1 معکوس می گردد.



• RAID 2 (striping of bytes or bits with ECC)

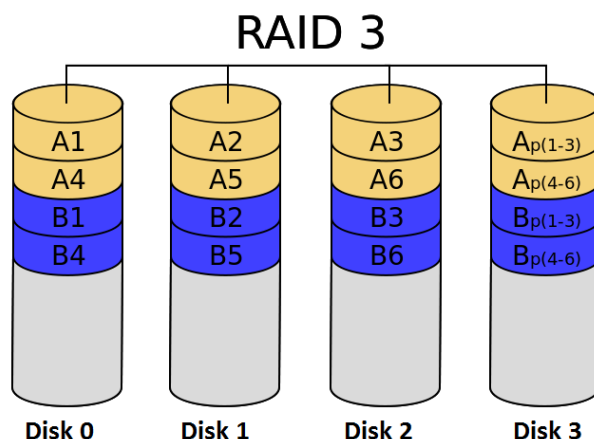
این نوع RAID با تقسیم اطلاعات بر روی چند دیسک و نوشتن اطلاعات کنترلی خطا (ECC) روی دیسکهای دیگر در قالب بیت‌های داده ذخیره می‌کند. در هنگام خواندن، داده با اطلاعات کنترلی تطابق داده می‌شود و اگر خطایی وجود داشته باشد، تصحیح می‌شود. سرعت خواندن با توجه به Striping داده در چند دیسک افزایش یافته و با کنترل خطا کمی از آن کاسته می‌شود. سرعت نوشتن هم تقریباً در حد خواندن است با این تفاوت که محاسبه امکان تصحیح خطا کمی تاخیر ایجاد می‌کند. علاوه بر تصحیح خطا این آرایه می‌تواند در صورت خارج شدن یکی از دیسک‌ها به کار خود ادامه دهد.



• RAID 3 (level 2 with parity)

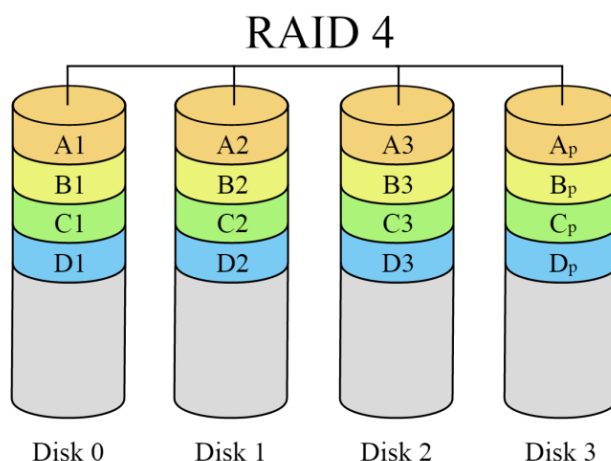
در این نوع RAID ساختار Striping در سطح بایت انجام می‌شود. در این حالت تعدادی دیسک داریم که یک دیسک به parity اختصاص داده می‌شود مثلاً اگر سه دیسک داشته باشیم در دیسک اول یک بایت داده و در دیسک دوم هم یک بایت داده و دیسک سوم parity دیتای دو دیسک دیگر را در خودش نگه می‌دارد و اگر مثلاً

دیسک دوم از بین برود از طریق دیسک parity میتوان اطلاعات دیسک دوم را برگرداند (با xor کردن دیسک اول و parity اطلاعات دیسک دوم برمیگردد).



• RAID 4 (level 0 with parity block)

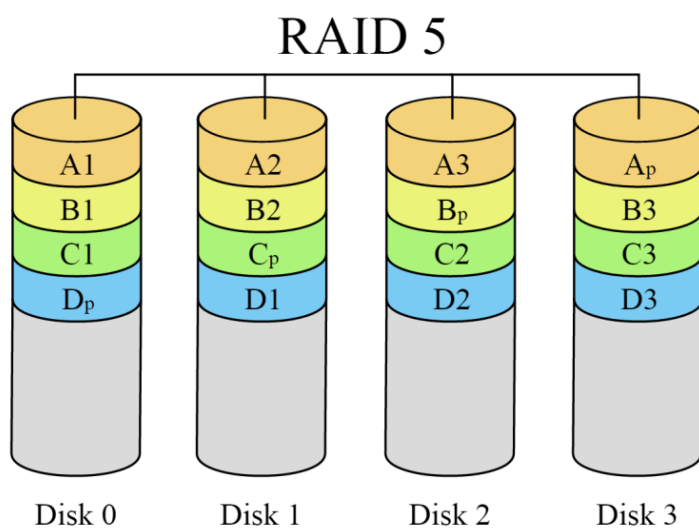
همانند RAID 3 عمل می کند با این تفاوت که به جای خواندن و نوشتن در قالب بایت، از بلاک داده استفاده می کند. مانند RAID 3 دیسکی مختص Parity برای تشخیص خطا دارد. فضای حاصله معادل جمع دیسک ها منهای یک دیسک فضای Parity است. این آرایه ظرفیت خرابی یکی از دیسک ها را داراست.



• RAID 5 (level 4 with distributed parity blocks)

این نوع RAID همانند RAID 1 یکی از پرکاربردترین نوعها می باشد و مشابه آرایه نوع ۴ است. با این تفاوت که Parity آن در دیسکی خاص ذخیره نمی شود و بین تمامی دیسک ها پخش می شود. این نوع آرایه پر کاربردترین نوع در استفاده های حرفه ای است چرا که از نظر کارایی و فضای حاصله، تعادلی بهینه در آن برقرار است. پخش شدن اطلاعات Parity در میان دیسک ها باعث افزایش کارایی می شود. این آرایه به حداقل ۳

دیسک نیاز دارد و دیسک‌های بیشتر برای استریپ کردن و بالا بردن کارایی کاربرد دارد. فضای حاصله معادل جمع دیسک‌ها منهای یک دیسک فضای Parity است. این آرایه نیز مانند RAID 4 ظرفیت خرابی یکی از دیسک‌ها را داراست.





University of Kurdistan

Faculty of Engineering

Department of Computer Engineering

The Booklet of:

Operating Systems

Priveded by:

Dr. Alireza Abdollahpouri