سیستم عامل

Operating System

Operating Systems are an essential part of any computer. Similarly, a course on operating system is an essential part of any computer-science education...

سیستم عامل، یک بخش حیاتی از هر کامپیوتر است. بنابراین درس سیستم عامل نیز یک بخش حیاتی از هر تحصیلات مرتبط با کامپیوتر است...

حمید رضا نیرومند

http://niroomand.ir

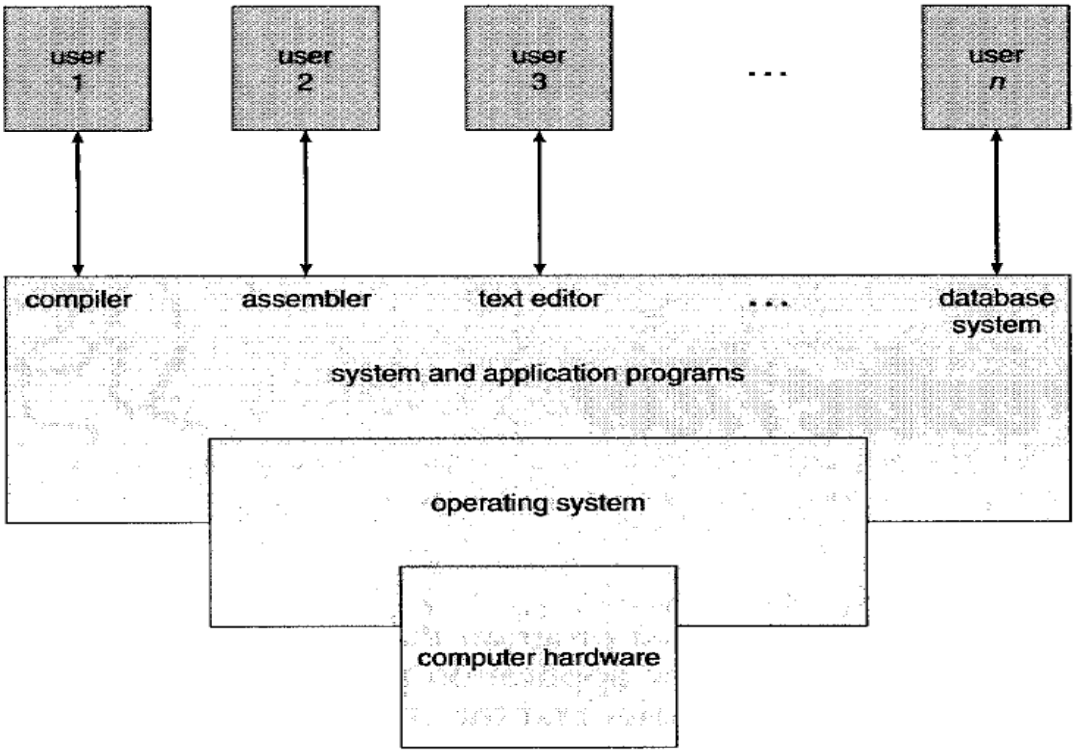
منابع:

- **Operating System Concepts**
  Abraham Silberschatz, Peter Baer Galvin, Greg Gagne
- **Operating Systems**
  William Stalling
- **Wikipedia.org**

توجه:

- استفاده از جزوه بدون حضور در کلاس‌ها پیشنهاد نمی‌شود.
- این یک انتشار غیررسمی از جزوه و ویژه دانشجویان مهندس نیرومند است.
- این نسخه ۱.۰,۲ از جزوه است و به مرور غنی‌تر خواهد شد. (لطفاً اشتباهات سهوی که مشاهده می‌کنید را به ایمیل info@niroomand.ir ارسال نمایید.)

نقش سیستم عامل در یک سیستم کامپیوتری



چهار جز اصلی یک سیستم کامپیوتری:

1- **Hardware**: CPU, Memory, Input/output Devices
The hardware provides the basic computing resources for the system.

سخت‌افزار، منابع پردازشیِ اساسی را برای یک سیستم فراهم می‌کند.

2- **Operating System**:
The operating system controls and coordinates the use of the hardware among the various application programs for the various users.

سیستم عامل، استفاده از سخت‌افزار را بین برنامه‌های کاربردی مختلف و کاربران مختلف کنترل و هماهنگ می‌کند.

3- **Application Programs:**
The application programs - such as word processors, spreadsheets, compilers and web browsers - define the ways in which these resources are used to solve users computing problems.

برنامه‌های کاربردی مثل Word، Excel (صفحات گسترده)، کامپایلر و مرورگرهای وب، روش‌هایی برای حل مشکلات پردازشی کاربران از طریق به کار گیری منابع سخت‌افزاری هستند.

4- **Users:**
The users simply use application programs to do their jobs.

کاربران، از برنامه‌های کاربردی برای انجام کارهای خود استفاده می‌کنند.

تعاریف مختلف سیستم عامل:

یک تعریف کلی که برای همه قابل قبول باشد از سیستم عامل وجود ندارد. اما اگر بخواهیم سیستم عامل را نسبت به وظیفه‌ای که انجام می‌دهد، تعریف کنیم، عمومی‌ترین تعریف این است که:

- OS is the one program running at all times on the computer (usually called the kernel) with all else being system programs and application programs.

سیستم عامل مجموعه‌ای است از: ۱- آن برنامه‌ای که در تمام لحظات روی کامپیوتر در حال اجراست (که معمولاً «کرنل» نامیده می‌شود) و ۲- تمام برنامه‌های دیگری که برخی، برنامه‌های سیستمی و برخی، برنامه‌های کاربردی هستند.

تعاریف دیگری از سیستم عامل:

- An OS acts as an intermediary between the user of a computer and the computer hardware.

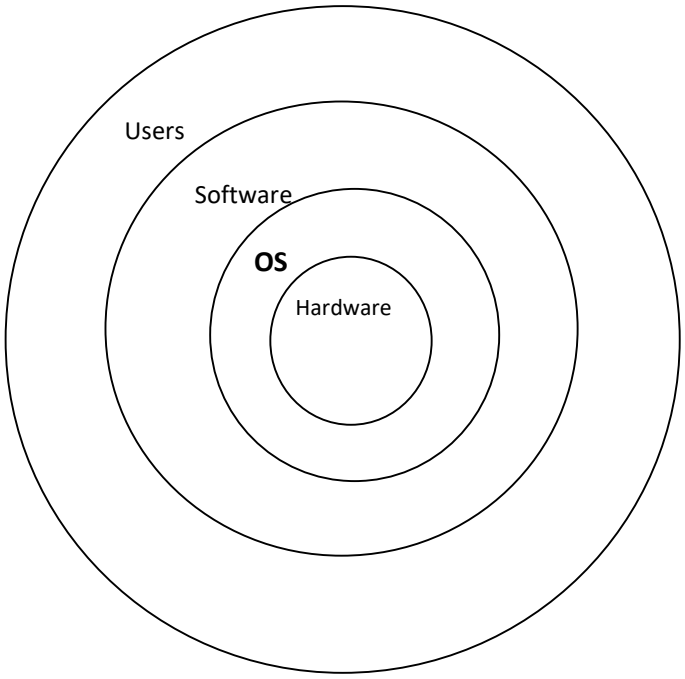
سیستم عامل به عنوان واسطی بین کاربران یک کامپیوتر و سخت‌افزار عمل می‌کند.

- The purpose of an OS is to provide an environment in which a user can execute programs in a convenient and efficient manner.

هدف یک سیستم عامل این است که محیطی را فراهم کند که کاربر بتواند برنامه‌هایش را به روشی موثر و راحت اجرا کند.

- An OS is software that manages the computer hardware.

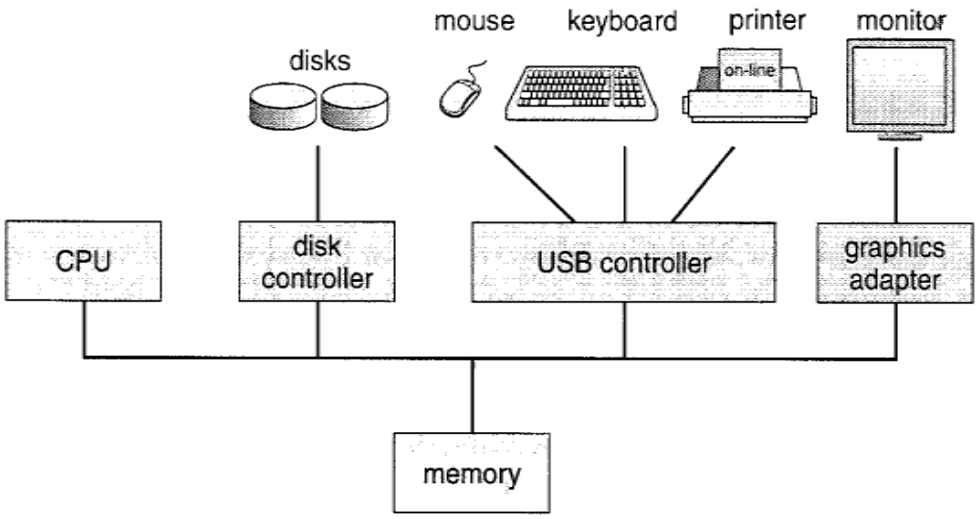
سیستم عامل یک برنامه برای مدیریت سخت‌افزار کامپیوتر است.



- یک سیستم عامل شبیه یک government (دولت) است. به خودی خود کاری انجام نمی‌دهد، اما environment (محیطی) فراهم می‌کند که برنامه‌های دیگر بتواند کارهای مفیدی انجام دهند.
- دو وظیفه اصلی هر سیستم عامل:
  - سیستم عامل یک Resource Allocator است: یعنی منابعی همچون CPU time، Memory space، File-storage space، I/O devices را طبق الگوریتم‌هایی به متقاضیان اختصاص می‌دهد.
  - سیستم عامل یک Control Program است: یعنی اجرای برنامه‌ها را کنترل می‌کند تا از خطاها و استفاده نادرست از سیستم جلوگیری می‌کند.

ساختار یک سیستم کامپیوتری:

۱. روند کلی عملکرد یک سیستم کامپیوتری:



- در یک سیستم کامپیوتری یک یا چند CPU یا چندین Device Controller توسط یک باس مشترک که دسترسی به Memory را فراهم می‌کنند به هم متصل هستند.

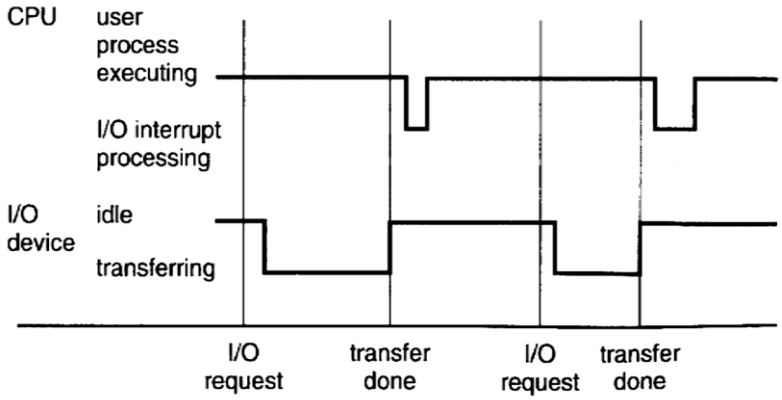
- در هنگام آغاز به کارِ کامپیوتر، به یک برنامه اولیه برای راه اندازی نیاز داریم. به این برنامه در اصطلاح bootstrap (خودراه‌انداز) گفته می‌شود. که روی ROM یا روی EEPROM ذخیره شده است.

- Bootstrap سیستم عامل را روی Memory بارگذاری می‌کند. سپس سیستم عامل شروع به اجرای اولین وقفه کرده و منتظر اتفاق افتادن یک رخداد (event) می‌ماند.

- اتفاقِ event معمولاً توسط (interrupt از طرف سخت‌افزارها یا نرم‌افزارها) به CPU سیگنال می‌شود.

- «سخت‌افزار» برای فرستادن وقفه معمولاً از طریق System Bus یک سیگنال به CPU می‌فرستد و «نرم‌افزار» برای این کار یک عملیات خاص را اجرا می‌کند که System Call یا Monitor Call گفته می‌شود.

- وقتی CPU وقفه را دریافت می‌کند. کاری را که در حال انجام آن است متوقف کرده، سریعاً محل اجرا را به محل مربوط به وقفه جدید منتقل می‌کند.

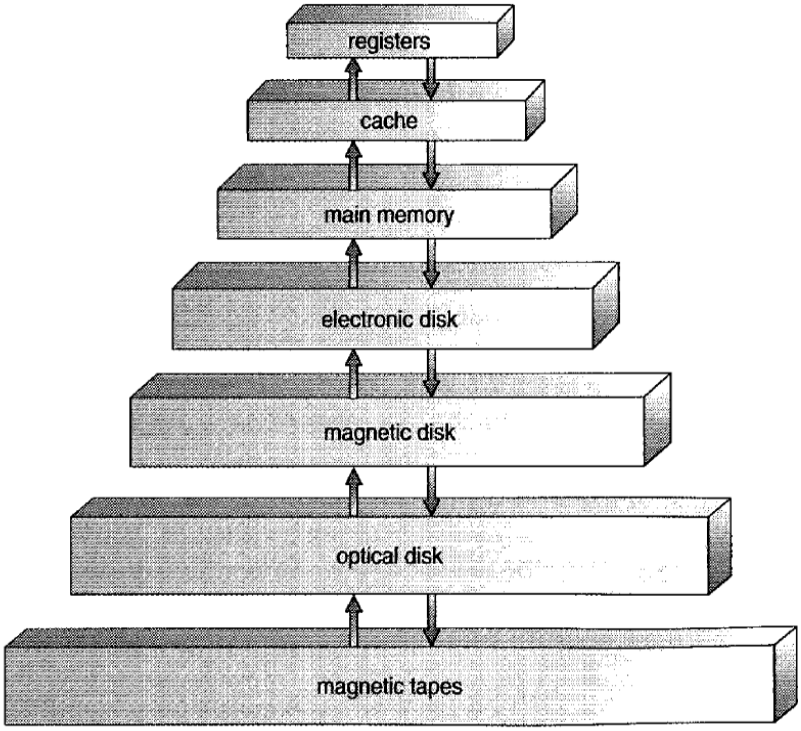


۲. ساختار ذخیره‌سازی:

Computer programs must be in main memory to be executed. Main memory is the only large stowage area that the processor can access directly.

برنامه‌های کامپیوتری باید روی حافظه اصلی باشند تا اجرا شوند. حافظه اصلی تنها حافظه بزرگی است که CPU می‌تواند مستقیماً به آن دسترسی داشته باشد.

سلسله مراتب حافظه:



وظیفه سیستم عامل در رابطه با حافظه‌ها:

- File-System management: /مدیریت سیستم فایل/
- Mass-Storage management: /مدیریت حافظه‌های بزرگ/
- Caching: /انتقال داده‌ها از روی RAM به روی cache/

مطالعه آزاد:

آشنایی با مفهوم bit و حافظه‌ها:

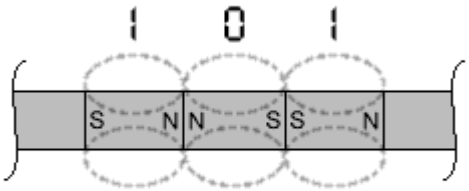
همانطور که می‌دانید، در کامپیوتر برای نمایش داده‌ها از دو مفهوم استفاده می‌کنیم: بودن یا نبودن! منظور همان صفر و یک است. توجه کنید که صفر و یک، **نماد** هستند. صفر، نماد یک وضعیت و یک، نماد وضعیتی دیگر. همه اطلاعات کامپیوتر در نهایت تبدیل به همین 0ها و 1ها می‌شوند. چندین 0 و 1

کنار هم قرار می‌گیرند و داده‌های مختلف را پدید می‌آورند. به طور مثال اگر ۷ تا صفر و یک به صورت 1000001 کنار هم قرار بگیرند، طبق جدولی به نام جدول اسکی) که تصویر آن‌را می‌توانید **اینجا** مشاهده کنید) کامپیوتر آن‌ها را کاراکتر a به حساب خواهد آورد.

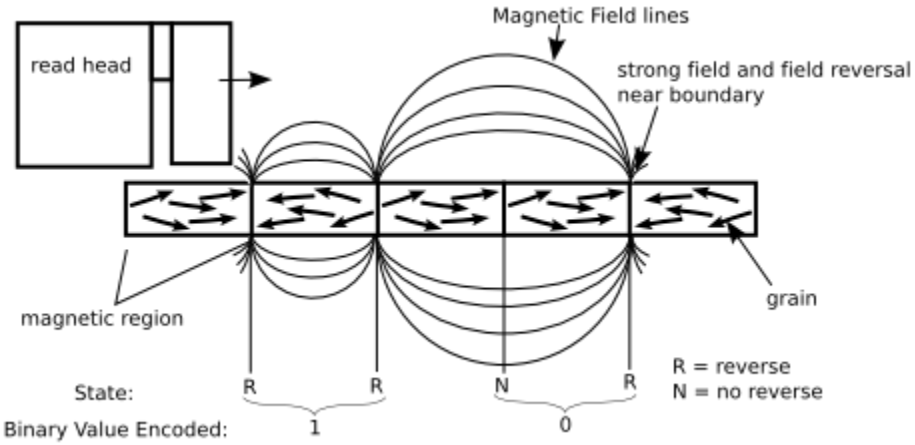
اما از نظر سخت افزاری بحث چگونه است؟ یعنی اگر از شما بپرسند بر روی دیسک نرمی که داخل یک فلاپی است (فلاپی یعنی نرم)، چه چیزی قرار دارد که می‌توان داده‌ها را یعنی همان صفر و یک‌ها را به وسیله آن‌ها نمایش داد، چه خواهید گفت؟ Hard Disc چطور؟ در مورد CD و DVD چطور؟

من سعی می‌کنم با چند عکس و ویدئو مفهوم را کمی شفاف کنم.

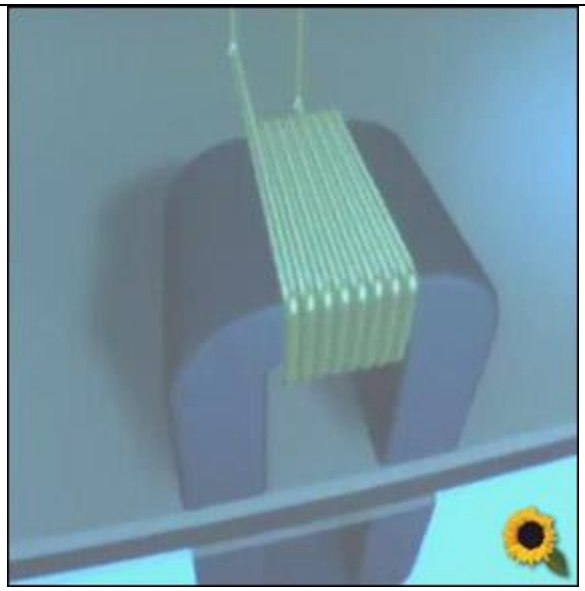
باید بدانید که فلاپی و سی‌دی برای نگه داشتن اطلاعات بر روی خود به دو روش مختلف عمل می‌کنند. فلاپی یک دیسک مغناطیسی (Magnetic Disc) است (هارد دیسک هم همینطور) اما سی.دی و دی.وی.دی دیسک‌های نوری (Optical Disc) هستند.
تفاوت این دو در این است که بر روی فلاپی و هارد دیسک ما قطعات بسیار ریز مغناطیسی (Magnetic Particles) داریم که توانایی حفظ خاصیت مغناطیسی خود را برای مدت طولانی دارند.
هر قطعه قطب شمال (N) و جنوب (S) مربوط به خود را دارد.



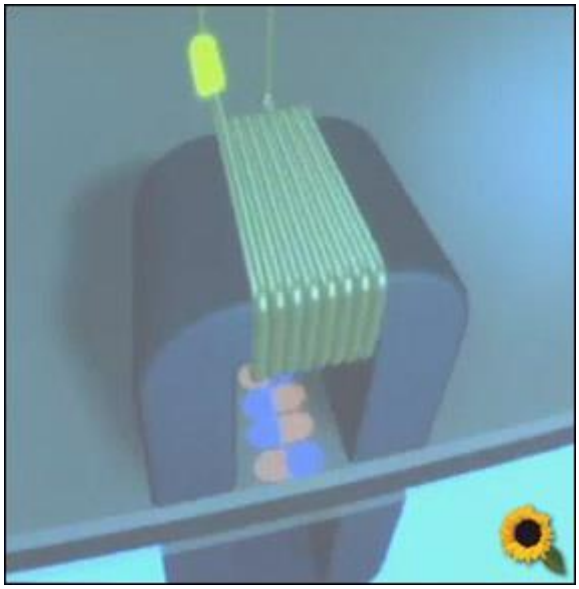
عکسی دیگر از دیسک و هد:



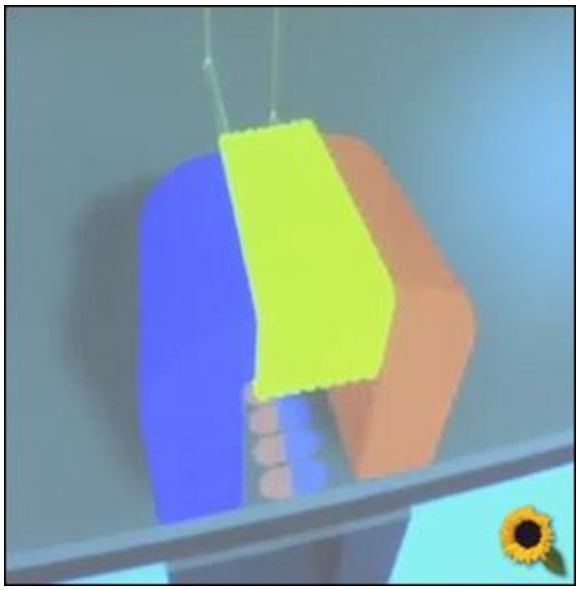
حالا باید تصور کنید که چطور می‌توان با این قطعات ریز، 0 و 1 را نمایش داد؟
خیلی ساده است، به طور مثال اگر قطب S در سمت راست بود، این قطعه را می‌گیریم نماد 0 و اگر قطب S در سمت چپ بود، می‌گیریم نماد 1.
سؤال این است که چطور قطب این قطعات را جا به جا کنیم که نماد صفر یا یک شوند؟ یعنی در حقیقت **چطور بر روی فلاپی و هارد دیسک بنویسیم؟**
در فلاپی درایو (Floppy Drive) و هارد دیسک درایو (H.D. Drive) یک هد (head) برای خواندن و نوشتن داریم.



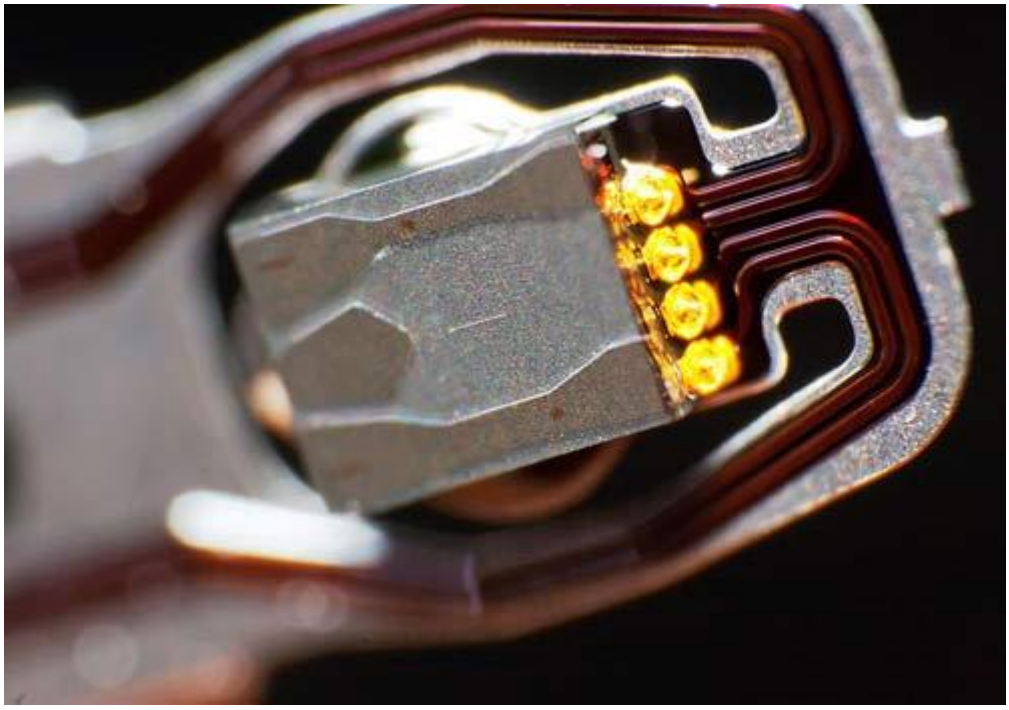
این هد با سطح دیسک در تماس است. این هد در حقیقت یک سیم‌لوله است. وقتی قرار است یک قطعه نماد صفر شود، به طور مثال پالس مثبت را به سیم سمت چپ هد می‌فرستیم:



میدانی که در سیم‌پیچ ایجاد می‌شود، آن را تبدیل به یک آهن‌ربا می‌کند و باعث می‌شود سمت راست میله قطب N شود و در نتیجه قطب N مربوط به هر قطعه‌ای که بین این آهن‌ربا قرار گیرد دفع شده و از آن دور می‌شود (قطب‌های مشابه همدیگر را دفع می‌کنند). یعنی در حقیقت چنین اتفاقی در مورد قطعات روی دیسک می‌افتد:



قطعات با توجه به پالسی که به هد دادیم، سر و ته می‌شوند. (قرمز از قرمز دور می‌شود و آبی از آبی)
این قطعات می‌توانند تا مدت‌ها خاصیت مغناطیسی خود را حفظ کنند.
پس تکلیف نوشتن بر روی فلاپی دیسک و هارد دیسک مشخص شد.
بد نیست نگاهی به یک عکس میکروسکوپی از هد هارد دیسک بیندازیم:



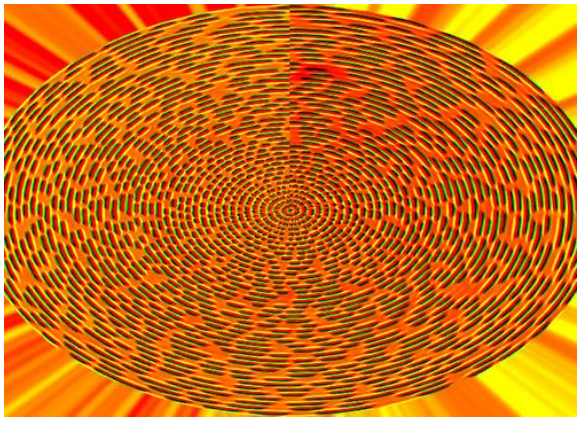
اطلاعات بیشتر در مورد هارد دیسک در مقاله :**داخل هارد دیسک چه می‌گذرد؟(Inside Hard Disk)**

اما چطور اطلاعات را از روی این دیسک بخوانیم؟

این کار هم با همین هدها انجام می‌شود. هنگام خواندن، عمل عکس نوشتن رخ می‌دهد. قطعات، خاصیت مغناطیسی خود را حفظ کرده‌اند. همین که به میان سیملوله می‌رسند، سیملوله یک بار بسیار ضعیف از آن‌ها دریافت می‌کند و نسبت به اینکه کدام جهت آن مثبت شد، میدانی در سیملوله ایجاد می‌شود و جریان به سمت یکی از سیم‌های بالا جاری می‌شود. اگر سیم سمت چپ جریان را رساند یعنی صفر و اگر سیم سمت راست جریان را رساند یعنی یک.
منبع عکس‌ها و صحبت‌هایم در مورد فلاپی، ویدئوهای مبانی کامپیوتر دانشگاه هاروارد است.

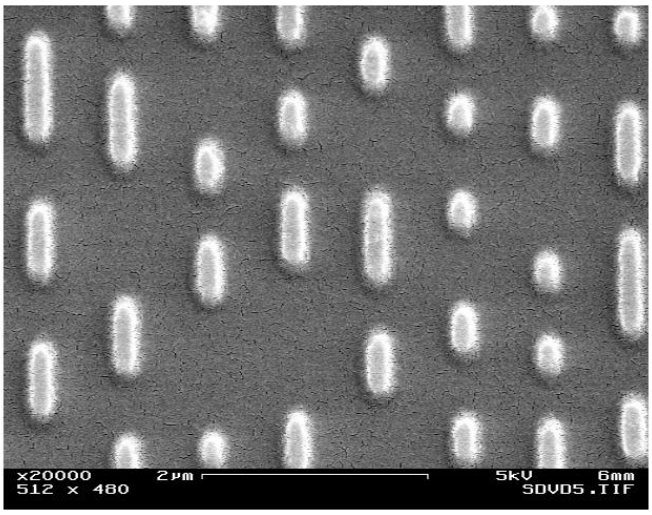
و اما عملکرد CD و DVD به چه صورت است؟

در دیسک‌های نوری، همانطور که از اسمش مشخص است، از طریق بازتاب نور، صفر و یک تشخیص داده می‌شود. روی سطح این دیسک‌ها پر است از قطعاتی که به طور پیشفرض نوری که از طریق هد به آن‌ها تابیده می‌شود را بازتاب می‌کنند .وقتی بر روی یک CD ، رایت می‌کنید، در حقیقت نقاطی که قرار است 0 شوند را با لیزر می‌سوزانید تا بازتاب نکنند! به همین دلیل است که بهتر است به جای رایت کردن سی.دی بگوییم Burn کردن یعنی سوزاندن.
برای خواندن از روی دیسک نوری، یک نور شروع به تابیدن به سطح دیسک می‌کند و یک دیود بازتاب را دریافت می‌کند، هر کجا که بازتاب نور قطع شد یعنی صفر و هر کجا که بازتاب داشت یعنی یک.
به چند تصویر که با میکروسکوپ از سطح یک دیسک نوری گرفته شده است دقت کنید:



تصویر بالا و پایین یک CD یا همان Compact Disc را نمایش می‌دهد.

تصویر زیر سطح یک DVD را از نمایی نزدیک‌تر نمایش می‌دهد:



برای درک بهتر این موضوع یک ویدئوی جالب معرفی می‌کنم. (از اینجا ببینید) اگر لینک مستقیم دانلود با کیفیت‌های مختلف می‌خواهید، اینجا را ببینید.
فکر می‌کنم لازم به توضیح نیست که چه اتفاقی می‌افتد که یک DVD حجم بیشتری داده نسبت به یک CD گنجایش دارد در حالی که از لحاظ فیزیکی یک اندازه هستند!؟

طبیعتاً هر چه تکنولوژی بیشتر پیشرفت می‌کند، می‌توان قطعاتی ریزتر تولید کرد. یعنی بر روی یک DVD قطعات ریز بیشتری نصب شده است، همین!
و یا در دیسک‌های Blue Ray تعداد این قطعات چندین برابر شده است و در نتیجه ۰ و ۱ های بیشتری را می‌توان نمایش داد.
در این ویدئو با یک مثال، به خوبی فرق سی.دی، دی.وی.دی و بلوری نمایش داده شده است:

آن قطعات سفید را نماد همان قطعات ریز روی دیسک‌ها بگیرید. به دو بیتی که سوزانده شده‌اند دقت کنید!
اینکه برخیDVD ها حجم بیشتری نسبت به دیگر دی.وی.دی‌ها دارند به خاطر این است که گاهی دو لایه بر روی هم قرار می‌گیرند یا اینکه یک لایه است اما در دو طرف آن می‌توان نوشت.

۳. ساختار I/O:

بخش عمده‌ای از کد سیستم عامل برای مدیریت I/O است. هم به خاطر اهمیت آن و هم به خاطر تنوع دستگاه‌های ورودی و خروجی، نوعاً سیستم عامل‌ها برای هر Device Controller یک Device Driver دارند.

برای آغاز یک عملیات I/O، «راه‌انداز قطعه» (Device Driver) رجیسترهای لازم را در Device controller (کنترل کننده‌ی قطعه) لود می‌کند. Device controller به نوبت محتوای این رجیسترها را چک می‌کند تا نهایتاً مشخص شود چه عملی باید انجام شود. (مثلاً ممکن است این عمل، خواندن یک کاراکتر از کیبورد باشد)

تحقیق: DMA چیست؟ نقش DMA در جلوگیری از سربار اضافی (Overhead) چیست؟

تاریخچه سیستم عامل:

- جهت بررسی تاریخچه سیستم عامل باید تاریخچه معماری ماشین‌هایی را در نظر بگیریم که سیستم عامل‌ها بر روی آن اجرا می‌شده‌اند:

1- **Early systems (1945-1955)**
   - Hardware expensive, Humans cheap
   - ENAIC [30 tons, 200KW power, filled an entire room]
   - No operating system
   - No protection
   - Human operators

2- **Second Generation systems (1956-1965)**
   - Transistors and batch systems
   - Interrupts
   - Minimal protection

3- Third Generation system (1965-1980)
   - ICs and Multi-programming
   - Time sharing
   - Virtual Memory
   - Spooling

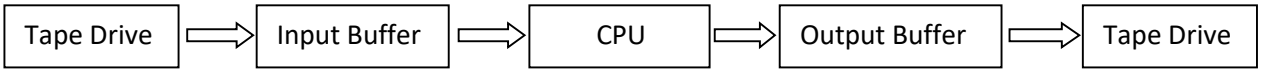
4- **Fourth Generation and beyond (1981-…)**
   - Personal Computers
   - MS-DOS
   - Computer Networks -> Network OSs
   - Distributed computing -> Distributed OSs

- **انواع سیستم‌ها از نظر ارتباط با کاربر:**
  - **سیستم‌های Interactive یا محاوره‌ای:**
    - کاربر به طور مستقیم و Online با کامپیوتر در ارتباط است. کاربر دستوراتی را وارد می‌کند و منتظر پاسخ می‌ماند. پس از دریافت پاسخ، مجدداً دستوراتی را وارد می‌کند...
  - **سیستم‌های Batch یا دسته‌ای:**
    - دریافت دستورات در یک زمان و اجرای آن‌ها در یک زمان دیگر انجام می‌گیرد. برنامه‌های با نیاز مشابه (مثلاً برنامه‌هایی که قرار است کامپایل شوند) در یک دسته به سیستم وارد می‌شوند و پس از بارگذاریِ کامپایلرِ مورد نیازشان، اجرای آن‌ها به طور متوالی انجام می‌شود.

- **انواع سیستم از نظر ارتباط با وسایل جانبی:**
  - **سیستم‌های آنلاین یا ارتباط مستقیم:**
    - پردازنده، مستقیماً با دستگاه‌های ورودی-خروجی در ارتباط است و به دلیل کند بودن این دستگاه‌ها، کارایی پردازنده به مقدار قابل توجهی کاهش می‌یابد.
  - **سیستم آفلاین یا ارتباط غیرمستقیم:**
    - پردازنده به طور مستقیم با دستگاه‌های ورودی-خروجی در ارتباط نیست.

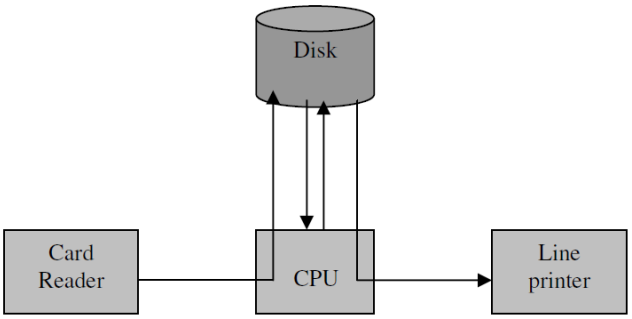
مفهوم Buffering:

اگر CPU به طور مستقیم با دستگاه‌های ورودی و خروجی در ارتباط باشد با توجه به سرعت کم دستگاه‌های ورودی و خروجی و سرعت بالای CPU، استفاده بهینه‌ای از CPU نمی‌شود و در مواقعی ممکن است CPU، idle (بیکار) باشد. در این حالت برای هماهنگی بین وسایل I/O و پردازنده، از قسمتی از حافظه (RAM) استفاده می‌شود که به آن Buffer و به این عملیات Buffering گفته می‌شود، یعنی داده‌ها قبل از ورود به CPU، ابتدا وارد این بخش از حافظه می‌شود (بافِر می‌شوند) و CPU به جای دسترسی مستقیم به دستگاه‌های ورودی و خروجی با این بخش از حافظه که سرعت بالاتری دارد در ارتباط خواهد بود.



مفهوم Spooling:

اسپولینگ نیز همان معنای بافرینگ است با این تفاوت که در بافرینگ امکان همزمانی پردازش یک کار به کمک حافظه اصلی فراهم می‌شود در حالی که در اسپولینگ امکان همزمانی پردازش ورودی و خروجی چند کار توسط حافظه جانبی صورت می‌گیرد.



مفهوم چند برنامگی (Multi-programming):

با استفاده از بافر و اسپولینگ می‌توان چند کار را به طور همزمان انجام داد. در چند برنامگی اجرای یک کار تا زمانی که به I/O نیاز داشته باشد ادامه می‌یابد. سپس پردازنده، آن کار را مشغول I/O می‌کند و اجرای کار دیگری را شروع کرده یا ادامه می‌دهد. توجه دارید که اگر کار I/O مربوط به برنامه قبلی تمام شد، CPU توسط یک وقفه مطلع می‌شود و با الگوریتم‌های Job-Scheduling (زمان‌بندی کار) اگر نیاز بود، آن کار را در نوبت پردازش قرار می‌دهد.

مفهوم سیستم‌های اشتراک زمانی Time-sharing:

شکل خاصی از چند برنامگی است که در آن تعویض یک کار، بر اساس یک مقطع زمانی و نه بر اساس مدتِ نیاز آن کار به ورودی یا خروجی صورت می‌گیرد. اجرای یک کار تا پایان بازه زمانی (مثلاً ۲ میلی ثانیه) برای هر کار ادامه می‌یابد. سپس پردازنده، اجرای برنامه دیگری را شروع می‌کند. عمل switch کردن بین پردازنده‌ها چنان سریع است که کاربر فکر می‌کند سیستم به تنهایی فقط در اختیار اوست!

مفهوم سیستم‌های توزیع شده:

A distributed system is a collection of physically separate, possibly heterogeneous computer systems that are networked to provide the users with access to the various resources that the system maintains.

Access to shared resources increases computation speed, functionality, data availability and reliability.

یک سیستم توزیع شده مجموعه‌ای از سیستم‌هاست که از نظر فیزیکی مجزا و حتی احتمالاً غیر همجنس هستند و این سیستم‌ها با هم شبکه شده‌اند تا به کاربران دسترسی به منابع مختلفی که در سیستم موجود است را بدهند.

دسترسی به منابع مشترک، سرعت پردازش، کارایی، در دسترس بودن و قابل اعتماد بودن داده‌ها را افزایش می‌دهد.

مدیریت پردازه‌ها

Process Management

تعریف Process یا فرآیند: به برنامه‌ی در حال اجرا، پردازه یا فرآیند گفته می‌شود.

A process is a program in execution.

یک پردازه بیش از یک قطعه کد است. اجزای دیگری نیز دارد، مانند:

۱- بخش متنی پردازه که شامل کدهای آن است.
۲- Program Counter که شامل شماره برنامه است.
۳- Stack که شامل داده‌های موقت؛ مانند: پارامترهای توابع، آدرس‌های return و متغیرهای محلی و... است.
۴- Data section که شامل متغیرهای Global است.
۵- Heap حافظه‌ای که در حین اجرای پردازه به آن اختصاص داده می‌شود.

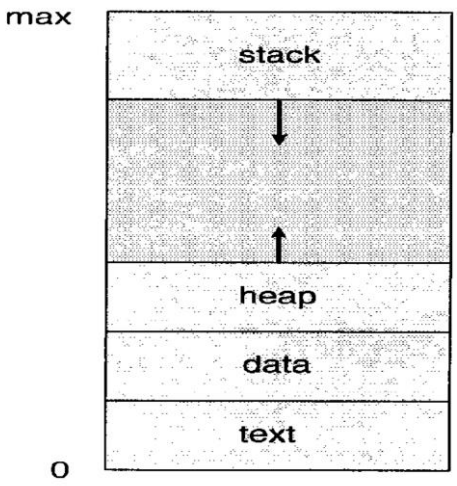


Figure 3.1 Process in memory.

نکته: Process ممکن است job هم گفته شود چون شغل افراد در سیستم‌های Batch این بود که اطلاعات را وارد سیستم کنند.

وضعیت پردازه (Process State)

پردازه‌ها در حین اجرا وضعیت خود را تغیر می‌دهند. هر پردازه ممکن است در یکی از وضعیت‌های زیر باشد:

1- **New**: The process in being created.

در این وضعیت پردازه ایجاد می‌شود.

2- **Running**: instructions are being executed.

در این وضعیت دستور العمل‌ها اجرا می‌شوند.

3- **Waiting**: The process is waiting for some event to occur.

در این وضعیت پردازه منتظر اتفاق یک رخداد است. (مثل انتظار برای اتمام عملیات I/O)

4- **Ready**: The process is waiting to be assigned to a processor.

در این وضعیت پردازه منتظر این است که به یک پردازنده (CPU) ارجاع داده شود. نکته: منابع و حافظه مورد نیاز هر پردازه در این مرحله به پردازه تخصیص داده می‌شود.

5- **Terminated**: The process has finished execution.

در این وضعیت اجرای پردازه تمام شده است. بنابراین تمامی منابعی که به پردازه اختصاص داده شده بود آزاد می‌شود.

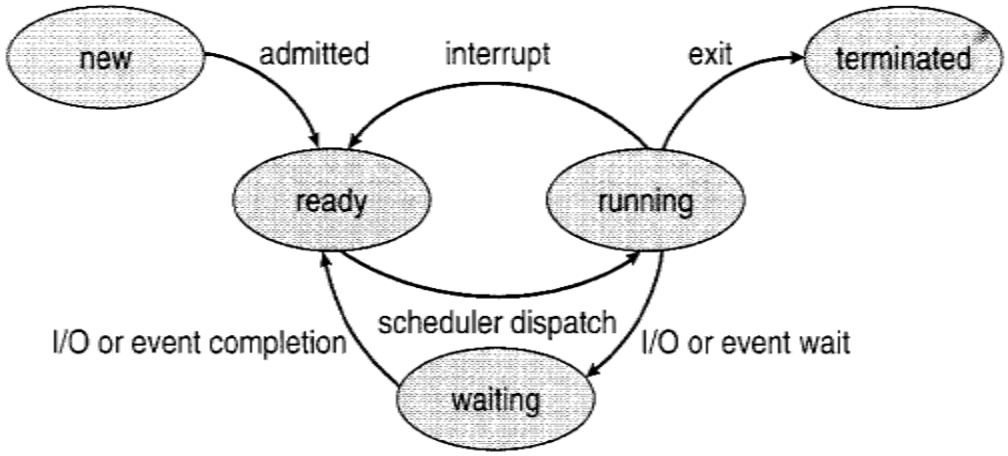


Figure 3.2 Diagram of process state.

توجه۱: نام Stateها در هر سیستم عامل ممکن است فرق کند اما در کل، وضعیت‌ها در همه سیستم عامل‌ها یکی هستند.

توجه۲: به جهت فلش‌ها در شکل بالا بسیار دقت کنید. اینکه یک پردازه از کدام حالت می‌تواند به کدام حالت برود، بارها سؤال کنکور بوده است.

دو اصطلاح مهم:

۱- پردازه I/O-limited: یعنی بیشتر زمان اجرای پردازه از نوع I/O است.

۲- CPU-Limited: بیشتر زمان اجرای فرآیند در CPU می‌گذرد.

بلاک کنترل پردازه یا PCB (Process Control Block):

هر پردازه علاوه بر سورس‌کد، Data، Stack و... همراه با یک Block کنترل پردازه یا PCB در سیستم عامل نمایش داده می‌شود. واضح است که این Block شامل اطلاعاتی جهت کنترل پردازه توسط سیستم عامل است.

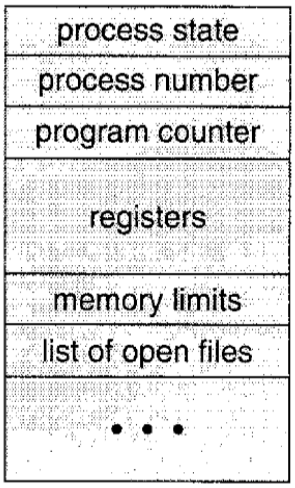


Figure 3.3 Process control block (PCB).

اجزای PCB:

۱- وضعیت پردازه

۲- شماره پردازه

۳- Pc یا program Counter: شامل آدرس دستورالعمل بعدی این پردازه است که قرار است اجرا شود.

۴- CPU Registers: رجیسترهایی مثل ACC, stack pointer, index Register و...

۵- Memory limits: نهایت میزان memory مورد نیاز پردازه

۶- List of open files: لیست فایل‌های باز در این برنامه

۷- اطلاعات مربوط به صف‌بندی پردازه‌ها و جایگاه پردازه در صف

۸- اطلاعات مربوط به مدیریت حافظه

۹- اطلاعات حسابرسی مربوط به مقدار CPU و زمان قابل استفاده برای پردازه و...

۱۰- اطلاعات مربوط به I/Oها: شامل لیست دستگاه‌های I/O که به پردازه اختصاص داده شده، لیست فایل‌های باز مورد استفاده برای این پردازه و...

PCB چه کاربردی در مدیریت پردازه‌ها دارد؟

به محض وقوع یک interrupt یا system call تمامی اطلاعات مربوط به پردازه اول در PCB ذخیره می‌شود (مثل «حالت پردازه») و پردازه بعدی اجرا می‌شود و این روند با چک کردن PCB توسط سیستم عامل به خوبی مدیریت می‌شود.

به تصویر زیر دقت کنید:

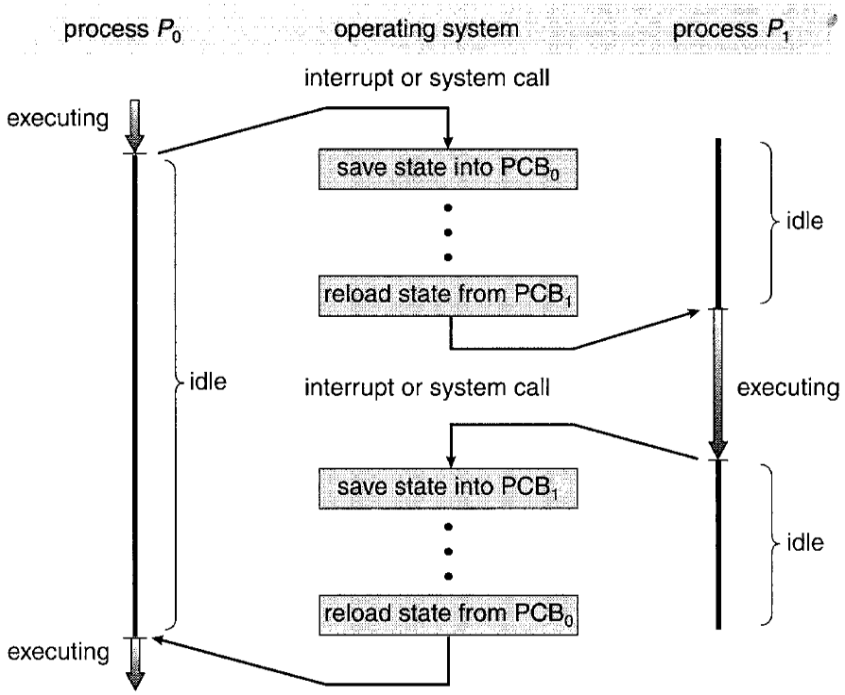


Figure 3.4 Diagram showing CPU switch from process to process.

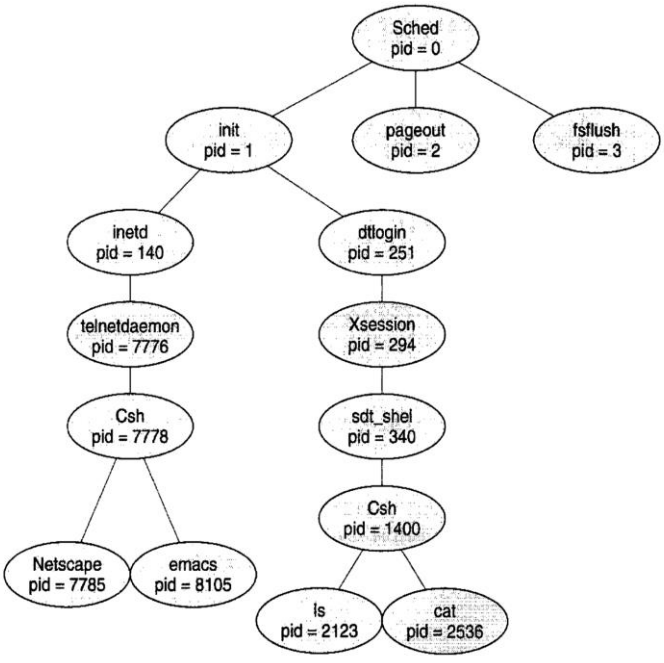
باید بتوانید به خوبی این تصویر را توضیح دهید...

عملیات بر روی پردازه‌ها:

۱- **ایجاد پردازه**: هر پردازه ممکن است در حین اجرا چندین پردازه دیگر ایجاد کند. به پردازه اول، parent یا والد و به پردازه‌های جدید children یا بچه‌های پردازه می‌گویند.

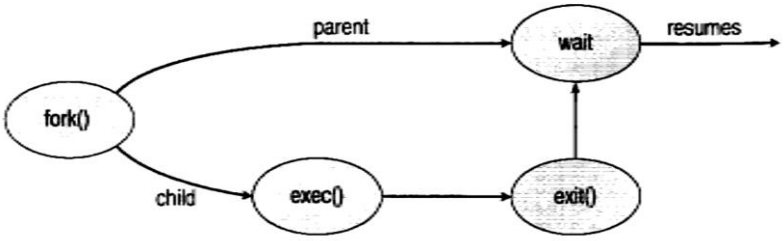
هر پردازه جدید ممکن است پردازه‌های جدیدتری ایجاد کند که در کل تشکیل درختی از پردازه‌ها (tree) را می‌دهند.

هر سیستم عامل به پردازه‌های جدید یک Process Identifier یا PID اختصاص می‌دهد.



وقتی که یک پردازه، پردازه دیگری ایجاد می‌کند ممکن است دو حالت رخ دهد:

۱. هر پردازه والد تا پایان یافتن پردازه‌های فرزند منتظر می‌ماند و سپس پردازش می‌شود یا خاتمه می‌یابد.



۲. پردازه والد همزمان با فرزندان اجرا می‌شود.

۲- ختم پردازه:

دلایل ختم یک پردازه:

۱- پردازه، آخرین دستور را اجرا می‌کند و از سیستم عامل تقاضای حذف خود را می‌کند. (مثلاً تقاضا از طریق اجرای دستور ()exit یا از طریق یک system call)

۲- فرآیند والد، فرآیند فرزند را خاتمه می‌دهد.

دلایل ختم پردازه فرزند توسط والد:

۱. ممکن است فرآیند والد دچار مشکل شود و Exit شود. در این صورت اکثر سیستم عامل‌ها به فرآیند فرزند اجازه ادامه فعالیت نمی‌دهند. به این نوع خاتمه در اصطلاح «خاتمه آبشاری» یا «Cascading Termination» گفته می‌شود.

۲. فرآیند فرزند بیش از حد مجاز، منبع نیاز داشته باشد.

۳. وظیفه محول شده به فرآیند فرزند بیش از این، نیازی نیست.

نــخ

Thread

- سیستم عامل‌های مدرن اجازه Multi-Threading یا «چند نخی» یا «انجام چند رشته از عملیات به طور همزمان» را می‌دهند در حالی که در سیستم عامل‌های سنتی در هر لحظه فقط یک فرآیند سنگین (Heavy Weight Process) داشتیم.
- اگر یک فرآیند به چندین رشته تبدیل شود می‌تواند در هر لحظه بیش از یک وظیفه انجام دهد.

تعریف Thread:

A Thread is a basic unit of CPU utilization

یک نخ یک واحد از کارکرد CPU است.

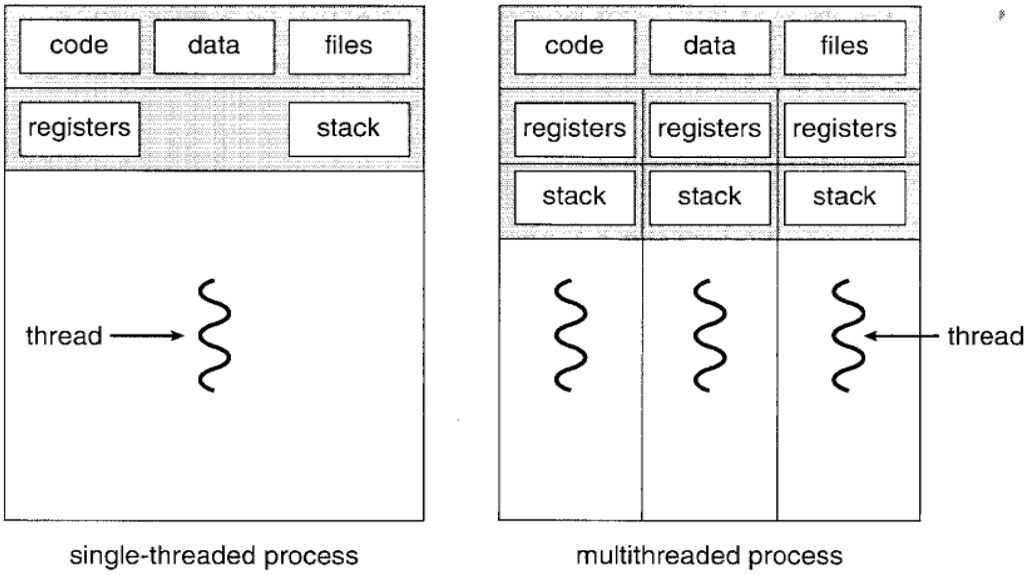


Figure 4.1 Single-threaded and multithreaded processes.

دلایل استفاده از Thread (دلایل و فواید Multi-Threading)

۱. **Responsiveness**: پاسخ دهی سریعتر و در نتیجه اجرای سریعتر برنامه‌ها:

مثال: یک مرورگر وبِ چند نخی می‌تواند در حالی که در یک نخ یک عکس را لود می‌کند، در نخی دیگر به کاربر اجازه پر کردن یک فرم را بدهد.

۲. **Resource Sharing**: اشتراک منابع:

نخ‌های یک پردازه، به طور پیش‌فرض حافظه و منابع مربوط به آن پردازه را بین هم به اشتراک می‌گذارد. در این صورت هر برنامه می‌تواند از یک آدرس از حافظه برای چندین نخ استفاده کند.

۳. **Economy**: مقرون به صرفه بودن (از نظر زمان):

اختصاص حافظه و منابع مورد نیاز یک پردازه و ایجاد یک پردازه جدید بسیار پر هزینه است. با توجه به اینکه نخ‌ها منابع پردازه مربوط به خود را به اشتراک می‌گذارند، ایجاد نخ جدید، اقتصادی‌تر از ایجاد پردازه‌ی جدید است. به طور مثال در یونیکس ایجاد پردازه جدید ۳۰ برابر کندتر از ایجاد نخ جدید است.

۴. **Utilization of multi-processor architectures**: بهبود کارآیی در سیستم‌های با معماری چند پردازشگری:

یک پردازه تک نخی فقط بر روی یک پردازشگر قابل اجراست، فرقی نمی‌کند چند پردازشگر داشته باشیم. در حالی که یک پردازه چند نخی می‌تواند هر نخ را بر روی یک پردازشگر اجرا کند. در نتیجه سرعت اجرا و کارآیی در این نوع معماری بسیار بهبود خواهد یافت.

CPU Scheduling

زمان‌بندی CPU

مقدمه:

در یک سیستم تک پردازه‌ای فقط یک پردازه در هر لحظه می‌تواند اجرا شود، بقیه پردازه‌ها باید منتظر بمانند تا CPU آزاد شود.

هدف از Multi-Programming این است که همیشه پردازه‌هایی در حال اجرا باشند تا به بهترین شکل از CPU استفاده شود.

اگر مفهوم Multi-Programming وجود نداشته باشد، پردازه‌ای که در حال اجراست ممکن است برای کامل شدن یک عمل I/O برای مدتی idle باشد و این یعنی هدر رفتن زمان. در چند برنامگی ما به دنبال به کارگیری این زمان هستیم.

چندین پردازه در RAM نگهداری می‌شوند و به محض اینکه یک پردازه مجبور می‌شود که منتظر بماند (Wait)، سیستم عامل، CPU را مشغول پردازه دیگری می‌کند.

مهم‌ترین هدف سیستم عامل، تخصیص منابع است و مهم‌ترین منبع هر سیستمی، CPU آن سیستم است. بنابراین یکی از مهمترین وظایف سیستم عامل، زمان‌بندی تخصیص CPU است.

مفاهیم پایه:

۱. CPU Scheduler: هرگاه CPU بیکار شود سیستم عامل باید یکی از پردازه‌هایی که در صف Ready وجود دارند را برای اجرا انتخاب کند. انتخاب پردازه توسط CPU Scheduler انجام می‌شود.
۲. Dispatcher: ماژولی است که کنترل CPU را به پردازه‌ای که توسط CPU Scheduler انتخاب شده است واگذار می‌کند.
۳. مفهوم Burst Time: مدت زمانی که یک پردازه نیاز دارد تا پردازشش شود. (معمولا برحسب میلی ثانیه)
۴. Preemptive Scheduling & none-Preemptive Scheduling(=Cooperative): (زمان‌بندی رقابتی و غیر رقابتی (همکاری)):

در حالات زیر می‌توان یک صف برای پردازه‌ها تصور کرد بنابراین تصمیم گیری در زمان‌بندی CPU نیز در یکی از حالات زیر اتفاق می‌افتد:

۱- یک پردازه از حالت Running به حالت Waiting برود. (مثل انتظار برای I/O)
۲- یک پردازه از حالت Running به Ready برود. (مثلاً وقتی یک interrupt رخ دهد)
۳- یک پردازه از حالت Waiting به Ready برود. (مثلاً بعد از اتمام I/O)
۴- یک پردازه Terminate شود. (یعنی آماده خروج از لیست پردازه‌ها شود)

در حالات ۱ و۴ با توجه به اینکه انتخابی وجود ندارد مفهوم زمان‌بندی نیز بی معناست. در حالات ۲ و۳، باید یک پردازه از حالت Ready برای اجرا انتخاب شود. در حالت ۱ و۴ در اصطلاح گفته می‌شود: زمان‌بندی از نوع none-Preemptive یا Cooperative (غیر رقابتی) است و در حالت ۲ و۳ زمان‌بندی از نوع Preemptive است.

طبیعتاً در زمان‌بندی none-Preemptive وقتی CPU به یک پردازه اختصاص داده شد، پردازه، CPU را تا زمانی که پایان یابد و یا به حالت Waiting برود، در اختیار می‌گیرد. در روش Preemptive، CPU تا پایان مقطع زمانی خاصی در اختیار پردازه است.

متود none-Preemptive در Windows 3.x به کار گرفته شد اما در Win 95 و تمامی ویندوزهای بعدی، Preemptive استفاده شد.

در روش Preemptive مشکلاتی ممکن است رخ دهد. از جمله اینکه یک پردازه بخواهد اطلاعاتی را آپدیت کند و زمانش تمام شود و CPU به پردازه دوم اختصاص یابد و پردازه دوم اجرا شود و بخواهد آن اطلاعات را بخواند! این مشکل در فصل «همزمانی پردازه‌ها» بحث خواهد شد...

الگوریتم‌های زمان‌بندی:

شرایط مقایسه الگوریتم‌ها:

برای زمان‌بندی CPU، الگوریتم‌های مختلفی وجود دارد باید بتوان طبق معیارهایی آن‌ها را با هم مقایسه کرد.

معیار اول: CPU Utilization یا بهره‌وری از CPU:

هدف ما این است که CPU را تا حد ممکن Busy (مشغول) نگه داریم. این معیار از ۰ تا ۱۰۰درصد درجه‌بندی می‌شود. در یک سیستم واقعی این محدوده باید بین ۴۰ تا ۹۰ درصد باشد.

معیار دوم: Throughput:

Throughput یعنی تعداد پردازه‌هایی که در هر واحدِ زمان تمام می‌شوند:

The Number of processes that are completed per time unit.

برای پردازه‌های طولانی ممکن است throughput برابر با «یک پردازه در ساعت» باشد و برای پردازه‌های سبک مثلاً «۱۰ پردازه در ثانیه» باشد.

معیار سوم: Turnaround Time یا زمان گردش کار:

The interval from the time of submission of a process to the time of completion.

فاصله زمانی از لحظه ورود یک فرآیند تا لحظه پایان آن.

زمان گردش کار= زمان‌هایی که برای انتقال به حافظه RAM سپری شده + زمان‌های سپری شده در صف Ready + زمان پردازش و اجرا روی CPU + زمان‌های I/O

معیار چهارم: Waiting Time یا زمان انتظار:

جمع زمان‌هایی که یک پردازه در صف Ready منتظر مانده است.

توجه: زمان سپری شده برای اجرا و I/O ربطی به الگوریتم‌های زمان‌بندی ندارد.

معیار پنجم: Response time یا زمان پاسخ گویی:

The time from the submission of a request until the first response is produced.

فاصله زمانی از لحظه ایجاد یک درخواست تا لحظه تولید اولین پاسخ.

توجه: turnaround time، زمان انتظار تا پایان یک پردازه بود. این معیار، معیار خوبی نخواهد بود چرا که سرعت دستگاه‌های ورودی خروجی ممکن است پایین باشد و این زمان را تحت تاثیر قرار دهد، در حالی که این مشکل ربطی به الگوریتم زمان‌بندی ندارد.

نتیجه: ما خواهان افزایش CPU Utilization و Throughput و کاهش سه معیار دیگر هستیم.

لیست الگوریتم‌های زمان‌بندی:

1- First-Come-First-Served = FCFS [First-In-First-Out = FIFO]
2- Shortest-Job-First = SJF
3- Shortest-Remaining-Time first = SRT
4- Priority Scheduling
5- Round-Robin = RR
6- Multi-level Queue

۱- الگوریتم FCFS:

اولین پردازه‌ای که CPU را تقاضا می‌کند، اولین پردازه‌ای است که CPU به آن اختصاص داده می‌شود.
این الگوریتم به سادگی با یک صف FIFO پیاده سازی می‌شود.
به محض اینکه CPU آزاد شد، پردازه‌ای که در ابتدای صف Ready است، CPU را در اختیار می‌گیرد.

مزایا:

- ساده‌ترین روش است
- کد نویسی و پیاده سازی آن بسیار ساده است.

معایب:

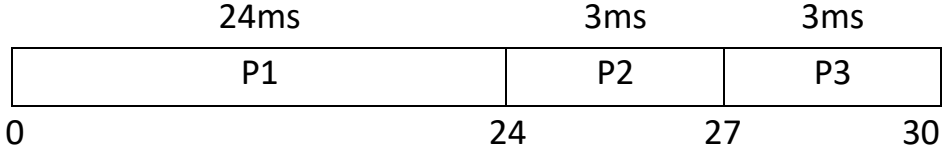
- زمان انتظار (waiting time) آن بسیار طولانی است.

مثال) پردازه‌های زیر را در نظر بگیرید که در زمان 0 به CPU می‌رسند. میانگین زمان انتظار را با استفاده از الگوریتم FCFS حساب کنید.

process	Burst time
P1	24
P2	3
P3	3

پاسخ:

چون پردازه‌ها به ترتیب P1 و P2 و P3 می‌رسند، پس داریم:



زمان انتظار P1=0

زمان انتظار P2=24 →Average Waiting Time = $\frac{0+24+27}{3}$=17

زمان انتظار P3=24+3=27

نکته: اگر همین پردازه‌ها به ترتیب P2 و P3 و P1 اجرا شوند، خواهیم داشت:

زمان انتظار P2=0

زمان انتظار P3=3 → Average Waiting Time = $\frac{0+3+6}{3}$=3

زمان انتظار P1=6

یعنی فقط با جابجایی یک پردازه، میانگین زمان انتظار بسیار کاهش می‌یابد. می‌توان نتیجه گرفت که این الگوریتم بهینه نیست و نسبت به طول پردازه‌ها بسیار متفاوت می‌باشد.

نکته: این الگوریتم یک الگوریتم غیر رقابتی است. هر پردازه وقتی CPU را در اختیار گرفت تا انتهای اتمام اجرایش CPU را در اختیار دارد.

۲- الگوریتم SJF (کوتاه‌ترین کار، اولین):

این الگوریتم به دو صورت پیاده می‌شود:

۲-۱- نوع none-Preemptive (غیر پس گرفتنی) این الگوریتم:

هر پردازه با کوتاه‌ترین زمان اجرا (Burst time) در ابتدای صف قرار می‌گیرد.

نکته کنکوری: اگر دو پردازه با زمان اجرای یکسان به CPU وارد شوند. برای تعیین اولویت آن‌ها از FCFS استفاده می‌شود. یعنی هر کدام زودتر رسیده بود، زودتر CPU را در اختیار می‌گیرد.

نکته۲: نام اصلی این الگوریتم Shortest-Next-CPU-Burst است.

مثال) پردازه‌های زیر را بدون در نظر گرفتن زمان ورود، با روش SJF غیرپس‌گرفتنی مرتب کرده، سپس AWT را محاسبه کنید.

Process	Burst Time
P1	6
P2	8
P3	7
P4	3

3ms	6ms	7ms	8ms
P4	P1	P3	P2

زمان انتظارP4	0
زمان انتظارP1	3
زمان انتظارP3	3+6=9
زمان انتظارP2	3+6+7=16

→ Average Waiting Time= $\frac{0+3+9+16}{4} = 7$

با الگوریتم FCFS:

6ms	8ms	7ms	3ms
P1	P2	P3	P4

زمان انتظارP1	0
زمان انتظارP2	6
زمان انتظارP3	6+8=14
زمان انتظارP4	6+8+7=21

→Average Waiting Time=$\frac{0+6+14+21}{4}$=10.25

مزایای SJF:

- این الگوریتم به خاطر انتقال پردازه‌های طولانی به عقب صف و راه اندازی سریع‌تر پردازه‌های کوتاه، Optimal یا بهینه است.

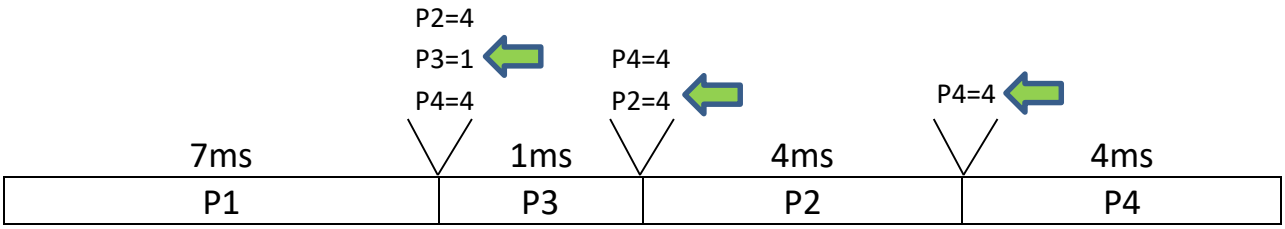
معایب:

- مهم‌ترین مساله در این الگوریتم دانستن طول پروسه بعدی است. باید با استفاده از فرمول‌هایی، مدت پروسه بعدی را تخمین بزنیم که این محاسبات پردازش بیشتر یا سربار بیشتری را می‌طلبد.

نتیجه: این الگوریتم بیشتر برای Batch systemها مفید است.

مثال دوم) Average Waiting Time را با در نظر گرفتن زمان ورود با الگوریتم SJF به دست آورید.

Process	Arrival Time	Burst Time
P1	0.0	7
P2	2.0	4
P3	4.0	1
P4	5.0	4



زمان انتظار P1	0
زمان انتظار P3	7-4=3
زمان انتظار P2	8-2=6
زمان انتظار P4	12-5=7

→ Average Waiting Time= $\frac{0+3+6+7}{4}$=4

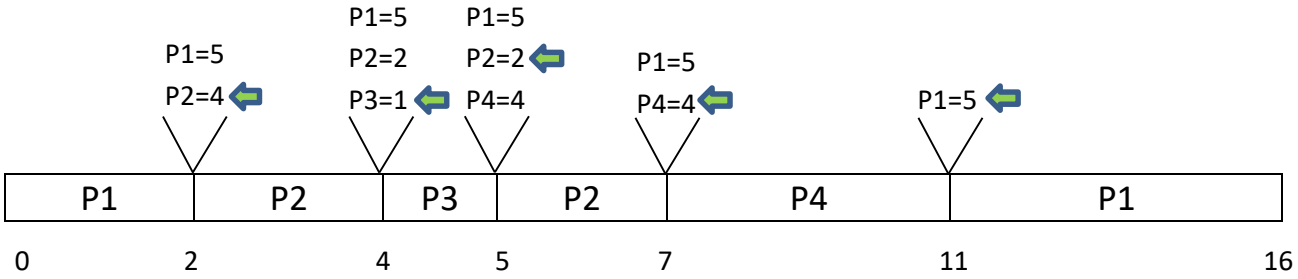
۲-۲- نوع Preemptive (پس گرفتنی) این الگوریتم:

اگر پردازه‌ای در حال اجرا باشد (P1) و در حین اجرا پردازه دوم (P2) برسد، باقیمانده‌ی پردازه‌ی اول با کلِ زمانِ پردازه‌ی دوم مقایسه می‌شود، اگر آن زمانِ باقیمانده، کمتر از زمانِ کلِ P2 باشد، به اجرا ادامه می‌دهد. اما اگر زمانِ باقیمانده‌ی P1 بیشتر از P2 باشد CPU از P1 پس گرفته می‌شود و به P2 اختصاص داده می‌شود.

به این الگوریتم که بر اساس زمان باقیمانده پردازه‌ها عمل می‌کند، **SRT یا Shortest-Remaining-Time** نیز گفته می‌شود.

مثال) همان مثال قبل را با الگوریتم SJF، اما از نوع پس گرفتنی(SRT) مرتب کنید و AWT را محاسبه کنید.

Process	Arrival Time	Burst Time
P1	0.0	7
P2	2.0	4
P3	4.0	1
P4	5.0	4



زمان انتظارP1	0+(11-2)=9
زمان انتظارP2	0+(5-4)=1
زمان انتظارP3	0
زمان انتظارP4	7-5=2

→ Average Waiting Time= $\frac{9+1+0+2}{4}$=3

۳- الگوریتم Priority (اولویتی):

در این الگوریتم یک عدد به عنوان میزان اهمیت و اولویت به هر پردازه نسبت داده می‌شود. (عدد کوچکتر=اولویت بالاتر) یعنی CPU به پردازه با بالاترین اولویت اختصاص داده می‌شود.

نکته کنکوری: SJF حالت خاصی از این الگوریتم است. (در مورد این جمله فکر کنید...)

مثال) AWT را برای مثال زیر با استفاده از الگوریتم اولویتی محاسبه کنید.

Process	Burst time	Priority
P1	10	3
P2	1	1
P3	2	4
P4	1	5
P5	5	2

زمان انتظارP2	0
زمان انتظارP5	1
زمان انتظارP1	6
زمان انتظارP3	16
زمان انتظارP4	18

$\rightarrow$Average Waiting Time=$\frac{0+1+6+16+18}{5}$=8.2

عیب بزرگ الگوریتم Priority:

Starvation or Indefinite Blocking (**گرسنگی** یا بلاک شدن بی نهایت)

پردازه‌های با اولویت بالا مدام وارد می‌شوند و در ابتدای صف قرار می‌گیرند. بنابراین ممکن است یک پردازه برای مدتی نامحدود در حالت بلاک باشد.

البته راه حلی برای این مشکل با عنوان Aging (سن دهی) وجود دارد.

برای مثال اگر اولویت پردازه از ۰ تا ۱۲۷ رتبه‌بندی شده است می‌توانیم پردازه‌ای را که در حالت انتظار است را هر ۱۵ ثانیه، یک اولویت بالاتر ببریم.

۴- الگوریتم Round Robin (نوبت دهی چرخشی):

- برای سیستم‌های Time Sharing طراحی شده است.
- شبیه الگوریتم FCFS عمل می‌کند اما ویژگی «پس گرفتنی» نیز به آن اضافه شده است.
- مانند صف نانوایی است که به هر نفر مثلاً ۲تا نان بدهیم و این کار را آن قدر تکرارکنیم تا همه به تعداد مورد نظرشان نان بگیرند.

توضیح روش کار:

یک واحدِ زمانی بسیار کوچک به نام Time Quantum یا Time Slice در نظر گرفته می‌شود. (معمولاً بین ۱۰ تا ۱۰۰ میلی ثانیه) پردازه‌های صف Ready از نوع FIFO در نظر گرفته می‌شود. یعنی پردازه‌های جدید در انتهای صف قرار می‌گیرند. CPU Scheduler اولین پردازه را از ابتدای صف برمی‌دارد و CPU را به آن اختصاص می‌دهد اگر Burst time مربوط به پردازه‌ی انتخاب شده، کمتر از یک Quantum باشد خودش بعد از آن زمان، CPU را آزاد می‌کند اما اگر بیش از یک Quantum باشد، یک interrupt به سیستم عامل فرستاده می‌شود سیستم عامل یک Context Switch را اجرا می‌کند و این ماژول، پردازه را در انتهای (Tail) صف قرار می‌دهد و سپس پردازه بعدی توسط CPU Scheduler انتخاب می‌شود.

نکته کنکوری: اگر در این الگوریتم، q بسیار بزرگ در نظر گرفته شود این الگوریتم شبیه FCFS عمل می‌کند و اگر بسیار کوچک تصور شود به دلیل اجرای بیش از حد Dispatcher سربار اضافی به CPU تحمیل می‌شود.

مزایا:

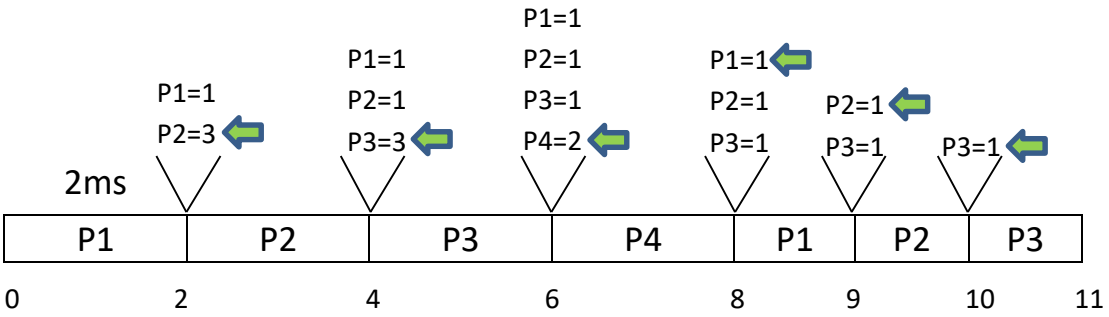
- زمان پاسخ گویی نسبتاً مناسب
- غیر انحصاری بودن
- عدم نیاز به دانستن مقاطع زمانی پردازه‌ها

معایب:

- میانگین زمان انتظار، بالاست.

مثال) در مثال زیر AWT را با الگوریتم RR محاسبه کنید؟ (q=2)

Process	Burst Time
P1	3
P2	3
P3	3
P4	2



زمان انتظار P1	0+(8-2)=6
زمان انتظار P2	2+(9-4)=7
زمان انتظار P3	4+(10-6)=8
زمان انتظار P4	6

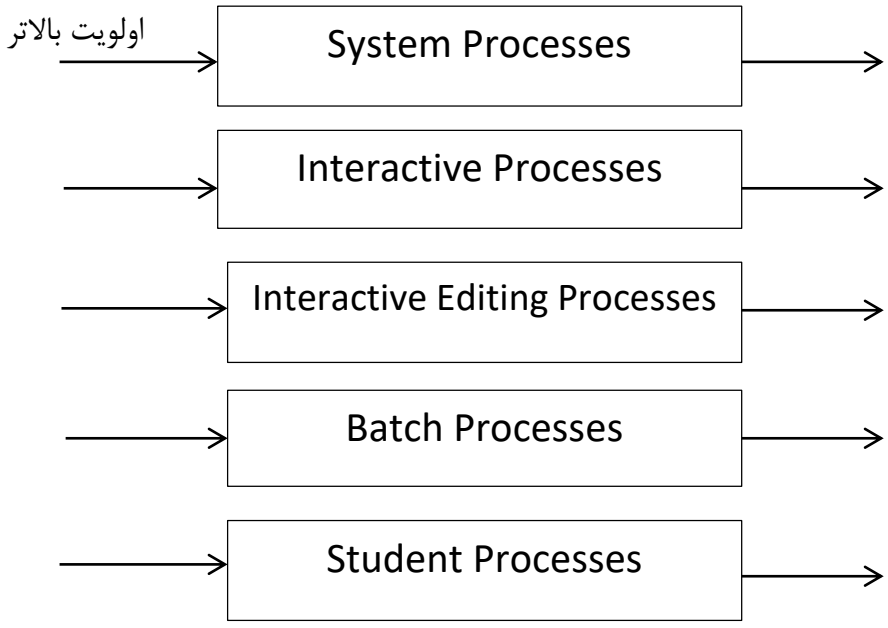
$$\text{Average Waiting Time} = \frac{6+7+8+6}{4} = 8$$

۵- الگوریتم Multi-level Queue Scheduling یا زمان‌بندی چند سطحی:

در مواردی که بتوان پردازه‌ها را به گروه‌های مختلف تقسیم کرد و هر گروه بتواند با نوعی زمان‌بندی متفاوت عمل کند از این الگوریتم استفاده می‌شود. (مثلاً برنامه‌هایی که کاربر در حال کار با آن‌هاست طبیعتاً اولویت بیشتری دارند، تا برنامه‌های پشت صحنه سیستم)

این الگوریتم صف Ready را بر اساس معیارهایی مثل میزان حافظه مورد نیاز، اولویت یا نوع پردازه به چندین صف مجزا تقسیم می‌کند. هر صف الگوریتم زمان‌بندی مخصوص به خود را دارد. مثلاً ممکن است پردازه‌های Foreground از الگوریتم RR استفاده کنند و پردازه‌های Background از FCFS.

باید الگوریتمی نیز برای برنامه‌ریزی صف‌ها وجود داشته باشد که این الگوریتم، یک الگوریتم بر اساس اولویتِ پس‌گرفتنی است. مثلاً صف پردازه‌های Foreground نسبت به Background اولویت دارند یعنی اگر پردازه‌ای از نوع Background در حال اجرا باشد و یک پردازه از نوع Foreground وارد شود CPU از Background پس گرفته می‌شود و به Foreground اختصاص داده می‌شود.



تمرین و تست:

۱- کدامیک از الگوریتم‌های زیر می‌تواند گرسنگی (Starvation) ایجاد کند؟

الف. FCFS ب. SJF ج. Priority د. RR

۲- پردازه‌های زیر را در نظر بگیرید. میانگین زمان انتظار را برای آن‌ها در الگوریتم‌های FCFS و RR محاسبه کنید:

Process	Burst time	Arrival Time
P1	3	0
P2	1	3
P3	4	3
P4	6	2

۳- سه پردازه P1، P2 و P3 با زمان اجرا و زمان وارد شدن آن‌ها در زیر آمده‌اند. متوسط زمان انتظار را برای چهار الگوریتم FCFS، SJF، SRT و Priority محاسبه کنید. (متن سؤال از: کنکور ارشد ۸۳)

	Priority	Arrival Time	Burst Time
P1	2	t	4
P2	0	t	2
P3	1	t+3	1

Process Synchronization

همزمانی پردازه‌ها

- **پردازه همکار یا Coordinating Process:** پردازه‌ای است که می‌تواند پردازه دیگری را تحت تأثیر قرار دهد و یا اینکه از پردازه دیگری تأثیر بپذیرد.
- **پردازه‌های** همکار ممکن است داده‌هایی را بین خود به اشتراک بگذارند و یا به فایل‌ها و یا داده‌هایی به صورت اشتراکی دسترسی داشته باشند.
- دسترسی همزمان به داده‌های به اشتراک گذاشته شده ممکن است موجب تناقض و بی ثباتی (inconsistency) شود در نتیجه به مکانیزم‌هایی نیاز است تا با اجرای صحیح پردازنده‌های همکار، از این مشکل جلوگیری به عمل آید.

تشریح مشکل: پردازه‌های P1 وP2 را در نظر بگیرید P1 در بخش تولید محصول اجرا می‌شود و پردازه P2 مربوط به بخش تحویل سفارش است.

پردازه P1 به محض تولید محصول، یک واحد به تعداد کل محصولات می‌افزاید و پردازه P2 پس از تحویل محصول، یک واحد از تعداد کل محصولات می‌کاهد.

P1:	P2:
Produce;	Counter--;
Counter++;	Consume;

در سطح پردازنده، برای طراحی P1 چنین عبارتی خواهیم داشت:

```
Register1=Counter;
Register1=Register1+1;
Counter=Register1;
```

برای P2 داریم:

```
Register2=Counter;
Register2=Register2-1;
Counter=Register2;
```

اگر فرض کنیم تعداد کل محصولات ۵ باشد و هر بار دو عمل از عملیات‌های هر پردازه اجرا شود، با اجرای همزمان این دو پردازه داریم:

در لحظهT_0: Producer →Register1=Counter [Register=5]

در لحظهT_1: Producer → Register1=Register1+1 [Register=6]

در لحظهT_2: Consumer →Register2=Counter [Register2=5]

در لحظهT_3: Consumer →Register2=Register-1 [Register2=4]

در لحظهT_4: Producer →Counter=Register1 [Counter=6]

در لحظهT_5: Consumer →Counter=Register2 [Counter=4]

همانطور که می‌بینید در نهایت تعداد محصولات ۴ درج می‌شود که اشتباه است چون یک محصول افزوده و یک محصول نیز کاسته شد، پس کانتر باید همان ۵ باقی بماند...

به شرایطی شبیه این مثال که چندین پردازه به یک داده مشابه دسترسی دارند و در آن تغییر ایجاد می‌کنند و خروجی اجرا وابسته به این است که پردازه‌ها به چه ترتیبی اجرا شوند، **شرایط رقابتی** (Race Condition) گفته می‌شود.

شرایطی این چنین به وفور در هر سیستم عاملی رخ می‌دهد (چون پردازه‌های مختلف به منابع مشترکی دسترسی دارند) در نتیجه بحث Process Synchronization یا Coordination بحث مهمی در سیستم عامل است.

ناحیه بحرانی یا Critical-Section:

پردازه‌های مختلف قطعه کدهایی دارند که در این بخش از کدهایشان ممکن است برخی متغیرها را تغییر دهند، یک جدول در بانک اطلاعاتی را آپدیت کنند، در یک فایل بنویسند و ... این بخش از کدها را ناحیه بحرانی می‌نامند.

سیستم عامل باید الگوریتمی به کار بگیرد تا وقتی یک پردازه در ناحیه بحرانی خود در حال اجراست، هیچ پردازه دیگری اجازه ورود به ناحیه بحرانی را نداشته باشد.

هر پردازه قبل از ورود به ناحیه بحرانی، باید درخواست اجازه ورود به ناحیه بحرانی را صادر کند. قطعه کدی که این درخواست را پیاده سازی می‌کند Entry Section نام دارد.

بعد از اجرای ناحیه بحرانی ممکن است قطعه کدی برای اعلان خروج از ناحیه بحرانی وجود داشته باشد. که Exit Section نام دارد این بخش کاری می‌کند که پردازه‌های دیگر بتوانند وارد ناحیه بحرانی‌شان شوند.

مابقی کدها Remainder Section نامیده می‌شوند.

```
do {
    entry section
        critical section
    exit section
        remainder section
} while (TRUE);
```

Figure 6.1 General structure of a typical process P_i.

راه حل‌های مشکل ناحیه بحرانی:

راه حل‌های حل مشکل ناحیه بحرانی باید سه مورد زیر را تامین کنند:

شرط اول: Mutual Exclusion یا شرط انحصار متقابل:

اگر یک پردازه‌ی در حال اجرا در ناحیه بحرانی خود است، هیچ پردازه دیگری نباید وارد ناحیه بحرانی خود شود.

شرط دوم: Progress یا شرط پیشرفت:

اگر هیچ پردازه‌ای در حال اجرا در ناحیه بحرانی نباشد و چندین پردازه تقاضای ورود به ناحیه بحرانی را دارند، فقط پردازه‌هایی در تصمیم‌گیری برای ورود به ناحیه بحرانی شرکت داده می‌شوند که در حال اجرا در ناحیه Remainder Section نباشند و این انتخاب نباید به طور بی نهایت به تأخیر بیفتد. **به عبارت دیگر هیچ پردازشی نباید از بیرون ناحیه بحرانی خود امکان بلوکه کردن پردازش‌های دیگر را داشته باشد.**

شرط سوم: Bounded Waiting یا شرط انتظار محدود:

یک پردازه منتظر برای ورود به ناحیه بحرانی نباید به صورت نامحدود در حالت انتظار باقی بماند. (باید محدودیتی در تعداد اجازه‌های ورود به ناحیه بحرانی برای پردازه‌ها وجود داشته باشد تا پردازه‌ای که در انتظار است نوبت دهی شود)

راه حل Peterson

در نگاه اول ممکن است راه حل‌هایی برای حل مشکل ناحیه بحرانی به ذهن برسد؛ از جمله:

راه حل اول: می‌توان یک متغییر مشترک به نام turn (نوبت) تعریف کرد و در ابتدای ورود به ناحیه بحرانی مقدار turn را چک کرد. اگر پردازه P_i مقدار turn را چک کرد و مقدار آن i بود، اجازه ورود دارد. در غیر این صورت هیچ کاری انجام نمی‌شود.

پردازه P_i پس از خروج از ناحیه بحرانی برای اجازه به پردازه P_j، مقدار turn را j می‌کند.

```
Pi
⋮
while (turn<>i)
;// do    nothing
ناحیه بحرانی
turn=j;
⋮
```

```
Pj
⋮
while (turn<>j)
 ;//do nothing
ناحیه بحرانی
turn=i;
⋮
```

این الگوریتم شرط اول و سوم را دارد. اما شرط پیشرفت را ندارد. چون اگر به فرض، پردازه P_i بخواهد دو بار متوالی وارد ناحیه بحرانی شود چون در انتهای اجرای خود turn را برابر با j کرده است. بار دوم نخواهد توانست وارد ناحیه بحرانی شود.

راه حل دوم:

```
Pi
⋮
Flag[i]=true;
While(flag[j]==true)
  ; // do nothing
ناحیه بحرانی
Flag[i] False;
⋮
```

```
Pj
⋮
Flag[j]=true;
While(Flag[i]==true)
   ; // do nothing
ناحیه بحرانی
Flag[j]=False;
⋮
```

مشکل الگوریتم اول در این الگوریتم حل شده است. یعنی P_i هر چند بار بخواهد می‌تواند اجرا شود.

اما اگر پس از اجرای Flag[i]=true، پردازنده از P_i گرفته شود و به P_j داده شود، دستور Flag[j] =true اجرا می‌شود و در نتیجه نه P_i می‌تواند وارد ناحیه بحرانی شود و نه P_j. یعنی شرط پیشرفت (شرط دوم) برآورده نمی‌شود.

راه حل سوم: راه حل Peterson (ترکیبی از دو راه حل)

```
do {
    flag[i] = TRUE;
    turn = j;
    while (flag[j] && turn == j);

        critical section

    flag[i] = FALSE;

        remainder section

} while (TRUE);
```

Figure 6.2 The structure of process P_i in Peterson's solution.

این الگوریتم، هر سه شرط را دارد.

- در کل، حل ناحیه بحرانی با یک Lock امکان‌پذیر است.

روش‌های سخت افزاری حل مشکل ناحیه بحرانی:

روش‌های سخت افزاری نیز وجود داردکه استفاده از آن‌ها مؤثرتر و ساده‌تر است.

Hardware features can make any programming task easier and improve system efficiency.

در سیستم‌های تک پردازنده‌ای (uniprocessor) ساده‌ترین روش این است که به محض ورود یک پردازه به ناحیه بحرانی، interruptها را غیر فعال کنید در این حالت مطمئن خواهید بود که هیچ وقفه‌ای رخ نخواهد داد و پردازه مثل حالت none-preemptive پردازشگر را در اختیار خواهد داشت. این روش در سیستم‌های Multi-Processor کاربردی نخواهد داشت و به دلیل وقت‌گیر بودن باعث کاهش کارآیی سیستم می‌شود. (چون پیغام عدم دریافت وقفه باید به همه CPUها ارسال شود.

در سیستم‌ها Multi-Processor با تعریف دو تابع که عموماً TestAndSet() و Swap() نامیده می‌شوند، الگوریتمی طراحی و پیاده سازی می‌شود، که شروط سه‌گانه را تامین کند.

Semaphores (سمافورها)

پیاده سازی روش‌های سخت افزاری حل مشکل ناحیه بحرانی در برنامه‌ها برای برنامه نویسانِ برنامه‌های کاربردی، پیچیده است. برای رفع این مشکل می‌توانیم از ابزار همگام‌سازی‌ای به نام Semaphore استفاده کنیم.

یک سمافور مثل s یک متغییر integer است که فقط از طریق دو عملیات اتمی به نام‌های ()Wait و ()Signal قابل دسترسی است. تابع ()Wait به صورت زیر پیاده سازی می‌شود:

```
Wait(s)
{
  While (s<=0)
   ; // do nothing
  s--
}
```

و تابع Signal به صورت زیر پیاده‌سازی می‌شود:

```
Signal(s)
{
  S++
}
```

- *در مورد عملیات اتمی (Atomic Operation) تحقیق کنید.*

توجه: گاهی اوقات، تابع Wait را با P و تابع Signal را با V نشان می‌دهند.

انواع Semaphore:

۱-Counting Semaphore: که نوع عددی را می‌پذیرد.

۲-Binary Semaphore: که فقط مقادیر صفر و یک را می‌پذیرد.

نکته: به Binary Semaphore در اصطلاح Mutual Exclusion Locks گفته می‌شود. چرا که قفل‌هایی (Locks) هستند که امکان انحصار متقابل Mutual Execution را فراهم می‌کنند.

در حل مشکل ناحیه بحرانی از طریق Semaphoreها n پردازنده یک Semaphore از نوع Binary را به اشتراک می‌گذارند که در ابتدا مقدار 1 در آن است. یک Counting Semaphore نیز در نظر گرفته می‌شود که تعداد منابعِ در دسترس در آن قرار می‌گیرد.

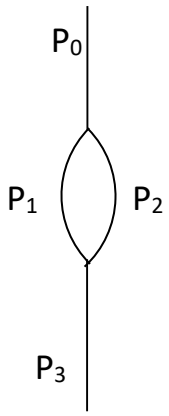
هر پردازه‌ای که منتظر استفاده از یک منبع است یک عملیات ()Wait روی Semaphore اجرا می‌کند (که در نتیجه تعداد منابع یکی کم می‌شود) و وقتی یک پردازه منبعی را آزاد می‌کند یک عملیات ()Signal اجرا می‌کند. (که در نتیجه یکی به تعداد منابع افزوده می‌شود).

وقتی تعداد سمافور Counting صفر می‌شود این بدان معنی است که همه منابع در حال استفاده است. بنابراین تمام پردازه‌هایی که انتظار استفاده از یک منبع را دارند. بلاک می‌شوند تا زمانی که تعداد، بیشتر از صفر شود.

```
do {
   waiting(mutex);

      // critical section

   signal(mutex);

      // remainder section
}while (TRUE);
```

Figure 6.9 Mutual-exclusion implementation with semaphores.

مثال) فرض کنید پردازه‌های P_0 وP_1 و P_2 وP_3 می‌خواهند به صورت همروند اجرا شوند با استفاده از Semaphoreها دستوراتی بنویسید که هماهنگی بین پردازه‌ها را به صورت شکل مقابل فراهم کنند.



پاسخ:

دو سمافور در نظر می‌گیریم:

Semaphore S_{12}=0 Semaphore S_3=0

در نتیجه، اگر پردازه‌ها به این صورت از سمافورها استفاده کنند، به صورت شکل بالا اجرا خواهند شد:

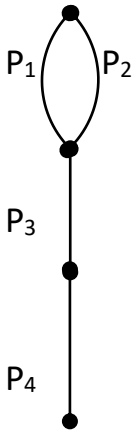
```
P0()
{
   ⋮ ناحیه بحرانی
   Signal (S12)
   Signal (S12)
}
```

```
P1()
{
   Wait(S12)
    ⋮
   Signal (S3)
}
```

```
P2()
{
   Wait(S12)
   ⋮
   Signal (S3)
}
```

```
P3()
{
   Wait(S3)
   Wait(S3)
   ⋮
}
```

تمرین: با استفاده از سمافورها، طوری برنامه‌ریزی کنید که پردازه‌های P_1 و P_2 و P_3 و P_4 به صورت شکل زیر اجرا شوند:



Dead lock

بن بست

در یک محیط چند برنامگی یا (Multi-Programming) چندین پردازه ممکن است برای تعداد نامحدودی منبع رقابت کنند.

یک پردازه درخواست منابع مورد نیازش را صادر می‌کند، اگر منابع در دسترس نباشد پردازه به حالت waiting می‌رود.

گاهی اوقات یک پردازه منتظر، هرگز قادر به تغییر وضعیت نخواهد بود. چون منابع مورد نظر آن پردازه در اختیار یک پردازه دیگر در صف waiting است. به این حالت Dead lock گفته می‌شود.

بهترین مثال، تقابل دو قطار با یکدیگر است. هر دو به طور کامل متوقف می‌شوند و هیچ یک حرکت نخواهد کرد مگر این که دیگری برود.

مثالی دیگر: یک پردازه، درایو DVD را تقاضا می‌کند و پرینتر را در اختیار دارد و پردازه دیگری درایو DVD را در اختیار دارد و پرینتر را تقاضا می‌کند.

در این فصل به تشریح روش‌های برخورد و جلوگیری از بن بست‌ها می‌پردازیم.

در سیستم‌های Multi-Programming بیشترین بن بست‌ها ممکن است رخ دهد چون نخ‌های مختلف برای دستیابی به منابع مشترک مسابقه می‌دهند.

شرایط ایجاد بن بست:

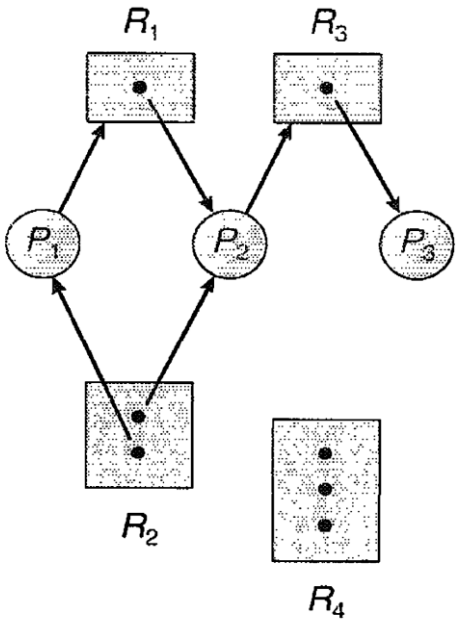
برای ایجاد موقعیت بن بست باید ۴ شرط به طور همزمان برقرار باشد.

۱- **انحصار متقابل (Mutual Exclusion):** حداقل یک منبع باید در حالت غیر اشتراک باشد، یعنی فقط یک پردازه بتواند از آن منبع در هر لحظه استفاده کند.
۲- **نگه داشتن و انتظار (Hold and Wait):** یک پردازه باید حداقل یک منبع در اختیار داشته باشد و نیاز به منابعی داشته باشد که در حال حاضر توسط پردازه‌های دیگر در اختیار گرفته شده باشد.

۳- **عدم پس گرفتن (none-Preemption)**: منابع باید غیر قابل پس‌گرفتنی باشد. یعنی یک منبع تنها زمانی می‌تواند آزاد شود که کار پردازه به طور کامل با آن منبع تمام شده باشد.

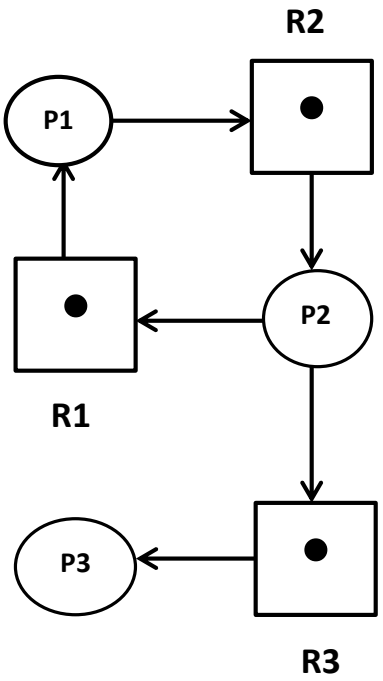
۴- **انتظار چرخشی (Circular Waiting)**: مجموعه پردازه‌های $\{P_0, P_1, ..., P_n\}$ که در صف انتظار قرار دارند. باید P_0 منتظر منبعی باشد که در اختیار P_1 است. P_1 منتظر منبعی است که در اختیار P_2 است. P_{n-1} منتظر منبعی است که در اختیار P_n است و P_n منتظر منبعی است که در اختیار P_0 است.

روش توصیف بن بست: استفاده از گراف تخصیص منابع (Resource Allocation Graph):

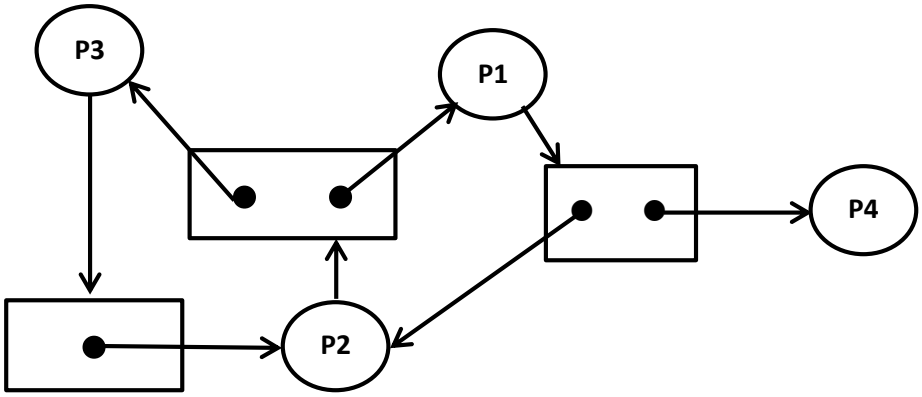


- گره‌ها از نوع منابع یا پردازه‌اند.
- منابع با مستطیل و تعداد نمونه‌های آن با نقطه داخل آن مشخص می‌شود.
- پردازه‌ها با دایره مشخص می‌شوند.
- یال‌های گرافِ تخصیص منابع، جهت‌دار هستند که یا از پردازه به منابع می‌باشند (بدین معنا که پردازه P_i منتظر پردازه P_j است) و یا از منبع به پردازه می‌باشند (بدین معنا که یک نمونه از منبع در اختیار پردازه P_i است)

مثال) در کدام یک از گراف‌های زیر بن بست رخ داده است؟

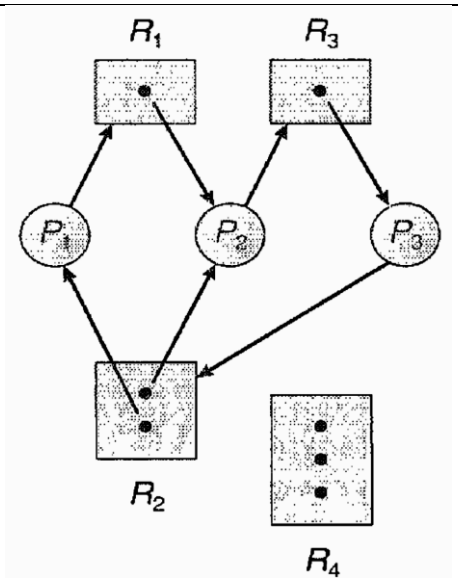


(بن بست هست)

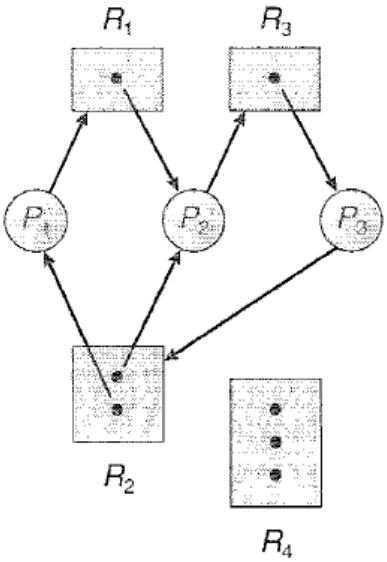


(بن بست نیست)

نکته: حلقه (loop) شرط لازم برای بن بست است اما کافی نیست.



گرافی با loop اما با بن بست.



گرافی با loop و بدون بن‌بست

تمرین: شکل زیر، دو نوع نمایش «گراف تخصیص منابع[1]» و «گراف انتظار-برای[2]» را برای یک شرایط نشان می‌دهد. آیا در این شرایط بن‌بست رخ خواهد داد؟

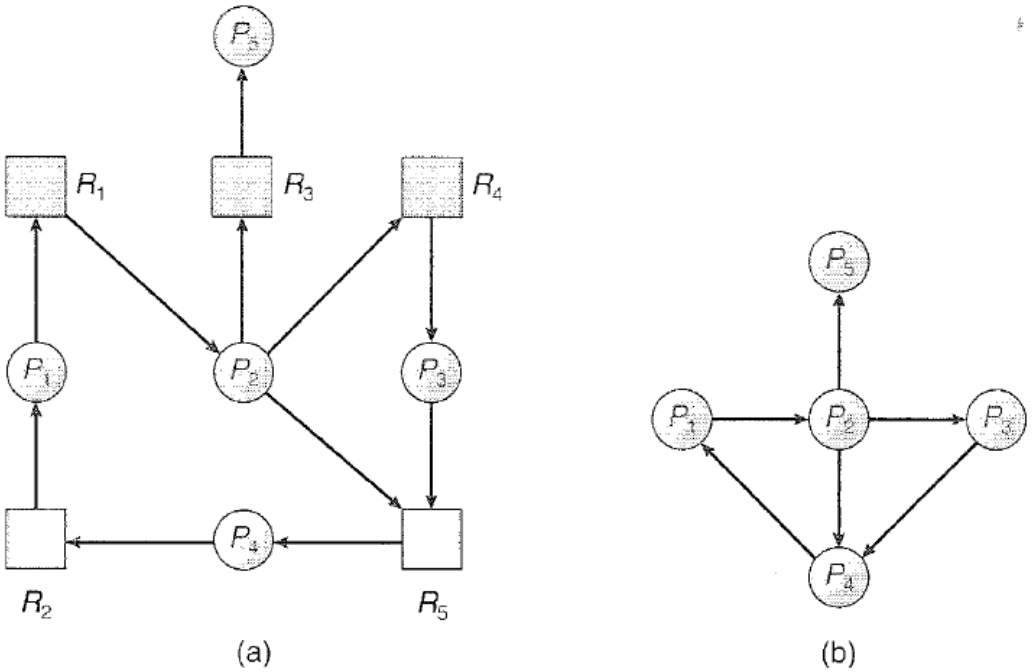


Figure 7.8 (a) Resource-allocation graph. (b) Corresponding wait-for graph.

روش‌های برخورد با بن بست:

به سه راه میتوان با بن بست برخورد کرد:

۱- می‌توان از پروتکلی برای پیشگیری و اجتناب از بن بست استفاده کرد، تا مطمئن شد که سیستم هرگز به حالت بن بست نمی‌رود.
۲- می‌توان به سیستم اجازه ورود به بن بست داد، سپس آن را تشخیص داده و برطرف کرد.
۳- می‌توان به طور کلی بحث برخورد با بن بست را در سیستم عامل مطرح نکرد و آن را از وظایف برنامه نویس‌ها دانست که برنامه‌هایی بنویسند که با بن بست مواجه نشود. در این نوع سیستم عامل‌ها مثل Unix و Windows و خیلی دیگر از سیستم عامل‌ها، سیستم به دلیل ورود به بن بست هنگ می‌کند و شما مجبورید به صورت دستی سیستم را Reset کنید.

[1] Resource-Allocation Graph
[2] Wait-for Graph

محققان معتقدند باید ترکیبی از این سه روش به کار گرفته شود.

پیشگیری از بن بست (Deadlock Prevention):

مجموعه راهکارهایی را فراهم می‌کند تا حداقل یکی از شرایط لازم اتفاق نیفتد.

برای اجتناب از بن بست، لازم است که یک سری اطلاعات اضافی مانند تعداد منابعی که یک پردازه در حین حیاتش درخواست یا استفاده خواهد کرد، پیش از اجرای پردازه بدانیم. در این صورت می‌توان تصمیم گرفت که چه پردازه‌ای اجرا شود تا چه پردازه‌ای منتظر بماند تا بن بست رخ ندهد.

الگوریتم بانکداران (Banker's Algorithm)

یکی از الگوریتم‌های مطرح در مورد پیشگیری از بن‌بست، الگوریتم بانکداران است. در این الگوریتم یک بانک شام تعداد منابع در دسترس و تعداد منابع مورد نیازِ هر پردازه و تعداد منابعی که هر پردازه فعلاً در اختیار دارد در نظر می‌گیریم و سپس با توجه به این بانک تصمیم می‌گیریم که کدام پردازه می‌تواند اجرا شود یا کدام نمی‌تواند.

دقیقاً مانند بانک که اگر پول نقدش رو به اتمام باشد، مجبور است ابتدا مشتریانی را راه بیندازد که پول نقد در اختیار دارند، در اینجا نیز ممکن است مجبور باشیم ابتدا پردازه‌هایی که منابع بیشتری در اختیار دارند را زودتر راه بیندازیم تا منابعشان را برای استفاده‌ی بقیه آزاد کنند...

مثال: با توجه به الگوریتم بانکداران، بررسی کنید در شرایط زیر، پردازه‌ها باید به چه ترتیبی اجرا شوند تا بن‌بست رخ ندهد؟

	Allocation	*Max*	*Available*
	A B C	*A B C*	*A B C*
P_0	0 1 0	7 5 3	3 3 2
P_1	2 0 0	3 2 2	
P_2	3 0 2	9 0 2	
P_3	2 1 1	2 2 2	
P_4	0 0 2	4 3 3	

در این مثال، A و B و C منابع ما هستند و Allocation تعداد منابع در اختیار پردازه‌ها را مشخص می‌کند و Max تعداد منابعی که نیاز دارند و Available تعداد منابع در دسترس را.

تمرین ۲:

	Allocation	*Need*	*Available*
	A B C	*A B C*	*A B C*
P_0	0 1 0	7 4 3	2 3 0
P_1	3 0 2	0 2 0	
P_2	3 0 2	6 0 0	
P_3	2 1 1	0 1 1	
P_4	0 0 2	4 3 1	

Memory Management

مدیریت حافظه

- همانطور که گفته شد جهت بالا بردن کارایی سیستم، برنامه‌ها و داده‌های مورد نیازشان باید به حافظه اصلی (RAM) منتقل شوند.
- پردازه‌ها در طول حیاتشان ممکن است بارها از روی دیسک به حافظه و از حافظه به دیسک منتقل شوند. پردازه‌های روی دیسک که منتظر انتقال به حافظه هستند در یک صف ورودی (Input Queue) قرار می‌گیرند و سیستم عامل یکی از پردازه‌ها را از این صف انتخاب و آن را روی Memory لود می‌کند. پردازه در حین اجرا به اطلاعات و دستورالعمل‌های مورد نیازش روی حافظه دسترسی پیدا می‌کند و در نهایت بعد از اجرا، حافظه‌ای که در اختیار داشته آزاد می‌کند.
- اطلاعات برنامه آدرس‌های مختلف حافظه (RAM) قرار می‌گیرند.
- به عمل نسبت دادن آدرس‌های حافظه به اطلاعات، Bind کردن (مقید کردن= انقیاد) گفته می‌شود.

تعریف آدرس منطقی و آدرس فیزیکی:

به آدرسی که توسط CPU تولید می‌شود، آدرس منطقی یا Logical Address، و به آدرسی که حافظه اصلی آن را می‌بیند یا می‌شناسد، آدرس فیزیکی یا Physical Address گفته می‌شود.

انواع Bind کردن حافظه:

۱- **Bind کردن در زمان کامپایل:**

اگر در زمان کامپایل بدانیم که پردازه در چه بخشی از حافظه قرار خواهد گرفت، می‌توان «کد مطلق» (Absolute Code) تولید کرد. یعنی مثلاً آدرس ۱۰۰ ذکر شده در برنامه همان آدرس ۱۰۰ مطلق حافظه خواهد بود.
برنامه‌های با فرمت com. تحت سیستم عامل DOS اینگونه هستند.

۲- **Bind کردن در زمان بارگذاری (Load Time):**

اگر در زمان کامپایل ندانیم که برنامه در کجای حافظه قرار خواهد گرفت، در این صورت کامپایلر موقتاً باید کد قابل جابجایی Relocatable Cod تولید کند و انتساب کد مطلق را تا زمان بارگذاری به تأخیر بیندازد.
به طور مثال اگر در برنامه‌ای دستور mov AL,[100] را داشته باشیم (این دستور یعنی محتوای صدمین خانه را به رجیستر AL لود کن) و برنامه‌ی حاویِ این دستور، در زمان بارگذاری، در محل ۳۰۰ حافظه قرار گیرد، در این صورت آدرس فیزیکیِ معادلِ آدرس ۱۰۰، برابر با ۴۰۰ است.

۳- **Bind کردن در زمان اجرا (Execution time):** اگر پردازه بتواند در حین اجرا از محلی از حافظه به محلی دیگر منتقل شود، آنگاه عمل انقیاد باید تا زمان اجرا به تأخیر بیافتد.

تعریف Virtual Address: در انقیاد در زمان کامپایل و باگذاری، آدرس‌های منطقی و فیزیکی، یکسان هستند در حالی که در انقیاد در زمان اجرا آدرس منطقی با آدرس فیزیکی متفاوت است در این حالت، به آدرس منطقی، آدرس مجازی گفته می‌شود. این آدرس مجازی (یا منطقی) باید در زمان اجرا به آدرس فیزیکی تبدیل شود. تا محل قرارگیری آن روی حافظه مشخص شود.

عمل تبدیل کردن آدرس مجازی به فیزیکی در زمان اجرا توسط یک قطعه سخت افزاری به نام MMU مخفف Memory Management Unit انجام می‌شود.

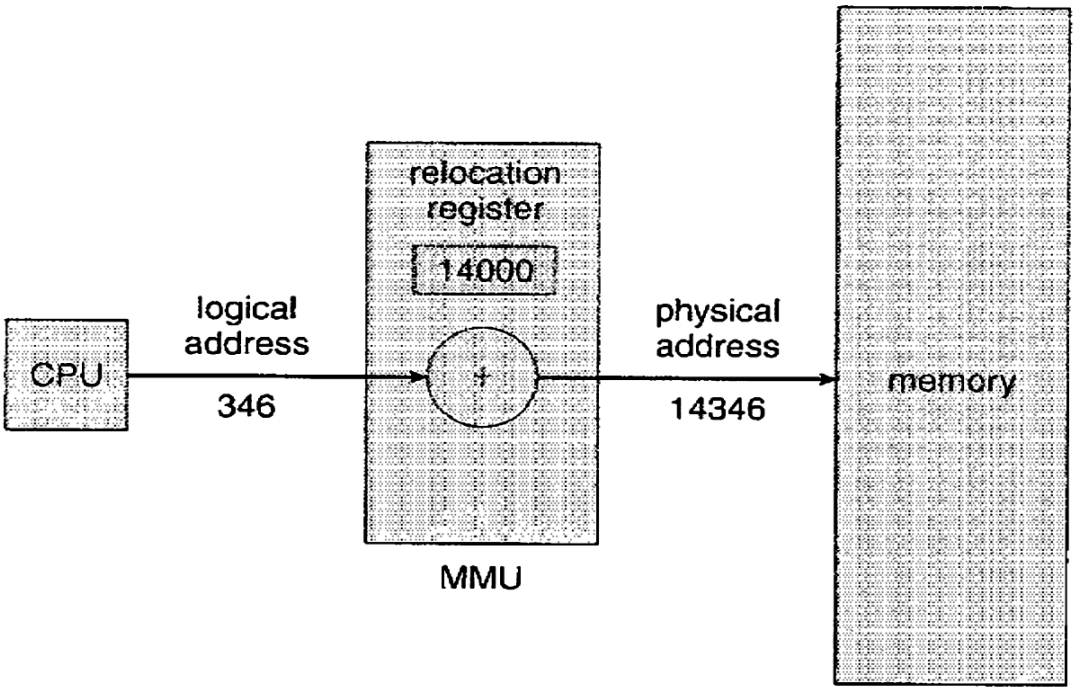


Figure 8.4 Dynamic relocation using a relocation register.

تخصیص حافظه (Memory Allocation):

یکی از ساده ترین روش‌های تخصیص حافظه به پردازه‌ها، تقسیم حافظه به چندین بخش با اندازه ثابت و غیر متغییر است که به روش Multi-Partition Method معروف است.
هر بخش به یک پردازه اختصاص داده می‌شود.
در این روش هرگاه یک پارتیشن خالی باشد یک پردازه از صف ورودی به آن پارتیشن لود می‌شود. و وقتی پردازه پایان می‌پذیرد، پارتیشن برای پردازه‌ای دیگر در دسترس قرار می‌گیرد.
هر چند این متد دیگر استفاده نمی‌شود، اما حالت تعمیم یافته آن در محیط‌های Batch و حتی Time Sharing قابل به‌کارگیری است.
برای پیاده سازی این روش سیستم عامل یک جدول (Table) در نظر می‌گیرد که در آن مشخص می‌شود که چه بخش‌هایی از حافظه، در دسترس و چه بخش‌هایی، اشغال می‌باشد.
در ابتدا، تمامِ حافظه، خالی و در دسترس است. یعنی یک بلاک بزرگ از حافظه (Hole) در دسترس در نظر گرفته شود. وقتی یک پردازه از راه می‌رسد سیستم عامل جدول را برای یافتن یک فضای خالی کافی برای آن پردازه جستجو می‌کند، اگر یافت شد همان مقدار که پردازه نیاز دارد حافظه به آن اختصاص داده می‌شود و بقیه برای تقضاهای بعدی در دسترس قرار می‌گیرد. هر پردازه به محض ورود به سیستم در یک (Input Queue) قرار می‌گیرد. در هر لحظه لیستی از اندازه‌های بلاک‌های خالیِ در دسترس، و لیستی نیز به عنوان صفِ ورودی وجود دارد.

سیستم عامل می‌تواند طبق یک الگوریتم خاص، این صف را مرتب کند:

- اگر پردازه‌ای با حافظه مورد نیازِ زیاد، به ابتدای صف برسد. اگر سیستم عامل یک بلاک به آن اندازه یافت، به آن اختصاص می‌دهد. در غیر این صورت می‌تواند منتظر بماند تا یک بلاک خالی شود و یا این که سیستم عامل می‌تواند صف را جستجو کند تا یک پردازه با حافظه کمتر را یافته و حافظه مورد نیازش را به آن اختصاص دهد.
- ممکن است بلاکی که برای یک پردازه در نظر گرفته می‌شود بسیار بزرگ باشد. در این صورت آن بلاک به دو بخش تقسیم می‌شود، بخشی برای پردازه و بخشی دیگر به لیست حافظه‌های خالی اختصاص داده می‌شود. اگر کار پردازه با حافظه اختصاص داده‌اش تمام شود، آن بلاک نیز به لیست فضاهای خالی در دسترس افزوده می‌شود. اگر چندین بلاک خالی، همجوار باشند با هم ادغام می‌شوند تا بلاک بزرگتری تشکیل شود. آن بلاک نیز به لیست فضاهای خالی افزوده می‌شود (ممکن است سیستم عامل صف پردازه‌های ورودی را چک کند، تا اگر پردازه‌ای منتظر بلاک خالی بزرگ است. بتواند حافظه مورد نیازش در اختیار بگیرد)

این انتخاب‌ها و اختصاص‌ها در سیستم عامل با عنوان «مسأله تخصیص حافظه پویا» (Dynamic Storage-Allocation Problem) مطرح می‌شود.

پس، **صورت مسأله** این است: چطور برای پردازه‌ای به اندازه n، یک فضای خالی از لیست فضاهای خالی انتخاب کنیم؟

روش‌های مختلفی برای این مشکل وجود دارد که سه روش بیشترین استفاده را دارد.

استراتژی‌های انتخاب یک فضای خالی از مجموعه فضاهای خالی:

۱- First-Fit: اولین فضای خالی که به اندازه کافی برای پردازه متقاضی بزرگ است اختصاص داده می‌شود جستجو می‌تواند از ابتدای مجموعه یا از جایی که جستجوی قبلی به پایان رسیده شروع شود.
۲- Best-Fit: کوچکترین فضایی که با پردازه متقاضی متناسب است اختصاص داده می‌شود. در این حالت کل جدول فضای خالی باید جستجو شود مگر این که جدول بر اساس size مرتب شده باشد. این استراتژی کوچکترین فضای خالی را تولید می‌کند.
۳- Worst-Fit: بزرگترین فضای خالی به پردازه متقاضی اختصاص داده می‌شود. این استراتژی بزرگترین فضای خالی استفاده نشده را تولید می‌کند.

نکته تستی: First-Fit سریعتر از Best-Fit عمل می‌کند.

چند تکه‌گی (Fragmentation): به دلیل لود شدن‌ها و حذف شدن‌های مکرر پردازه‌ها، فضای خالی حافظه نیز مکرراً به قطعات کوچکتری تقسیم می‌شوند. زمانی خواهد رسید که حافظه‌ی کافی برای تخصیص به یک پردازه وجود دارد، اما این حافظه‌های در دسترس، همجوار نیستند. در حقیقت حافظه به تعداد زیادی فضایِ کوچکِ غیر همجوار تقسیم شده است. در این حالت گفته می‌شود، چند تکه‌گیِ خارجی دارد.

نکته: استراتژی‌های First-Fit و Best-Fit هر دو مشکل چند تکه‌گی را دارند، اما Worst-Fit کمتر این مشکل را دارد.

راه حل‌های Fragmentation:

- یکی از راه حل‌های مشکل چند تکه‌گی، Compaction (فشردگی) است. به این معنا که محتویات حافظه را در کنار هم قرار دهیم تا فضاهای خالی به عنوان یک فضای خالی بزرگتر قرار می‌گیرد.

توجه: فشردگی همیشه ممکن نیست. اگر عمل انقیاد به صورت Static باشد یا اینکه در زمان کامپایل یا لود شدن صورت گرفته باشد فشردگی ممکن نیست. فشردگی تنها زمانی ممکن است که عمل انقیاد به صورت پویا (Dynamic) باشد و در زمان اجرا انجام شده باشد.

- راه حل دیگر این است که به آدرس‌های منطقیِ پردازه‌های یک برنامه اجازه دهیم که غیر همجوار باشند.

دو تکنیک برای این کار وجود دارد:

۱- **Paging** (صفحه‌بندی)
۲- **Segmentation** (قطعه‌بندی)

صفحه‌بندی (Paging):

در این روش حافظه فیزیکی به بلاک‌های با اندازه ثابتی به نام Frame و برنامه کاربر (حافظه منطقی) نیز به بلاک‌هایی با همان اندازه به نام Page شکسته می‌شود.

هنگامی که یک برنامه قصد اجرا شدن دارد Pageهای آن از روی هارد دیسک از روی Frameهای در دسترس لود می‌شود.

صفحه‌بندی این امکان را فراهم می‌کند که قسمت‌های یک برنامه (Pageهای برنامه) در حافظه، پراکنده باشد. (لازم نیست همجوار باشد)

روش پیاده‌سازی:

آدرس‌های تولید شده توسط CPU (آدرس‌های منطقی) به دو بخش تقسیم می‌شوند.

یک Page Number که با (p) نمایش داده می‌شوند و یک Page Offset که با (d) نمایش داده می‌شود. (offset یعنی فاصله هر خانه از ابتدای Page یا Frame)

Page Number به عنوان یک اندیس (index) برای اشاره به جدول صفحات (Page Table) استفاده می‌شود.

Page Table، یک دیتابیس شامل آدرس‌های اصلی صفحات در حافظه اصلی (RAM) است.

آدرس اصلی با Page Offset ترکیب می‌شود تا آدرس هر Page بر روی حافظه (RAM) مشخص می‌شود.

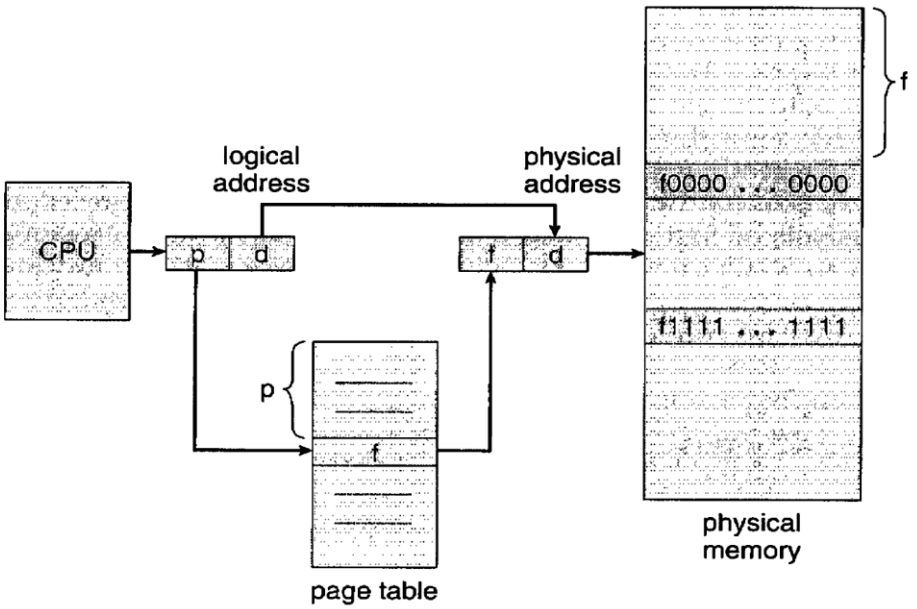
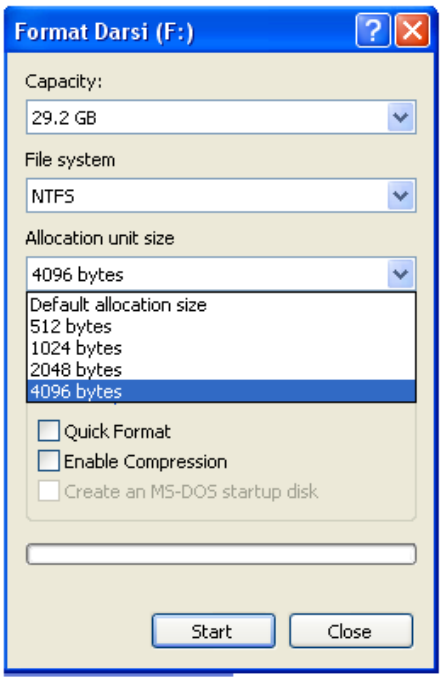
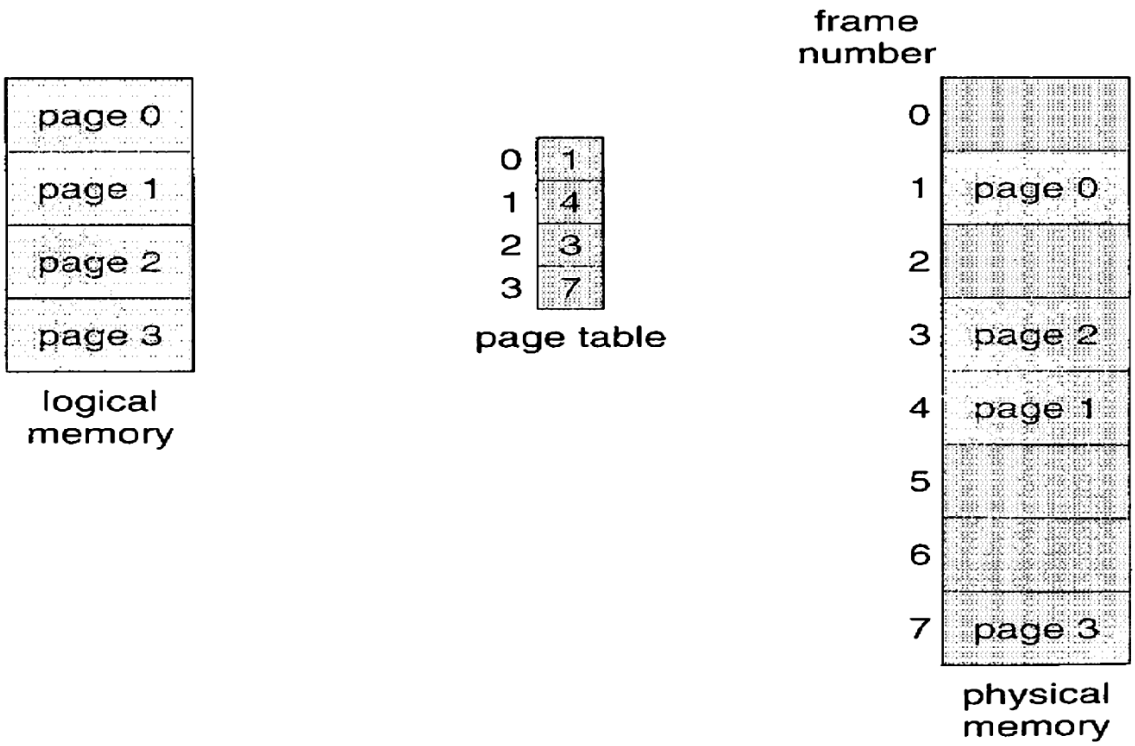


Figure 8.7 Paging hardware.

نکته: حجم یک صفحه توانی از دو است که بسته به معماری کامپیوتر از ۵۲ بایت تا ۱۶ مگابایت متفاوت است.



یک مثال از نحوه جادهی Pageهای برنامه روی Physical Memory با توجه به PageTable:



مثال) حافظه‌ای را در نظر بگیرید به اندازه ۳۲ بایت که اندازه صفحات در آن ۴ بایت است. با توجه به Page Table زیر، یک برنامه ۱۶ بایتی را در RAM (Physical Memory) جادهی کنید.

حل مسأله: می‌دانیم که حجم قاب‌های RAM و حجم صفحات برابر است. پس داریم:

Logical memory

0	A
1	B
2	C
3	D
4	E
5	F
6	G
7	H
8	I
9	J
10	K
11	L
12	M
13	N
14	O
15	P

0	5
1	6
2	1
3	2

Page table

Physical memory

۰	
۴	I J K L
۸	M N O P
۱۲	
۱۶	
۲۰	A B C D
۲۴	E F G H
۲۸	

قطعه‌بندی (Segmentation):

در صفحه‌بندی، پردازه‌ها به وسیله سخت افزار به صفحاتی به اندازه قاب‌های حافظه اصلی تقسیم‌بندی می‌شدند، اما در قطعه‌بندی کاربر می‌تواند به کمک کامپایلر، برنامه‌ی خود را به قطعات مختلفی تقسیم‌بندی کند.

مثلاً یک کامپایلر زبان C ممکن است قطعه‌های مجزایی برای بخش‌های زیر ایجاد کند:

- بخش کد
- متغیرهای Global
- stackهای به کار رفته توسط هر نخ
- Standard C library
- و ...

هر سگمنت یک نام و طول دارد. که برای سادگی در پیاده سازی به جای نام از شماره استفاده می‌شود.

کاربر یا کامپایلر برای هر خانه‌ی منطقیِ خود دو جزء مشخص می‌کند:

<Segment-Number و Offset>

Offset: فاصله خانه از ابتدای Segment

برای تبدیل این آدرس منطقیِ دو بعدی به آدرس فیزیکی یک بعدی از جدولی به نام Segment Table استفاده می‌شود. در این جدول هر رکورد شامل دو عدد است، یکی با نام Segment Base شناخته می‌شود که آدرسِ شروع Segment بر روی حافظه اصلی را مشخص می‌کند و یک عدد نیز با نام Segment-Limit شناخته می‌شود که طول سگمنت را مشخص می‌کند. (مزیت قطعه‌بندی بر صفحه‌بندی در همین «طول سگمنت» نهفته است)

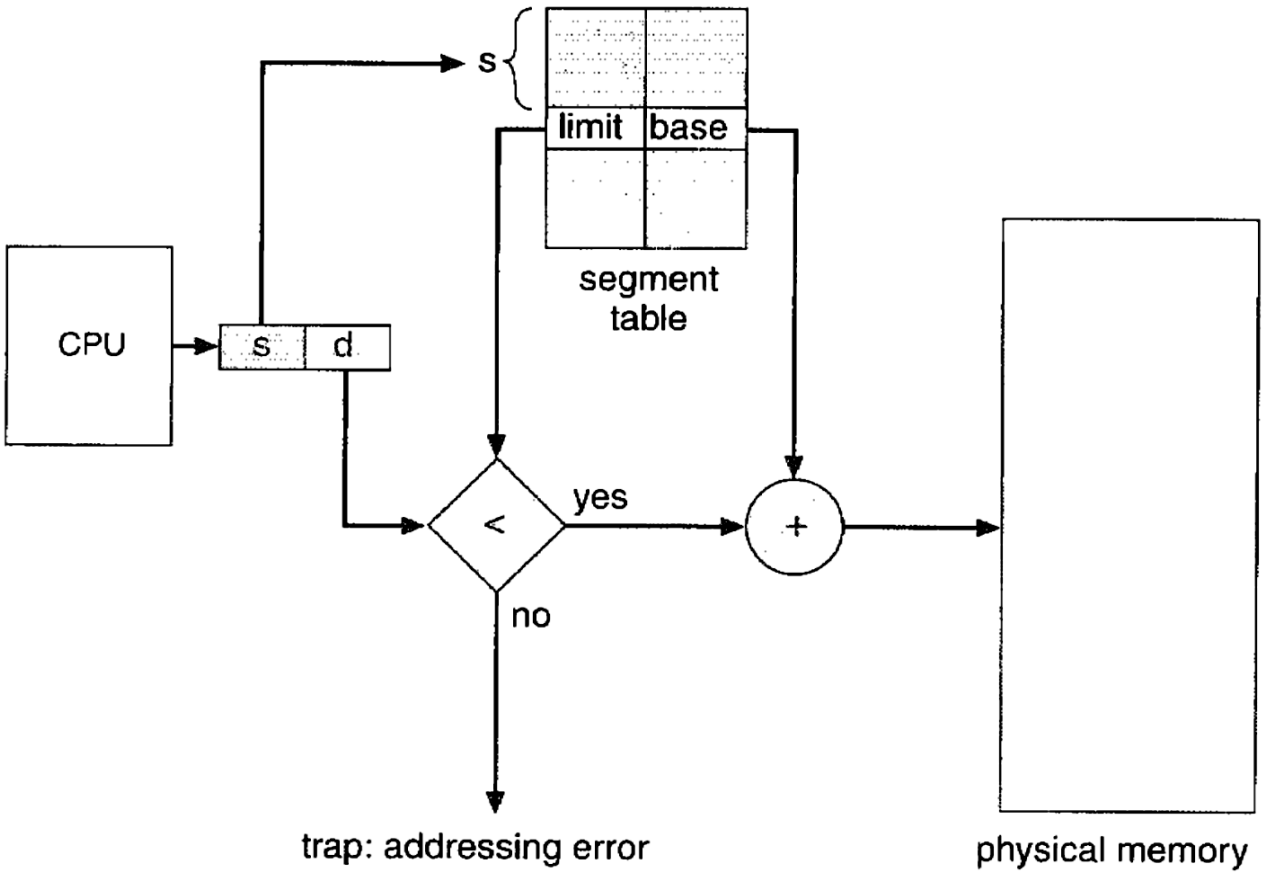


Figure 8.19 Segmentation hardware.

- **فرق Segmentation و Paging:**

در قطعه‌بندی، قطعات می‌توانند اندازه‌ها(طول‌ها)ی مختلفی داشته باشند. برای آدرس دهی هر خانه، شماره سگمنت به جدولِ قطعات ارسال می‌شود تا جای فیزیکی و limit آن مشخص شود. طبیعتاً Offset هر آدرسِ منطقی باید از limit کمتر باشد وگرنه آن قطعه بر روی بخش مورد نظر حافظه فیزیکی جا نمی‌شود.

پس قبل از تعیین آدرسِ فیزیکی چک می‌شود که Offset از Limit کمتر باشد در غیر این صورت هشدار داده می‌شود.

Distributed System Structures

ساختارهای سیستمیِ توزیع شده

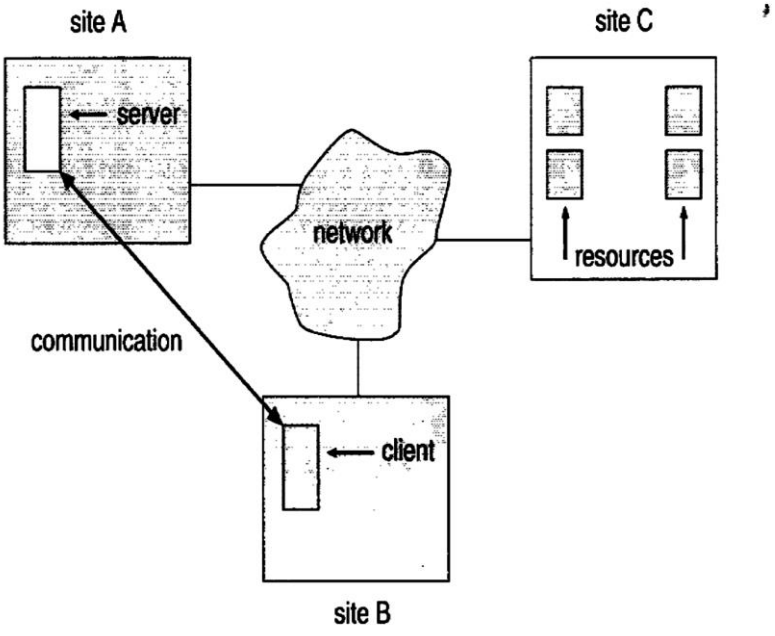
یک سیستم توزیع شده، مجموعه‌ای از پردازشگرهاست که حافظه یا clock اشتراکی ندارند، بلکه هر پردازشگر حافظه محلی مختص به خود را دارد. پردازشگرها از طریق شبکه‌های ارتباطی مختلف (مثل باس‌ها و خطوط تلفن) با یکدیگر تبادل داده دارند. پردازشگرهای مختلفِ یک سیستم توزیع شده ممکن است از نظر اندازه و عملکرد با هم متفاوت باشند. ممکن است سیستم‌های پردازشگر در نقاط مختلف، متفاوت باشد مثلاً یک گروه، Workstation باشند یا یک گروه Mini Computer و یا گروهی، کامپیوترهای عظیم چند منظوره باشند.

ممکن است به هر سیستمِ پردازشگر، نام‌های مختلفی نسبت داده شود. مثل:

Host, Machine, Computer Node, Site

اما معمولاً به منطقه‌ای که یک کامپیوتر قرار دارد، Site و به هر کامپیوترِ داخل آن منطقه، Host گفته می‌شود.

معمولاً یک Client (سرویس گیرنده) از یک سایت، قصد استفاده از منابع یک Server (سرویس دهنده) در سایت دیگر را دارد.



چهار دلیل عمده‌ی استفاده از سیستم‌های توزیع شده:

1- Resource Sharing

واضح است با اشتراک منابع، تمامی Hostها در Siteهای مختلف می‌توانند از منابع یکدیگر بهره ببرند.

در کل، یک سیستم توزیع شده مکانیزم کار برای:

۱- اشتراک فایل‌ها

۲- کار با دیتابیس‌های روی کامپیوترهای دیگر
۳- پرینت گرفتن فایل‌ها از راه دور
و... را فراهم می‌کند.

2- Computation Speedup

می‌توان با قابلیتی مثل Load Sharing اگر کارهای زیادی از یک Host انتظار می‌رود، تعدادی از آن jobها را به سیستم پردازشگر دیگری محول کرد.

3- Reliability

اگر یک سیستم از رده خارج شود، سیستم‌های دیگر در سایت‌های دیگر آماده به کارند.

4- Communication

ارتباط مراکز اصلی یک سازمان با شعبه‌های آن‌ها به راحتی ممکن خواهد شد.

انواع سیستم عامل‌های توزیع شده:

1- Network Operating Systems (NOS)
2- Distributed Operating Systems (DOS)

سیستم عامل‌های توزیع شده قابلیت‌های بیشتری ارائه می‌کند اما پیاده سازی آن‌ها و دسترسی کاربران به آن‌ها مشکل‌تر است.

۱- Network OSs:

سیستم عامل شبکه محیطی را فراهم می‌کند که کاربران امکان دسترسی از راه دور به منبع را داشته باشند (یا با login کردن و یا با انتقال اطلاعات از روی کامپیوتر دور به کامپیوتر خودشان)

- Remote login: Telnet امکانی در اینترنت است که امکان لاگین کردن به یک سیستم از راه دور را می‌دهد مثلاً فرض کنید کاربری در دانشگاهِ Westminster قصد دارد به کامپیوتری در دانشگاه Yale متصل شود او باید یک حساب کاربری بر روی آن سیستم داشته باشد سپس با وارد کردن آدرسی شبیه

```
telnet  cs.yale.edu
```

و وارد کردن نام کاربری خود یک Socket Connection بین دو کامپیوتر ایجاد می‌کند.

- Remote file transfer: مکانیزمی برای انتقال اطلاعات از راه دور از روی یک کامپیوتر به روی کامپیوتر دیگر است.

مثالی از یک اتصال FTP به یک سرور در راه دور:

```
ftp cs.yale.edu
get server.java
```

۲- Distributed OSs:

در یک سیستم عامل توزیع شده، کاربران می‌توانند همانطور که به یک منبع محلی (local) دسترسی دارند، به منابع remote (در راه دور) نیز دسترسی داشته باشند در حقیقت مهاجرت پردازه‌ها و داده‌ها از یک سایت به سایت دیگر تحت کنترل سیستم عامل توزیع شده است.

- **مفهوم Data migration**:

تصور کنید کاربری در سایت A قصد دارد به فایلی که در سایت B است دسترسی یابد. یک راه حل این است که فایل به سایت A منتقل شود و بعد از پایان کار، مجدداً کل فایل به سایت B منتقل شود حتی اگر فقط بخشی از فایل تغییر یافته باشد. (همان کاری که FTP انجام می‌دهد)

راه حل دیگر این است که فقط بخشی از فایل که نیاز است Transfer شود و اگر بخش‌های بیشتری نیاز بود، ادامه آن‌ها لود شوند. روشی که سیستم فایل Sun Microsystems از آن استفاده می‌کند.

یک نمونه سؤال از آزمون پایان دوره:

مدت زمان امتحان: ۱۲۰ دقیقه | تعداد صفحات: ۱ | تاریخ اعلام نمرات: ۸۸/۱۰/۳۰ | نام و نام خانوادگی:

۱. شما هم اکنون در پایان یک دوره «سیستم عامل» هستید. اگر از شما بپرسند "سیستم عامل چیست؟" جواب شما چه خواهد بود؟

۲. Buffering و Spooling را تعریف کرده و با هم مقایسه کنید.

۳. وضعیت‌های مختلف یک پردازه را با رسم شکل توضیح دهید.

۴. دلایل ختم یک پردازه را ضمن اشاره به مفهوم Cascading Termination بیان کنید.

۵. پردازه‌های زیر را بر اساس الگوریتم‌های FCFS، SJF، SRT و RR (q=2) نوبت‌دهی کرده و میانگین زمان انتظار در هر حالت را محاسبه کنید.
(رسم Time Line الزامی است)

Process	Arrival Time	Burst Time
P1	0	8
P2	1	4
P3	2	9
P4	3	5

۶. به اختیار، به یکی از سؤالات زیر پاسخ دهید:

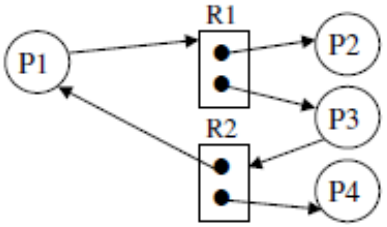
۶-۱- ناحیه بحرانی چیست؟ سه شرطی که راه حل‌های ناحیه بحرانی باید آن‌ها را تأمین کنند، نام برده و به اختصار توضیح دهید.

۶-۲- Deadlock را تعریف کنید و شرایط ایجاد آن را توضیح دهید.

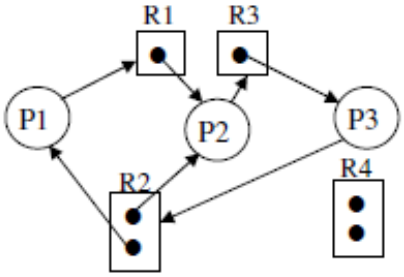
۷. سه راه حل که می‌توان ناحیه بحرانی را از طریق آن‌ها حل کرد، فقط نام ببرید.

۸. وضعیت بن بست را در گراف‌های زیر بررسی نمایید:

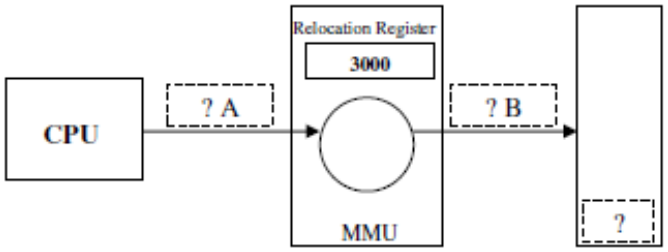
الف.



ب.



۹. شکل زیر را کامل کنید و با توجه به شکل، وظیفه MMU را بیان کنید و تعیین کنید اگر به جای A آدرس ۴۳۶ قرار گیرد، B چند خواهد بود؟



۱۰. استراتژی‌های انتخاب یک فضای خالی روی حافظه از بین فضاهای خالی در دسترس را نام برده، توضیح دهید.
سریع‌ترین استراتژی کدام است؟

۱۱. انواع سیستم عامل‌های توزیع شده را نام برده و توضیح مختصری برای هر یک بیان کنید. تفاوت آن‌ها در چیست؟

دانشجوی محترم: ارزشیابی این درس (از ۱۰۰) به شرح ذیل تقسیم بندی شده است.
فعالیت کلاسی: ۱۰ | تمرینات-میان ترم: ۳۰ | پایان ترم: ۶۰

موفق باشید؛
حمید رضا نیرومند