دانشگاه آزاد اسلامی واحد کرج

دانشکده مکاترونیک، گروه مهندسی کامپیوتر- نرم‌افزار

جزوه درس

سیستم‌های عامل

تهیه و تنظیم

ساسان حریفی

فهرست مطالب

فصل اول: مفاهیم و تعاریف اولیه

۱-۱. مقدمه ۲

۱-۲. نسلهای کامپیوتر ۲

۱-۲-۱. نسل اول (۵۵-۱۹۴۵) ۳

۱-۲-۲. نسل دوم (۶۵-۱۹۵۵) ۳

۱-۲-۳. نسل سوم (۸۰-۱۹۶۵) ۳

۱-۲-۳-۱. سیستم آنلاین اسپولینگ ۵

۱-۲-۳-۱-۱. سیستم ورودی ۵

۱-۲-۳-۱-۲. زمانبند کار (Job Scheduler) ۶

۱-۲-۳-۱-۳. پردازنده کار (Job Processor) ۶

۱-۲-۳-۱-۴. زمانبند خروجی ۶

۱-۲-۳-۱-۵. سیستم خروجی ۶

۱-۲-۳-۱-۶. مدیر دیسک ۶

۱-۲-۳-۲. اشتراک زمانی (Time Sharing) ۶

۱-۲-۴. سیستم عامل های کامپیوترهای شخصی و شبکه ۷

۱-۲-۴-۱. سیستم های توزیع شده (Distributed Systems) ۷

۱-۲-۴-۲. نخ (Thread) و سیستم های Multi Tasking ۷

۱-۲-۴-۳. سیستم های چند پردازنده ای (Multi Processor) ۸

۱-۲-۴-۳-۱. دسته بندی سیستم های چندپردازنده ای ۸

۱-۲-۴-۴. سیستم های بلادرنگ (Real Time OS) ۱۱

فصل دوم: ساختار سیستم عامل ها

۲-۱. بخش‌های تشکیل دهنده سیستم عامل‌ها ۱۳

۲-۱-۱. مدیریت پردازش و پردازنده ۱۳

۲-۱-۲. مدیریت حافظه اصلی و ثانویه ۱۳
۲-۱-۳. مدیریت فایل ۱۳
۲-۱-۴. مدیریت وسایل I/O ۱۴
۲-۲. وظایف فرعی سیستم عامل ۱۴
۲-۳. مفسر فرمان (Command Interpreter) ۱۴
۲-۴. تشخیص اتمام انتقال داده ها ۱۴
۲-۴-۱. روش سرکشی یا نمونه برداری یا Polling ۱۵
۲-۴-۲. روش وقفه یا Interrupt ۱۵
۲-۵. انواع وقفه ها ۱۵
۲-۵-۱. وقفه برنامه (Program Check) ۱۵
۲-۵-۲. وقفه های زمان سنج (Timer) ۱۶
۲-۵-۳. وقفه I/O ۱۶
۲-۵-۴. وقفه نقص سخت افزار (Hardware Defect) ۱۶
۲-۵-۵. وقفه SVC (Superviser Call) ۱۶
۲-۵-۶. وقفه Restart ۱۶
۲-۶. تقسیم بندی وقفه‌ها به صورت کلی‌تر ۱۶
۲-۷. مراحل عملیات I/O به کمک وقفه ها ۱۷
۲-۸. تعویض متن یا Context Switch ۱۸
۲-۹. انواع سیستم عامل ها از نظر ساختاری ۱۸
۲-۹-۱. سیستمهای یک پارچه (Monolithic) ۱۸
۲-۹-۲. سیستم های لایه ای ۱۹
۲-۹-۳. ماشین مجازی ۱۹
۲-۹-۴. مشتری/خدمتگذار Client/Server ۱۹

فصل سوم: پردازش و زمانبندی

۳-۱. زمانبند کار (Job Scheduler) ۲۱

۳-۲. حالات مختلف پردازش ۲۱
۳-۲-۱. حالت معلق (Suspend) ۲۲
۳-۳. حالات مختلف بین نخ و فرآیند ۲۳
۳-۴. حالات مختلف نخ ۲۴
۳-۵. زمانبندی در سیستم عامل ۲۴
۳-۶. انواع روش های زمانبندی ۲۴
۳-۶-۱. انحصاری (Non-Preemptive) ۲۴
۳-۶-۲. غیرانحصاری (Preemptive) ۲۵
۳-۷. معیارهای زمانبندی ۲۵
۳-۷-۱. کارایی پردازنده (CPU Performance) ۲۵
۳-۷-۲. زمان پاسخ (Response Time) ۲۵
۳-۷-۳. زمان بازگشت (TurnAround Time) ۲۶
۳-۷-۴. توان عملیاتی (Throughput) ۲۶
۳-۷-۵. زمان انتظار (Waiting Time) ۲۶
۳-۷-۶. عدالت (Fairness) ۲۶
۳-۸. الگوریتم های زمانبندی ۲۶
۳-۸-۱. FIFO (FCFO) ۲۶
۳-۸-۲. RR (Round Robin) ۲۷
۳-۸-۳. SJF (Shortest Job First) ۲۹
۳-۸-۴. SRT (Shortest Remaining Time) ۳۰
۳-۸-۵. HRRN (Highest Response Ratio Next) ۳۱
۳-۸-۶. اولویت (Priority) ۳۲
۳-۸-۷. صف های چندگانه (Multiple Queue) ۳۲
۳-۸-۸. صف های چندگانه با فیدبک (Multiple Feedback Queue) ۳۲
۳-۸-۹. شانسی (Lottery) ۳۲

۳-۸-۱۰. LPT (Longest Processing Time) ۳۳
۳-۹. محاسبه توان عملیاتی و راندمان CPU ۳۳
۳-۱۰. تکنیک Aging (سالمندی) ۳۳

فصل چهارم: بن بست

۴-۱. مفهوم بن بست ۳۵
۴-۲. شرایط لازم برای ایجاد بن بست (شروط کافمن) ۳۵
۴-۲-۱. انحصار متقابل (بالقوه) ۳۵
۴-۲-۲. گرفتن و منتظر ماندن (بالقوه) (Hold & Wait) ۳۵
۴-۲-۳. انحصاری بودن (بالقوه) ۳۵
۴-۲-۴. انتظار چرخشی (بالفعل) (Circular Wait) ۳۵
۴-۳. منبع بحرانی (Critical Resource) و بخش بحرانی (Critical Area) ۳۶
۴-۴. گراف تخصیص منبع ۳۶
۴-۵. استراتژی های مدیریت بن بست ۳۸
۴-۵-۱. صرف نظر کردن از بن بست ۳۸
۴-۵-۲. جلوگیری از بن بست ۳۸
۴-۵-۳. اجتناب از بن بست ۳۹
۴-۵-۳-۱. اجتناب از بن بست با گراف تخصیص منبع ۴۰
۴-۵-۳-۲. اجتناب از بن بست با الگوریتم بانکدار ۴۰
۴-۵-۴. کشف یا بازیابی بن بست ۴۴
۴-۵-۴-۱. کشف برای حالتی که چند نمونه از منبع وجود داشته باشد ۴۴
۴-۵-۴-۲. کشف برای حالتی که یک نمونه از منبع وجود داشته باشد ۴۵
۴-۵-۴-۳. خاتمه پردازش (بازیابی) ۴۶

فصل پنجم: مدیریت حافظه

۵-۱. مدیریت حافظه ۴۸

۵-۲. مدیریت حافظه در سیستم های قدیمی ۴۸
۵-۳. تقسیم بندی فیزیکی و منطقی حافظه ۴۸
۵-۳-۱. عمل نگاشت (Binding) ۴۹
۵-۴. روش های تخصیص حافظه ۴۹
۵-۵. روش های تخصیص حافظه به پردازش ۵۰
۵-۵-۱. First Fit ۵۰
۵-۵-۲. Next Fit ۵۰
۵-۵-۳. Best Fit ۵۰
۵-۵-۴. Worst Fit ۵۰
۵-۵-۵. Quick Fit ۵۰
۵-۵-۶. Buddy ۵۱
۵-۶. روش های تقسیم بندی منطقی حافظه ۵۲
۵-۶-۱. روش صفحه بندی (Paging) ۵۲
۵-۶-۲. روش قطعه بندی (Segmentation) ۵۲
۵-۷. مقایسه صفحه بندی و قطعه بندی ۵۳

فصل ششم: همروندی فرایند ها و نواحی بحرانی

۶-۱. دسته بندی محاوره و ارتباط بین فرایندها ۵۵
۶-۲. ناحیه بحرانی ۵۵
۶-۳. شرایط ارائه شده برای حل مساله ناحیه بحرانی ۵۶
۶-۴. راه حل های ارائه شده برای مساله ناحیه بحرانی ۵۶
۶-۴-۱. راه حل نرم افزاری ۵۶
۶-۴-۱-۱. استفاده از متغیر قفل (Lock) ۵۶
۶-۴-۱-۲. استفاده از متغیر تناوبی (نوبتی) ۵۷
۶-۴-۱-۳. استفاده از متغیر Flag (روش اول) ۵۷

۶-۴-۱-۴. استفاده از متغیر Flag (روش دوم)..... ۵۸
۶-۴-۱-۵. استفاده از روش پیترسون..... ۵۹
۶-۴-۲. راه های سخت افزاری..... ۵۹
۶-۴-۲-۱. از کار انداختن وقفه ها..... ۶۰
۶-۴-۲-۲. استفاده از دستورات TSL (Test and Set Lock)..... ۶۰
۶-۴-۲-۳. استفاده از دستورات مربوط به تعویض یا swap یا exchange..... ۶۱
۶-۴-۳. راه حل هایی که توسط سیستم عامل ارائه می شوند..... ۶۲
۶-۴-۳-۱. روش سمافور (Semaphore)..... ۶۲
۶-۴-۴. راه حل هایی که توسط زبان های برنامه نویسی ارائه می شوند..... ۶۴
۶-۴-۴-۱. روش مانیتور (Monitor)..... ۶۴
۶-۴-۵. راه حل تبادل پیام (Message Passing)..... ۶۴

فصـــل اول

مفاهیم و تعاریف اولیه

۱-۱. مقدمه

سیستم عامل (Operating System): اصلی‌ترین برنامه سیستمی است که به عنوان رابط بین کاربر و سخت افزار عمل می‌کند. سیستم عامل یا به اختصار (OS) دو وظیفه (هدف) دارد که به شرح زیر است:

۱- استفاده از کامپیوتر را ساده می‌کند یعنی کاربر یا حتی برنامه‌نویس بدون درگیر شدن با مسائل سخت‌افزاری می‌تواند فایلی را بر روی دیسک ذخیره کند. این کار با به کار بردن دستورات ساده‌ای مانند فراخوانهای سیستمی (System Calls) انجام می‌شود. سیستم عامل، سخت افزار را از دید کاربر یا برنامه‌نویس مخفی می‌سازد.
۲- هدف دوم سیستم عامل مدیریت منابع است یعنی سیستم عامل باعث استفاده بهینه و سودمند از منابع فیزیکی و منطقی سیستم می‌شود. منظور از منابع فیزیکی پردازنده، حافظه، دیسک، موس، چاپگر، پورت و ... است و منظور از منابع منطقی اطلاعات (داده‌ها)، فایل‌ها و ... است.

به عنوان مثال اگر دو برنامه همزمان بخواهند از چاپگر استفاده کنند، سیستم اطلاعات آنها را به ترتیب و بر اساس اولویت تعیین شده به چاپگر ارسال می‌کند.

نکته اول) سیستم عامل اولین برنامه‌ای است که بعد از بوت شدن در حافظه بارگذاری می‌شود، پس از بارگذاری قسمتی از سیستم عامل به طور دائم در حافظه باقی می‌ماند. قسمت‌های دیگر با توجه به کاربرد کامپیوتر توسط کاربر از دیسک (HDD) به حافظه (RAM) آورده می‌شود.
نکته دوم) قسمت اصلی سیستم عامل که معمولا در تمامی اوقات بر روی کامپیوتر در حال اجرا است، هسته (Kernel) گفته می‌شود.

۱-۲. نسل‌های کامپیوتر

۱-۲-۱. نسل اول (۵۵-۱۹۴۵)

از لامپ خلاء برای ساخت کامپیوتر استفاده می‌شد. زبان برنامه‌نویسی وجود نداشت، سیستم عاملی هم وجود نداشت. کامپیوترها بسیار گران بودند و برنامه نویسان مجبور بودند برنامه‌های خود را به زبان ماشین به کامپیوتر تحویل دهند. از این کامپیوترها معمولا برای محاسبات عددی ساده استفاده میشد.

۱-۲-۲. نسل دوم (۶۵-۱۹۵۵)

این کامپیوترها از ترانزیستور ساخته شده بودند، طریقه کار با این سیستم‌ها از طریق یک کنسول بود که اپراتور مخصوص به خود را داشت و کاربران مستقیماً با این کامپیوترها محاوره‌ای نداشتند. کاربر برنامه خود را به زبان اسمبلی بر روی کاغذ می‌نوشت سپس آن را با دستگاه بر روی کارت‌هایی پانچ می‌کرد، این کارت به اپراتور تحویل داده می‌شد و اپراتور خروجی آن را بعد از مدتی به کاربر تحویل می‌داد. سیستم تنها می‌توانست یک برنامه را اجرا کند که برای رفع این مشکل سیستم‌های دسته‌ای (Batch Systems) مطرح شدند که این سیستم‌ها شامل دو کامپیوتر برای خواندن و پردازش بودند، کامپیوتری که برای خواندن استفاده می‌شد ارزان قیمت بود. به این ترتیب عملیات خواندن می‌توانست انجام شود و عملیات پردازش در زمان دیگری صورت می‌گرفت.

معایب این روش عبارتند از: گردش طولانی کار و عدم وجود اولویت و نیاز به سخت افزار اضافه.
مزایای این روش عبارتند از: راندمان بهتر و عملیات ساده‌تر و قابلیت استفاده از راه دور.

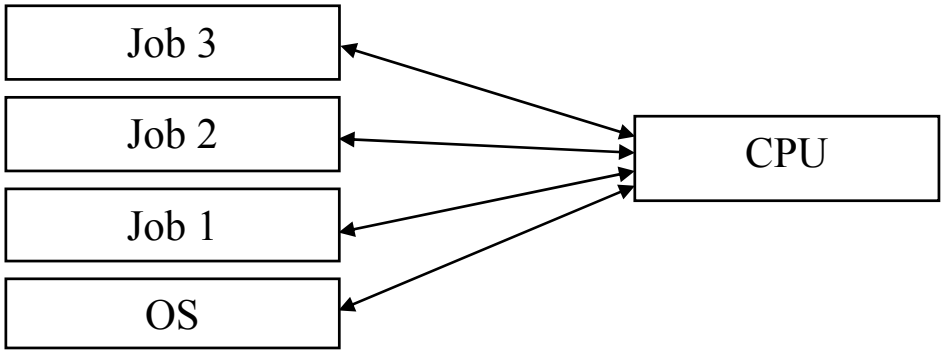
۱-۲-۳. نسل سوم (۸۰-۱۹۶۵)

از مدارات مجتمع یا IC برای ساخت کامپیوترها استفاده شده بود. در این نسل برنامه‌ها در دو دسته کلی بودند:

۱- CPU Limited: مانند محاسبات علمی سنگین که بیشتر زمان کامپیوتر صرف محاسبات CPU می‌شود.

۲- I/O Limited: مانند برنامه‌های تجاری که بیشتر زمان در آن صرف ورود داده‌ها و خروج اطلاعات میشود.

اشکال مهم سیستم‌های دسته‌ای بیکار ماندن CPU هنگام نیاز به عملیات I/O بود. برای رفع این مشکل Multi Programming مطرح شد به این ترتیب که حافظه به چند قسمت تقسیم می‌شد و در هر قسمت یک برنامه مجزا قرار می‌گرفت.



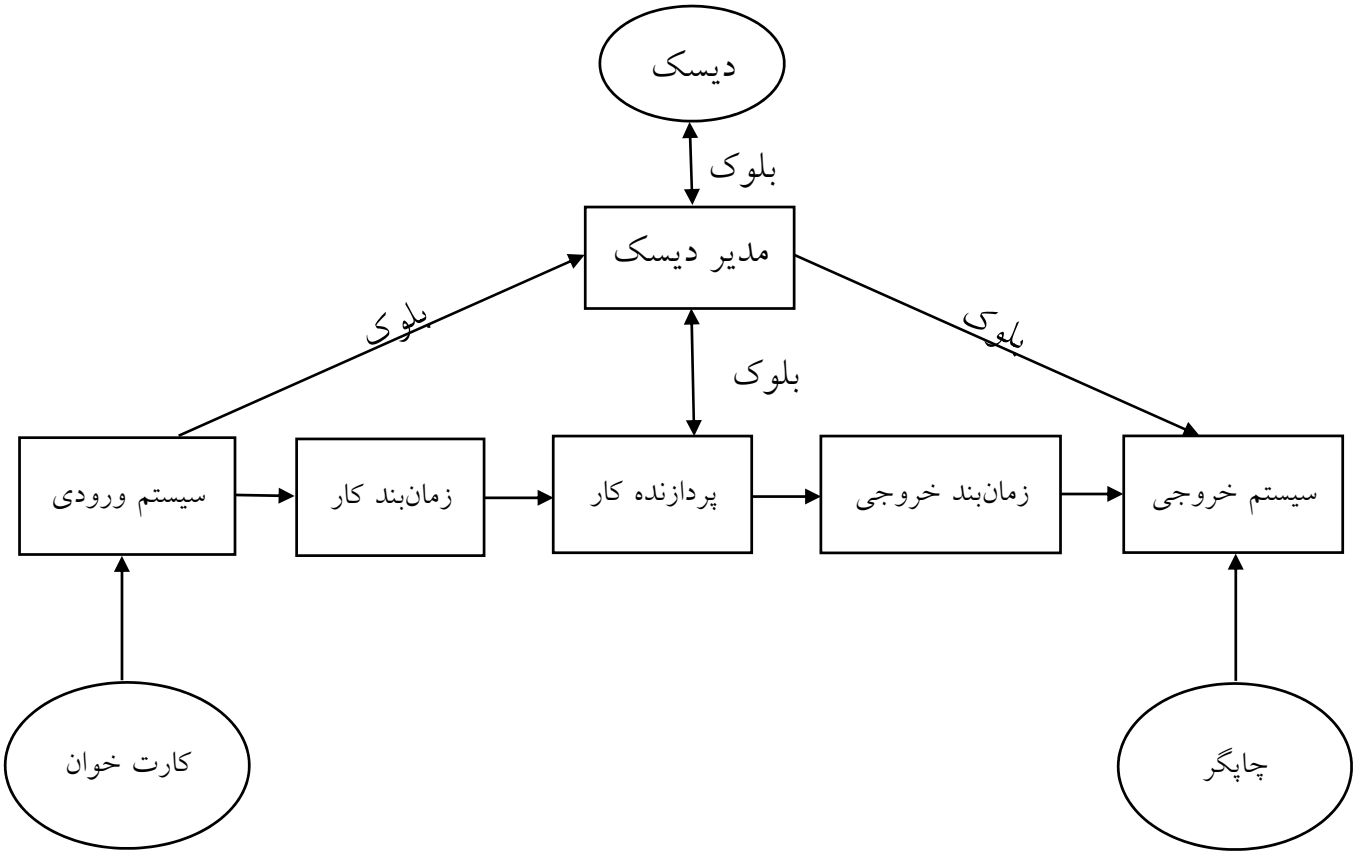
وقتی که یک کار برای تکمیل عملیات I/O منتظر می‌ماند، پردازنده به کار دیگری مشغول می‌شد.

نکته اول) اگر تعداد کارهای موجود در حافظه کافی باشد، می‌توان CPU را ۱۰۰٪ مشغول نگه داشت.

نکته دوم) نگه‌داری همزمان چند برنامه نیاز به مدیریت حافظه دارد.

نکته سوم) در این سیستم‌ها فقط یک CPU وجود دارد که به کمک مکانیزم وقفه بین کارهای CPU Limited و I/O Limited سوییچ می‌کند.

۱-۲-۳-۱. سیستم آنلاین اسپولینگ



در این سیستم، برنامه‌ها پس از خوانده شدن به صورت بلوک‌هایی بر روی دیسک ذخیره می‌شوند.

در این سیستم‌ها نسبت به سیستم‌های دسته‌ای مزایایی وجود دارد که عبارتند از: راندمان بالای CPU و حتی I/O و گردش سریع کار و دسترسی با اولویت، همچنین می‌توان همزمان چند ورودی و خروجی داشته باشیم.

۱-۲-۳-۱-۱. سیستم ورودی

ورودی‌های خوانده شده را در بلوک‌هایی جمع‌آوری می‌کند و به کمک مدیر دیسک در داخل دیسک ذخیره می‌کند.

در انتهای هر مدرک ورودی اطلاعات راجع به آن مدرک مانند مدل ذخیره‌سازی آن در دیسک، اولویت، اسم برنامه استفاده کننده و ... به قسمت زمان‌بند کار فرستاده می‌شود.

۱-۲-۳-۱-۲. زمان‌بند کار (Job Scheduler)

این زمان‌بند یک لیست از کارهای موجود در سیستم و اطلاعات لازم در مورد مدرک ورودی مورد نیاز را نگه می‌دارد. به این لیست Job List یا انبار کار نیز گفته می‌شود. زمان‌بند کار مشخص می‌کند که پردازنده کار، کدام کار را در مرحله بعدی اجرا کند.

۱-۲-۳-۱-۳. پردازنده کار (Job Processor)

کار داده شده را اجرا می‌کند و از محل کامپایلرها و سایر نرم‌افزارهای سیستم بر روی دیسک اطلاع دارد. هنگام اجرا خروجی‌ها را به صورت بلوک‌هایی بر روی دیسک می‌نویسد و مدارک خروجی را تشکیل می‌دهد و همچنین اطلاعاتی راجع به محل و اولویت مدارک خروجی به زمان‌بند خروجی تحویل می‌دهد.

۱-۲-۳-۱-۴. زمان‌بند خروجی

لیستی از مدارکی که باید چاپ شوند را نگه می‌دارد، هنگامی که چاپگر آزاد شد مدرک بعدی را برای چاپ انتخاب کرده و محل مدرک را به سیستم خروجی می‌دهد.

۱-۲-۳-۱-۵. سیستم خروجی

بلوک‌های خروجی را از دیسک خوانده و چاپ می‌کند.

۱-۲-۳-۱-۶. مدیر دیسک

مدیر دیسک وظیفه خواندن و نوشتن یک بلوک بر روی دیسک، تخصیص یک بلوک خالی و یا برگرداندن یک بلوک به مجموعه فضای آزاد دیسک را بر عهده دارد.

درخواست‌های مربوط به دیسک در صفی به نام (Disk Transfer Queue) یا DTQ ذخیره می‌شوند.

۱-۲-۳-۲. اشتراک زمانی (Time Sharing)

اوایل سال‌های ۱۹۷۰ و در نسل سوم کامپیوترها به وجود آمدند. این سیستم‌ها در واقع تامین چندبرنامه‌ای هستند. در سیستم‌های چندبرنامه‌ای کاربر ارتباطی با کامپیوتر نداشت و خطایابی برنامه مشکل بود.

در سیستم اشتراک زمانی کاربر به کمک ترمینال که شامل کیبورد و مانیتور است با کامپیوتر به صورت محاوره‌ای ارتباط برقرار می‌کرد.

۱-۲-۴. سیستم عامل‌های کامپیوترهای شخصی و شبکه

سال ۱۹۸۰ تاکنون که مدارات مجتمع با مقیاس بزرگ ابداع شدند، به عنوان نسل چهارم کامپیوترها شناخته می‌شوند. در این سال‌ها کامپیوترهای شخصی با قیمتی ارزان و کارایی بالا و محیط گرافیکی و محاوره‌ای گسترش یافتند. با توجه به هزینه اندک سخت افزار، اهداف سیستم عامل در طول زمان تغییر کرد و به جای ماکسیمم کردن درصد استفاده CPU، سیستم به سمت راحتی کاربر پیش رفت.
هنگامی که کامپیوترها از طریق شبکه به هم وصل شوند، به آنها ایستگاه کاری یا Work Station گفته می‌شود. در یک سیستم شبکه، کاربران از وجود ماشین‌های مختلف در شبکه باخبرند. آنها می‌توانند از راه دور به این ماشین وارد شوند یا فایل‌ها را از یک ماشین روی یک ماشین دیگر کپی کنند. در این نوع سیستم‌ها هر کامپیوتر سیستم عامل مربوط به خودش را اجرا می‌کند.

۱-۲-۴-۱. سیستم‌های توزیع شده (Distributed Systems)

سیستم عامل توزیع شده در یک محیط شبکه‌ای اجرا می‌شود. در این سیستم، قسمت‌های مختلف برنامه کاربر بدون این که خود او متوجه شود، می‌تواند همزمان در چند کامپیوتر مجزا اجرا شده و نتیجه نهایی به کامپیوتر اصلی برگردد. کاربر از این کار مطلع نیست و این کار توسط سیستم عامل به صورت خودکار انجام می‌شود. به عبارت دیگر سیستم از دید کاربر شفاف است.

مزایا: ۱- اشتراک منابع ۲- تسریع محاسبات ۳- قابلیت اعتماد ۴- ایجاد ارتباطات مثل انتقال فایل.

۱-۲-۴-۲. نخ (Thread) و سیستم‌های Multi Tasking

در تکنیک چندنخی یک فرآیند که برنامه‌ای در حال اجرا است، می‌تواند به بخش‌ها یا نخ‌هایی تقسیم شود که این نخ‌ها می‌توانند همزمان اجرا شوند، برنامه‌هایی که چند وظیفه مستقل را انجام می‌دهند می‌توانند به صورت چندنخی اجرا شوند.

نکته) فرآیند یک برنامه‌ی در حال اجراست که می‌تواند شامل چند نخ نیز باشد که کلیه اطلاعات مربوط به آن در جدول PCB (Process Control Block) ذخیره می‌شوند.

نکات مربوط به اطلاعات موجود در PCB:

۱- حالت جاری پردازش ۲- شماره شناسایی پردازش ۳- اولویت پردازش ۴- نشانی حافظه پردازش ۵- نشانی محل برنامه پردازش بر روی دیسک و نشانی سایر منابع.

۱-۲-۴-۳. سیستم‌های چند پردازنده‌ای (Multi Processor)

این کامپیوترها می‌توانند به جای یک CPU چندین CPU داشته باشند. همچنین سیستم عامل مربوط به آن باید بتواند چندین برنامه را به صورت موازی واقعی اجرا کند. در Multi Tasking چند نخ بر روی CPU اجرا می‌شود و در Multi Processor چند CPU اجرا می‌شود.

مزایای سیستم‌های Multi Processing

۱- زیاد شدن توان عملیاتی (Throughput): منظور تعداد کارهایی است که در یک واحد زمانی انجام می‌شود. بدیهی است هر چه تعداد پردازنده بیشتر باشد، تعداد کارهای تمام شده نیز بیشتر می‌شود. نکته) اگر تعداد پردازنده‌ها n باشد، سرعت اجرا n برابر نمی‌شود چون بخشی از وقت CPU صرف مسائل کنترلی و یا سوییچ کردن به هدر می‌رود.

۲- صرفه جویی در هزینه‌ها: به علت استفاده اشتراکی از منابعی مثل دیسک، حافظه، منبع تغذیه و غیره.

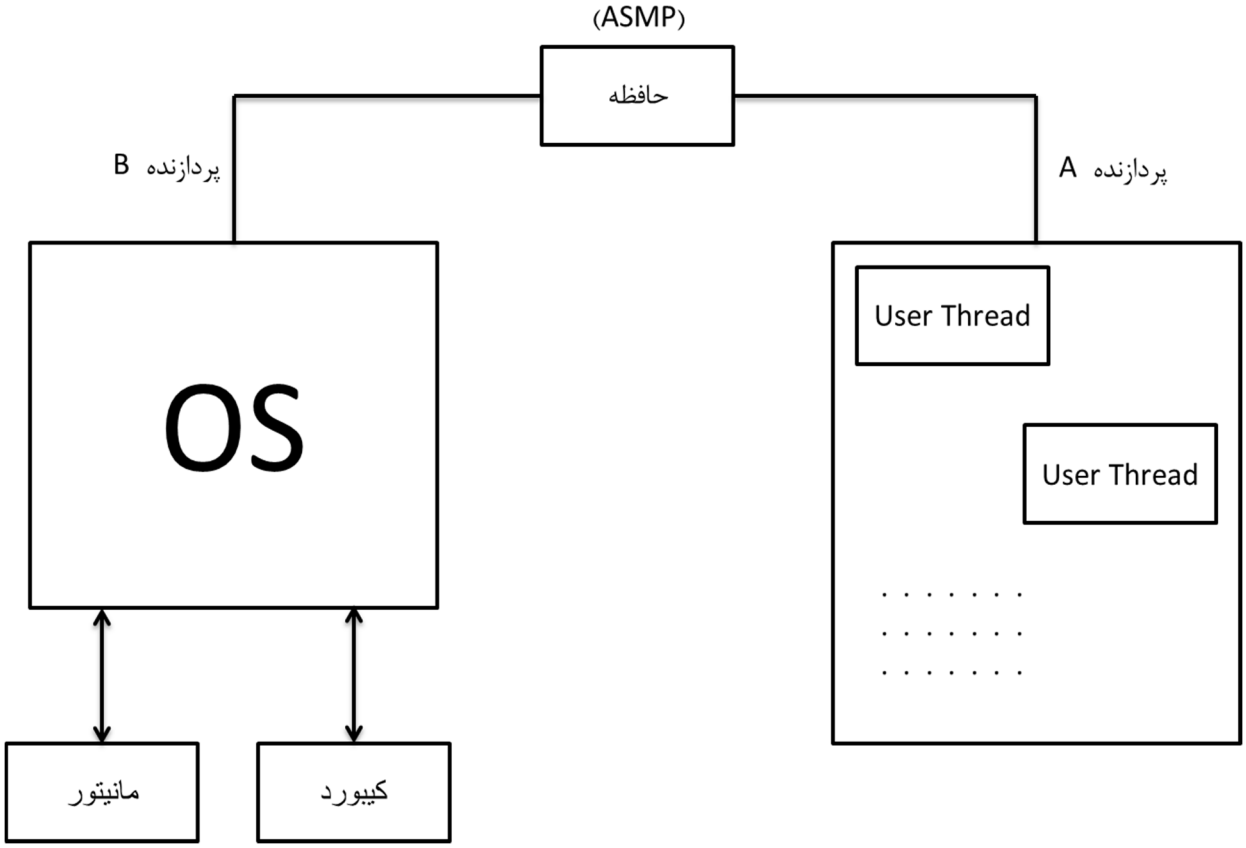
۳- تحمل‌پذیری در برابر خطا (Fault Tolerant): خرابی یک CPU باعث از کار افتادن سیستم نمی‌شود و فقط سیستم کند می‌شود.

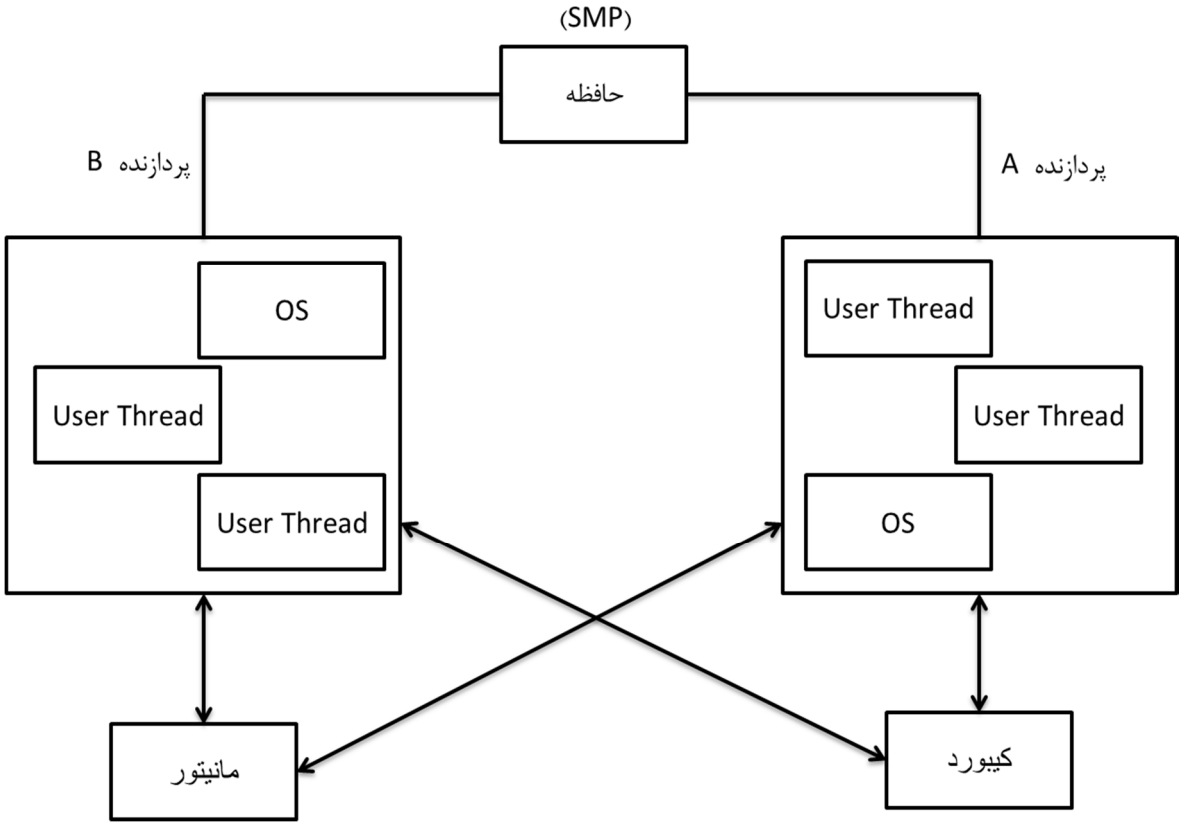
۱-۲-۴-۳-۱. دسته‌بندی سیستم‌های چندپردازنده‌ای

به دو دسته متقارن (SMP) و نامتقارن (ASMP) تقسیم می‌شوند.

سیستم‌های چندپردازنده‌ای نامتقارن (ASMP): یک پردازنده جهت اجرای سیستم عامل و پردازنده‌های دیگر جهت اجرای برنامه‌های کاربران استفاده می‌شود. چون خود سیستم عامل روی یک پردازنده اجرا می‌شود، ساخت این نوع از سیستم عامل‌ها ساده است. این سیستم‌ها قابل حمل نیستند.

سیستم‌های چندپردازنده‌ای متقارن (SMP): سیستم عامل می‌تواند روی هر یک از CPUها اجرا شود و در این حالت نیز حافظه به صورت مشترک استفاده می‌شود.





سیستم‌های متقارن از چند جنبه نسبت به نامتقارن برتری دارند:

۱- از آنجا که سیستم عامل یک پردازش سنگین است، اگر فقط روی یک CPU اجرا شود باعث می‌شود که پردازنده همواره بار سنگینی داشته باشد در حالی که احتمالا CPUهای دیگر بیکار هستند. پس اجرای سیستم عامل بر روی چند پردازنده باعث متعادل شدن بار سیستم یا Balancing می‌شود.

۲- در نامتقارن اگر پردازنده اجرا کننده سیستم عامل خراب شود، کل سیستم خراب می‌شود اما در حالت نامتقارن اینطور نیست که این باعث امنیت بیشتری می‌شود.

۳- سیستم‌های متقارن قابل حمل بر روی سیستم‌های سخت افزاری مختلف هستند.

۱-۲-۴-۴. سیستم‌های بلادرنگ (Real Time OS)

معمولا به عنوان کنترل کننده در یک کاربرد خاص استفاده می‌شود. سیستم در این حالت می‌بایست در زمان مشخص حتما جواب مورد نظر را بدهد. این نوع از سیستم‌ها در صنعت پزشکی، موشکی و سیستم-های کنترل صنعتی استفاده می‌شوند.

فصـــل دوم

ساختار سیستم عامل ها

۲-۱. بخش‌های تشکیل دهنده سیستم عامل‌ها

به طور کلی سیستم عامل‌ها از بخش‌های زیر تشکیل شده‌اند:

۲-۱-۱. مدیریت پردازش و پردازنده

یک برنامه حاوی دستورالعمل‌هایی است که توسط CPU اجرا می‌شوند همچنین حاوی داده‌هایی است که هنگام اجرای دستورات از آن‌ها استفاده می‌شود. برنامه یک موجودیت Passive است مانند محتوای یک فایل برروی دیسک ولی پردازش یک برنامه در حال اجراست وموجودیتی Active دارد. یک پردازش منابع مشخصی را نیاز دارد مثل CPU یا حافظه و ... . پردازش واحد کار سیستم است.

سیستم عامل در رابطه با مدیریت پردازش وظایف زیر را بر عهده دارد:

الف) ایجاد و حذف پردازش‌های کاربر و سیستم

ب) زمان‌بندی پردازش‌ها یعنی اینکه در هر زمان کدام پردازش، CPU را در اختیار داشته باشد.

ج) مدیریت همزمان پردازش‌ها و ارتباط بین آن‌ها و جلوگیری از بن بست (Dead Lock).

نکته) بن‌بست حالتی است که پردازش‌ها در یک حلقه‌ی بسته منتظر اتمام وظایف همدیگر هستند. بنابراین هیچ کدام نمی‌توانند اجرا شوند و سیستم قفل می‌شود.

۲-۱-۲. مدیریت حافظه اصلی و ثانویه

CPU به صورت مستقیم با حافظه‌ی اصلی سروکار دارد و برنامه‌ها جهت اجرا می‌بایست در حافظه اصلی قرار گیرند. سیستم عامل مشخص می‌کند که هر بخش از حافظه توسط چه پردازشی استفاده شود. تخصیص و بازپس گیری فضای حافظه و محافظت از تداخل فرآیندها از وظایف سیستم عامل است.

۲-۱-۳. مدیریت فایل

جهت استفاده ساده از اطلاعات کامپیوتر، سیستم عامل دید منطقی یکسانی از اطلاعات ذخیره شده روی انواع وسایل ذخیره سازی مثل هارد، دیسک نوری، فلش مموری و ... پدید می‌آورد. سیستم عامل خواص فیزیکی وسایل را از دید کاربر پنهان می‌سازد و یک واحد ذخیره‌ی منطقی به نام فایل ارائه می‌کند.

سیستم عامل در رابطه با فایل وظایف زیر را انجام می‌دهد:
۱- ایجاد و حذف فایل ۲- ایجاد و حذف دایرکتوری ۳- عملیات کپی ۴- انتقال و تغییرات بر روی فایل و دایرکتوری ۵- ذخیره سازی و مدیریت قرارگیری فایل برروی وسایل ذخیره‌سازی ۶- مدیریت دسترسی-های مختلف به فایل‌ها

۲-۱-۴. مدیریت وسایل I/O

هر دستگاه ورودی یا خروجی برای کار کردن به مجموعه‌ای از دستورالعمل‌ها نیاز دارد. سیستم عامل به این جزئیات می‌پردازد و برنامه نویس را از دانستن و پرداختن به این ارتباطات سطح پایین بی‌نیاز می‌کند. مثلاً اجرای درایورهای وسایل مختلف و یا جلوگیری از تداخل وسایل مختلف I/O و تخصیص وسایل I/O به پردازش‌ها.

۲-۲. وظایف فرعی سیستم عامل

علاوه بر وظایف اصلی سیستم عامل، وظایف فرعی نیز دارد که این وظیفه تشخیص خطا می‌باشد. این خطاها می‌توانند در CPU و حافظه (RAM) باشد، خطایی که در وسایل I/O وجود دارد مانند کاغذ نداشتن پرینتر (تمام شدن کاغذ چاپگر یا نویزی که روی کابل پرینتر می‌افتد) و خطایی که در برنامه‌ی کاربر باشد مانند خطای سرریزی (Overflow) یا دسترسی به پورت‌ها به صورت غیر مجاز.

۲-۳. مفسر فرمان (Command Interpreter)

مفسر فرمان واسطی بین کاربر و سیستم عامل است. مفسر دستورات کاربر را گرفته و بر روی هسته اجرا می‌کند. فرمان‌ها می‌توانند از طریق صفحه کلید وارد شوند و روی صفحه نمایش به صورت متنی دریافت شوند. همچنین ورود و خروج فرمان‌ها می‌تواند به صورت گرافیکی نیز باشد.

۲-۴. تشخیص اتمام انتقال داده‌ها

دو روش وجود دارد.

۲-۴-۱. روش سرکشی یا نمونه برداری یا Polling

در این روش هر دستگاه دارای ثبات کنترلی است که CPU با بررسی آن متوجه می‌شود که آیا عملیات انتقال آن دستگاه تمام شده یا خیر. لذا در این تکنیک CPU می‌بایست در بازه‌های زمانی خاص مرتباً ثبات‌های کنترلی دستگاه‌های مختلف را به صورت سرکشی بررسی کند. این تکنیک ساده است ولی سرعت آن پایین است.

۲-۴-۲. روش وقفه یا Interrupt

در این تکنیک هر دستگاه دارای سیگنال کنترلی مخصوص به خودش است که هرگاه انتقال داده توسط آن دستگاه تمام شود، سیگنال را که وقفه نام دارد به CPU می‌فرستد و آن را مطلع می‌سازد. سرعت این روش نسبت به قبلی بیشتر است.

وقفه‌ها جزء مهمی از معماری کامپیوتر هستند و عملکرد آن ها از ماشینی به ماشین دیگر ممکن است متفاوت باشد. وقفه‌ها راهکاری را فراهم می‌کنند تا اجرای دستورالعمل‌های جاری پردازنده موقتا متوقف شود و دستورات سرویس‌دهی دیگری اجرا شود و پس از آن کنترل دوباره به برنامه‌ی وقفه داده شده بازگردد.

۲-۵. انواع وقفه‌ها

وقفه ها به دسته های زیر تقسیم بندی می شوند.

۲-۵-۱. وقفه‌ی برنامه (Program Check)

به دلیل اجرای بعضی از دستورات رخ می‌دهد مانند سرریز شدن محاسباتی یا تقسیم بر صفر بودن یا اجرای دستورالعمل غیرمجاز و یا رجوع به آدرس خارج از محدوده‌ی مجاز کاربر، به این وقفه‌ها Trap یا تله نیز گفته می‌شود.

۲-۵-۲. وقفه‌های زمان‌سنج (Timer)

این وقفه به سیستم عامل این امکان را می‌دهد که بعضی از اعمال را به صورت مرتب در بازه‌های زمانی خاص انجام دهد مانند چک کردن سخت افزار و تنظیم ساعت.

۲-۵-۳. وقفه I/O

این وقفه‌ها به وسیله کنترل کننده‌های دستگاه I/O تولید می‌شوند تا کامل شدن طبیعی یک عمل یا بروز خطا در انجام عمل را نشان دهند.

۲-۵-۴. وقفه نقص سخت افزار (Hardware Defect)

وقفه‌ای است که ممکن است در سخت افزارها رخ دهد به عنوان مثال وقفه‌ای که در اثر نقص برق ممکن است در یک دستگاه رخ دهد.

۲-۵-۵. وقفه SVC (Superviser Call)

در واقع یک تقاضا از طرف کاربر یا برنامه‌ی کاربر جهت دریافت سرویس ویژه‌ای از طرف سیستم عامل است.

۲-۵-۶. وقفه Restart

با فشردن دکمه‌ی Reset اجرا می‌شود.

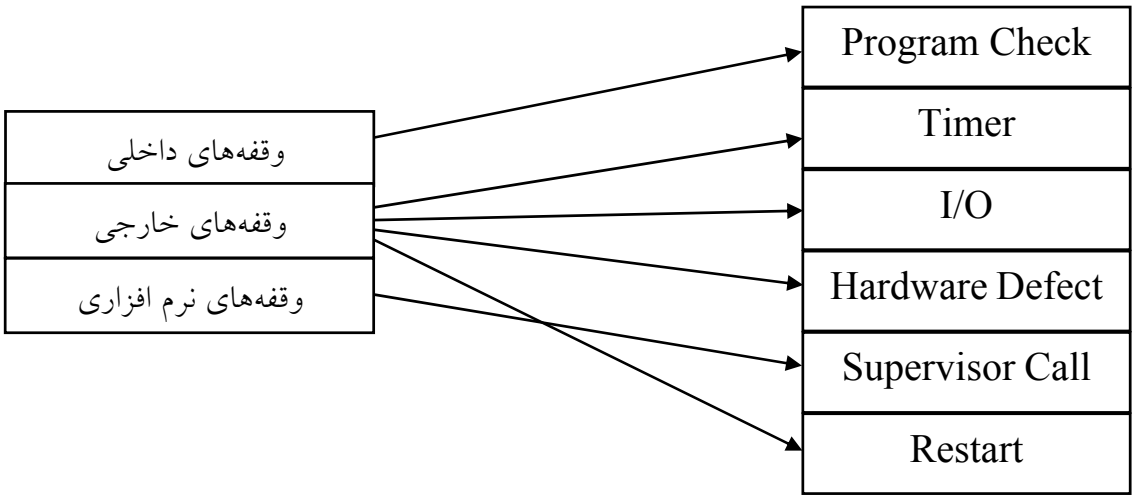
۲-۶. تقسیم بندی وقفه‌ها به صورت کلی‌تر

در یک تقسیم بندی کلی‌تر وقفه‌ها به سه دسته زیر تقسیم می‌شوند:

الف) وقفه‌های داخلی (Trap) که بر اثر دستورات خود برنامه به صورت داخلی در CPU رخ می‌دهد.

ب) وقفه‌های خارجی از دستگاه‌های خارجی مثل I/O یا دسترسی مستقیم به حافظه یا خطاهای سخت افزاری یا تایمرها رخ می‌دهد.

ج) وقفه‌های SVC (نرم افزاری) که بر اثر فراخوانی توابع سیستمی توسط برنامه و کاربر رخ می‌دهد.



نکته اول) با اجرای برنامه وقفه‌های داخلی به همان صورت قبلی رخ می‌دهند یعنی برنامه متوقف شده و پس از وقفه ادامه‌ی برنامه انجام می‌شود ولی وقفه‌های خارجی مستقل از دستورات برنامه و ممکن است ناهمگام با برنامه باشند.

نکته دوم) اگر چند منبع همزمان تقاضاهایشان را از طریق وقفه به CPU اعلام کنند در این حالت CPU با روش سرکشی (Polling) منبع وقفه دهنده را تشخیص می‌دهد.

۲-۷. مراحل عملیات I/O به کمک وقفه‌ها

مرحله اول: CPU دستگاه جانبی را مقداردهی می‌کند (مثلاً تعداد بایت‌هایی که باید خوانده شود) و سپس تقاضای شروع عمل I/O را به دستگاه می‌فرستد.

مرحله دوم: پردازنده به سراغ پردازش دیگری می‌رود و دستگاه جانبی نیز همان کار خود را آغاز می‌کند.

مرحله سوم: دستگاه I/O پس از اتمام کار با فرستادن سیگنال وقفه این موضوع را به CPU اعلام می‌کند.

مرحله چهارم: پردازنده در اولین فرصت به وقفه‌ی صادر شده سرویس می‌دهد.

۲-۸. تعویض متن یا Context Switch

قبل از آن که سیستم عامل کنترل را به یک روال وقفه گیر بفرستد، وضعیت پردازش جاری را در محلی حفظ می‌کند تا بعداً بتواند آن را ادامه دهد. به این عملیات تعویض متن یا (Context Switch) گفته می‌شود. برای عملیات تعویض متن از ثبات PSW (Program Status Word) که ترتیب اجرای دستورات را کنترل می‌کند، استفاده می‌شود.

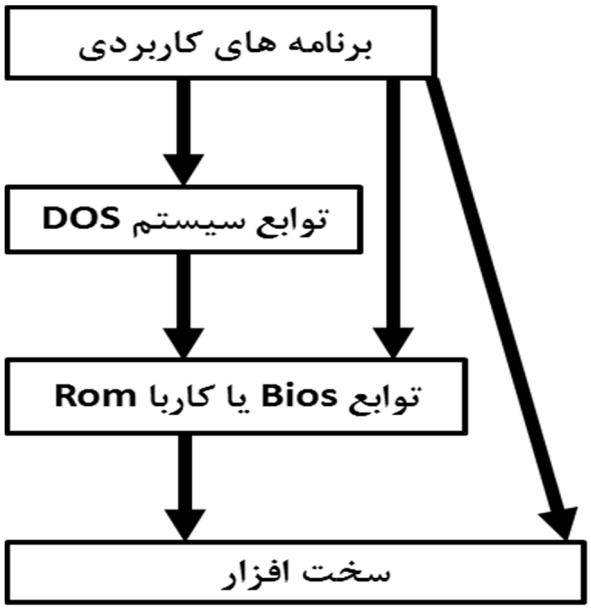
۲-۹. انواع سیستم عامل‌ها از نظر ساختاری

چهار ساختاری که برای ایجاد سیستم عامل‌ها که در عمل آزمایش شده‌اند عبارتند از:

۲-۹-۱. سیستم‌های یک پارچه (Monolithic)

سیستم عامل به صورت یک مجموعه از رویه‌ها نوشته شده است که هر یک از آن‌ها می‌توانند دیگری را درهنگام نیاز فراخوانی کنند. برای مخفی کردن اطلاعات امکاناتی وجود ندارد و هر رویه برای دیگر رویه‌ها کاملاً مشخص یا قابل مشاهده است.

مثال سیستم عامل DOS :



۲-۹-۲. سیستم‌های لایه‌ای

در روش لایه‌ای سیستم عامل به تعدادی سطح یا لایه تقسیم می‌شود که هر کدام در بالای لایه‌ی پایین‌تر قرار می‌گیرد مزیت این روش ماژولاریتی (Modularity) یا پیمانه‌ای است یعنی هر لایه به گونه‌ای تقسیم بندی می‌شود که فقط توابع و سرویس‌های لایه‌ی پایین‌تر را استفاده کند. به این ترتیب طراحی لایه‌ها مستقل هستند و خطایابی و گسترش لایه‌ها می‌تواند انجام شود.

۲-۹-۳. ماشین مجازی

در این نوع سیستم عامل قلب سیستم که به صورت مانیتور ماشین مجازی است بر روی سخت‌افزاری اجرا می‌شود و خاصیت چند برنامگی پدید می‌آورد. به این ترتیب به هر کاربر ماشین مجازی خودش داده می‌شود و او می‌تواند سیستم عامل یا بسته‌ی نرم افزاری موجود بر روی ماشین خودش را اجرا کند.

<table>
<tr><td>ناحیه کاربری ۱</td><td>ناحیه کاربری ۲</td><td>ناحیه کاربری ۳</td></tr>
<tr><td colspan="3">مانیتور ماشین مجازی</td></tr>
<tr><td colspan="3">سخت افزاری</td></tr>
</table>

۲-۹-۴. مشتری/خدمت‌گذار Client/Server

روند طراحی سیستم عامل‌ها همواره با این ایده همراه بود که تا جایی که ممکن است کدها به لایه‌های بالاتر منتقل شوند تا نهایتاً یک هسته‌ی کمینه پدید آید. برای این منظور اکثر وظایف سیستم عامل را در سطح کاربر پیاده‌سازی می‌کنیم. در این مدل تنها کاری که هستند انجام می‌دهد این است که ارتباط میان مشتری‌ها وخدمت‌گذارها را از طریق پیام برقرار می‌سازد.

فصـــل سوم

پردازش و زمانبندی

۳-۱. زمانبند کار (Job Scheduler)

زمان‌بند کار روتینی است که بر اساس الگوریتم خاص یکی از کارهای موجود در جدول مربوط به کارها را انتخاب کرده و جهت اجرا شدن آن را به حافظه می‌آورد. هنگامی که کارها در جدول کار منتظر اجرا شدن هستند اصطلاحاً Hold شده‌اند و هنگامی که توسط زمان‌بند کار جهت اجرا انتخاب می‌شوند اصطلاحاً به آنها حالت Ready گفته می‌شود.

نکته) الگوریتم‌های زمان‌بندی کار یا به عبارتی دیگر روش‌های انتخاب Job عبارتند از:

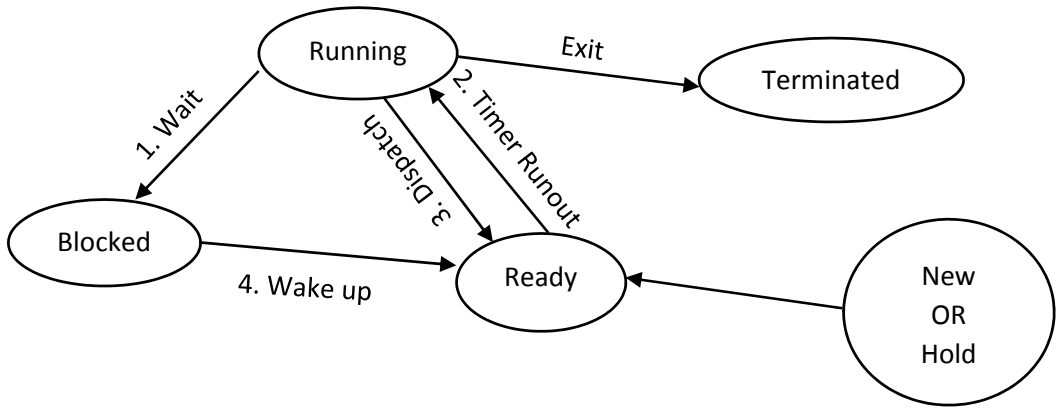
۱- الگوریتم FIFO (First In First Out): در این الگوریتم اولین کاری که وارد جدول مربوط به کارها شده است، اولین کاری است که به پردازش تبدیل می‌شود.

۲- الگوریتم SJF (Shortest Job First): در این الگوریتم کوتاه‌ترین کاری که در جدول کارها وجود دارد، پیدا شده و جهت اجرا انتخاب می‌شود.

۳- روش Mixed: در این روش ترکیبی از کارهای I/O Limited, CPU Limited جهت اجرا انتخاب می‌شوند.

۳-۲. حالات مختلف پردازش

یک پردازش در زمان حیات خود در ۳ وضعیت اصلی اجرا، آماده و مسدود قرار می‌گیرد.

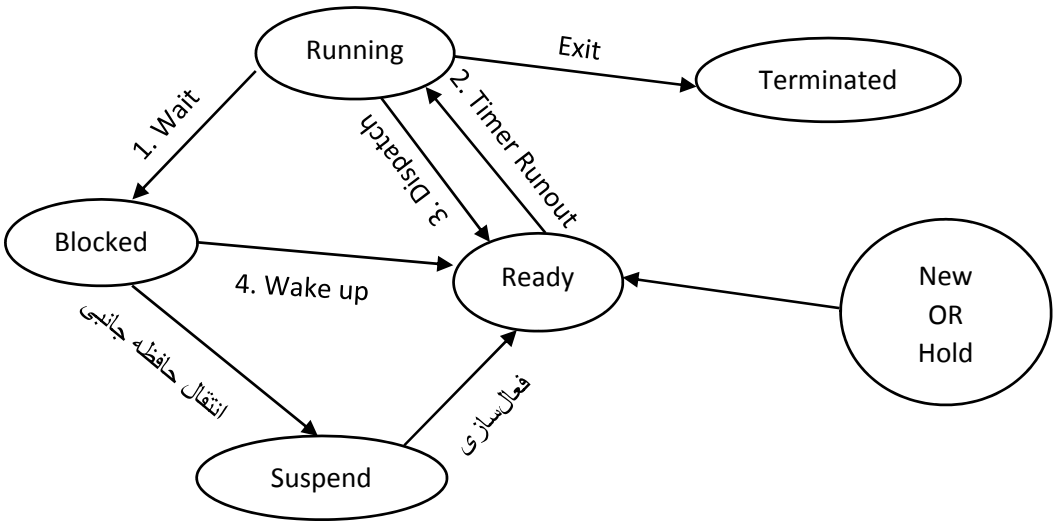


هنگامی که کاری از جدول کارها انتخاب می‌شود به صف آماده وارد شده و منتظر CPU می‌ماند. انتقال شماره‌ی ۲و۳ توسط زمان‌بند پردازش انجام می‌شود. در انتقال ۳، CPU به پردازش داده می‌شود تا اجرا

گردد، انتقال ۲ زمانی است که برش زمانی پردازش تمام شده و CPU باید از آن گرفته شود. پس پردازش از حالت اجرا به حالت آماده برمی‌گردد. در حالت آماده پردازش تمام منابع مورد نیاز را به جزء CPU در اختیار دارد. هنگامی که لازم باشد، پردازشی که در حال اجراست برای بروز رخدادی (مثل تکمیل عملیات I/O) صبر می‌کند. آن گاه از طریق انتقال ۱ از حالت آماده به مسدود می‌رود. پس از رفع علت انتظار پردازش از طریق شماره ۴ به حالت آماده رفته و در صف انتظار برای CPU قرار می‌گیرد.

۳-۲-۱. حالت معلق (Suspend)

بسیاری از سیستم عامل‌ها با سه حالت آماده، اجرا، مسدود، پیاده سازی شده‌اند ولی اضافه شدن حالت معلق نیز در بسیاری از موارد مفید است.



فرآیندهای بلاک شده همچنان در RAM هستند و CPU هنگام بلاک شدن یک فرآیند می‌تواند فرآیندی را از صف آماده اجرا کند اما سرعت CPU بالاست و گاهی تمام فرآیندها منتظر عمل I/O یا در حالت بلاک هستند پس CPU ممکن است بیکار بماند. در این حالت تمام یا بخشی از یک فرآیند از حافظه اصلی به حافظه جانبی منتقل می‌شود. در این زمان CPU برای آوردن فرایند جایگزین ۲ انتخاب دارد:

انتخاب ۱) می‌تواند فرآیندی که به تازگی ایجاد شده را بپذیرد.

انتخاب ۲) فرآیندی که قبلاً معلق شده و دسترسی به منبع برای آن صورت گرفته را از حالت تعلیق خارج نموده و اجرا کند.

نکته ۱) حالت آماده یعنی فرآیند در رم (RAM) بوده و آماده‌ی اجراست.

نکته ۲) حالت بلاک یعنی فرآیند در رم (RAM) است و منتظر رخدادی خاص است.

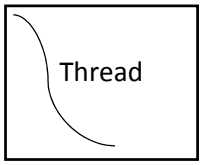
نکته ۳) حالت معلق و مسدود یعنی فرآیند در حافظه‌ی جانبی قرار دارد و منتظر رخدادی خاص است.

نکته ۴) حالت معلق و آماده فرآیند در حافظه جانبی است وآن رخداد خاص صورت گرفته و به محض بار شدن در حافظه اصلی (فعال‌سازی) قابل اجرا خواهد بود.

نکته ۵) انتقال بین این مراحل توسط بخشی از سیستم عامل به نام مدیریت پردازش صورت می‌گیرد.

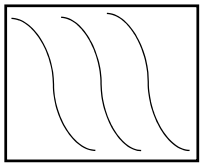
نکته ۶) انتقال یک یا بخشی از فرآیند از حافظه‌ی اصلی به حافظه جانبی و بر عکس آن را Swaping یا مبادله می‌گوییم و عامل Swaping کمبود حافظه‌ی اصلی است.

۳-۳. حالات مختلف بین نخ و فرآیند



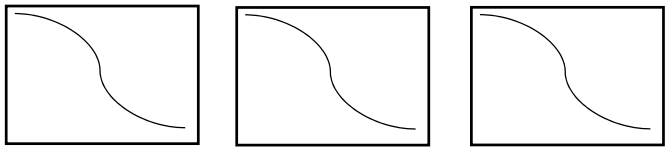
۱- یک فرآیند - یک نخ:

فرآیند همان نخ است!



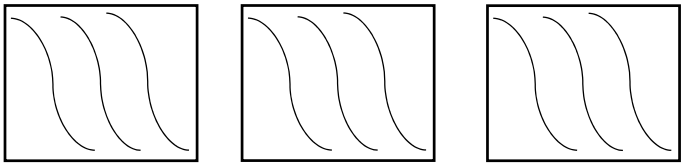
۲- یک فرآیند - چند نخ:

(Multy Threading) چند نخی



۳- چند فرآیند - یک نخ:

Multy Tasking ⟵ Unix



۴- چند فرآیند-چند نخ:

Multy Threaing
Multy Tasking

Windows/Linux

۳-۴. حالات مختلف نخ

۱- حالات Ready و Running برای نخ دقیقا همانند فرآیند است.
۲- در حالت Block اگر یک نخ بلاک شود، تاثیری در بلاک شدن سایر نخ‌ها ندارد و سایر نخ‌ها می‌توانند اجرا شوند.
۳- در حالت Suspend چون در هر خط ناحیه کد و داده بایستی معلق شود و از آنجا که این ناحیه بین نخ ها مشترک است، بنابراین در صورت معلق شدن یک نخ تمام نخ‌های مرتبط با آن نخ نیز معلق می‌شوند.

۳-۵. زمان‌بندی در سیستم عامل

زمان‌بندی به معنای این است که مشخص شود کدام فرآیند زودتر و کدامیک بایستی دیرتر اجرا شود و این زمان‌بندی می‌تواند بر اساس اولویت باشد و در بخشی از سیستم عامل به نام زمان‌بند انجام می‌شود.

نکته) پردازش در حال اجرا بنا به دلایل زیر می‌تواند به صف انتظار برود تا زمان‌بندی شود:
۱- پردازش می‌تواند یک درخواست I/O صادر کند و سپس منتظر بماند تا به آن سرویس داده شود.
۲- پردازش می‌تواند یک پردازش جدید ایجاد نموده و برای اتمام آن صبر کند.
۳- پردازش به علت تمام شدن برش زمانی (Time Slice) از CPU جدا می‌شود تا این امکان به بقیه پردازش‌ها داده شود که از CPU استفاده کنند.
۴- پردازش منتظر وقوع وقفه است.

۳-۶. انواع روش‌های زمان‌بندی

به طور کلی دو روش زمانبندی زیر وجود دارد:

۳-۶-۱. انحصاری (Non-Preemptive)

در این حالت تنها هنگامی CPU از فرآیند گرفته می‌شود که اجرا خاتمه یافته یا فرآیند بلاک شود.

۳-۶-۲. غیرانحصاری (Preemptive)

در این حالت با در نظر گرفتن برش‌های زمانی و خاتمه آن برای یک فرآیند، CPU از آن گرفته می‌شود و در اختیار پردازش‌های دیگر قرار می‌گیرد.

نکته ۱) شرط پایداری سیستم این است که میانگین نرخ ایجاد پردازش تقریبا برابر با میانگین نرخ خاتمه پردازش‌ها باشد. $\text{Mean Creation Rate} \leq \text{Mean Terminate}$

نکته ۲) نسبت برش زمانی به زمان تعویض متن بایستی زیاد باشد تا راندمان پردازنده کاهش پیدا نکند.

مثال) اگر زمان Time Slice برابر با 20 MS و زمان تعویض متن 2 MS باشد، چقدر از زمان CPU هدر می‌رود؟

$$\frac{\text{تعویض متن}}{\text{برش زمانی}} = \frac{2\ MS}{20\ MS} = 10\%$$

۳-۷. معیارهای زمان‌بندی

معیارهای کلی زمانبندی شامل موارد زیراست:

۳-۷-۱. کارایی پردازنده (CPU Performance)

یعنی این که CPU در تمامی زمان‌ها حتی‌الامکان مشغول باشد.

۳-۷-۲. زمان پاسخ (Response Time)

فاصله زمانی تحویل کار تا صدور این پاسخ از سمت پردازنده و یا به حداقل رساندن این زمان برای فرمان‌های محاوره‌ای کاربر.

۳-۷-۳. زمان بازگشت (TurnAround Time)

فاصله زمانی بین تحویل کار تا تکمیل آن.

۳-۷-۴. توان عملیاتی (Throughput)

تعداد پردازش‌هایی که به صورت همزمان اجرا می‌شوند یا تعداد پردازش‌هایی که در واحد زمان تکمیل میشوند.

۳-۷-۵. زمان انتظار (Waiting Time)

مجموع زمان‌هایی که پردازش در صف Ready برای پردازش توسط CPU منتظر می‌ماند.

۳-۷-۶. عدالت (Fairness)

اطمینان از این که هر پردازش سهم عادلانه یا منصفانه‌ای از CPU دریافت کند.

۳-۸. الگوریتم های زمانبندی

در این قسمت الگوریتم های زمانبندی بررسی می شوند.

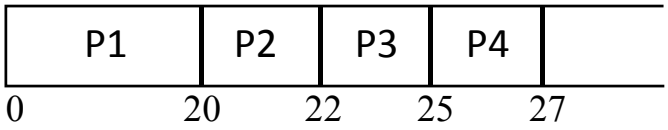
۳-۸-۱. FIFO (FCFO)

این زمان‌بندی انحصاری است. در این زمان‌بندی اولین پردازشی که وارد می‌شود، اولین پردازشی است که به آن CPU تخصیص داده می‌شود و تا زمان خاتمه اجرا CPU را رها نمی‌کند.

مثال) چهار پردازش با زمان اجرای زیر وارد صف می‌شوند. میانگین زمان انتظار برای این چهار پردازش را بدست آورید. (روش FIFO)

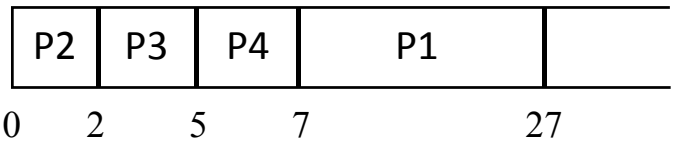
پردازش	P1	P2	P3	P4
Time	20	2	3	2

نمودار گانت



میانگین زمان انتظار $\rightarrow$ زمان انتظار $= \frac{0+20+22+25}{4} = 16.75$

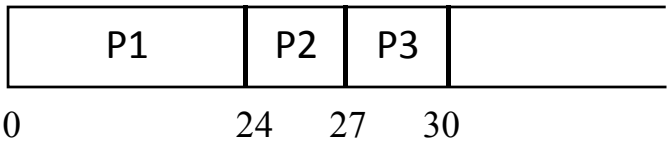
مثال) مثال قبل را در حالتی که ابتدا P2 سپس P3 و سپس P4 و در آخر P1 وارد شوند را محاسبه کنید.



میانگین زمان انتظار $\rightarrow$ زمان انتظار $= \frac{0+2+5+7}{4} = 3.5$

مثال) میانگین زمان انتظار را برای سه پردازش زیر محاسبه کنید. روش FIFO و زمان ورود اولین پردازش است.

پردازش	P1	P2	P3
Time	24	3	3



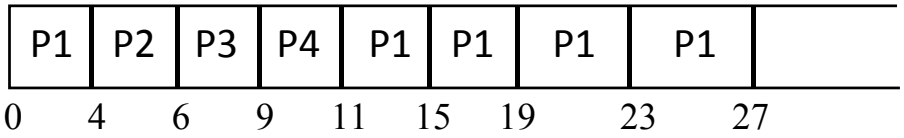
میانگین زمان انتظار $\rightarrow$ زمان انتظار $= \frac{0+24+27}{3} = 17$

۳-۸-۲. RR (Round Robin)

این زمان‌بندی غیرانحصاری است، در این زمان‌بندی با تخصیص کوانتوم زمانی یا برش زمانی، وقت پردازنده به صورت عادلانه بین پردازش‌ها تقسیم می‌شود.

مثال) چهار پردازش زیر با زمان اجرای زیر وارد صف می‌شوند. میانگین زمان انتظار برای این چهار پردازش را بدست آورید. (روش RR و کوانتوم زمانی 4MS)

پردازش	P1	P2	P3	P4
Time	20	2	3	2



میانگین زمان انتظار $\rightarrow$ زمان انتظار $\frac{[0+(11-4)]+4+6+9}{4} = 6.5$

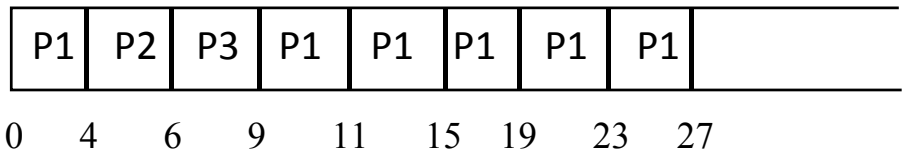
مثال) زمان بازگشت را برای مثال قبل محاسبه کنید.

میانگین زمان بازگشت $\rightarrow$ زمان بازگشت $\frac{27+6+9+11}{4} = 13.25$

مثال) میانگین زمان انتظار و زمان بازگشت و زمان اجرا را برای سه پردازش زیر محاسبه کنید. (روش RR و کوانتوم 4MS)

پردازش	P1	P2	P3
Time	24	3	3

میانگین زمان اجرا $\rightarrow$ زمان اجرا $\frac{24+3+3}{3} = 10$



میانگین زمان انتظار $\rightarrow$ زمان انتظار $\frac{[0+(10-4)]+4+7}{3} = 5.66$

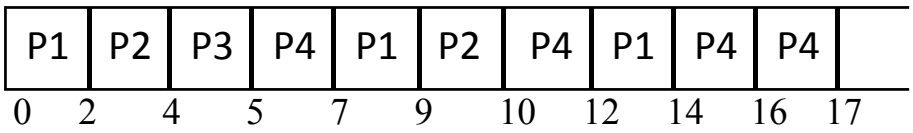
میانگین زمان بازگشت $\rightarrow$ زمان بازگشت $\frac{30+7+10}{3} = 15.66$

نکته) میانگین زمان برگشت برابر است با میانگین زمان اجرا به اضافه میانگین زمان انتظار

میانگین زمان انتظار + میانگین زمان اجرا = میانگین زمان بازگشت

مثال) برای چهار پردازش زیر میانگین زمان انتظار و برگشت را به روش RR با کوانتوم زمانی 2MS بدست آورید.

پردازش	P1	P2	P3	P4
Time	6	3	1	7



میانگین زمان بازگشت $\rightarrow$ زمان بازگشت $\frac{14+10+5+17}{4} = 11.5$

زمان انتظار$\rightarrow$ $\frac{[0+(7-2)+(12-9)]+[2+(9-4)]+4+[5+(10-7)+(14-12)]}{4} = 7.25$

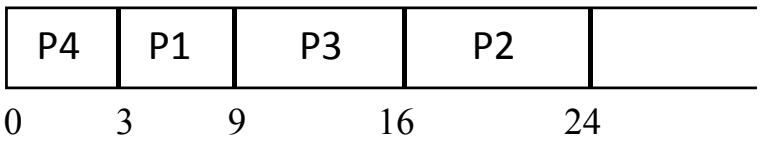
۳-۸-۳. SJF (Shortest Job First)

این زمان‌بندی انحصاری است، در این روش پردازنده به پردازش‌هایی با زمان کمتر تخصیص می‌یابد. مشکل این روش این است که با اولویت پردازش‌های کوچکتر، فرآیندهای طولانی‌تر دچار گرسنگی یا قحطی (Starvation) می‌شوند.

مثال) چهار پردازش زیر را در نظر بگیرید. میانگین زمان انتظار را به روش SJF بدست آورید و آن را به روش FIFO مقایسه کنید.

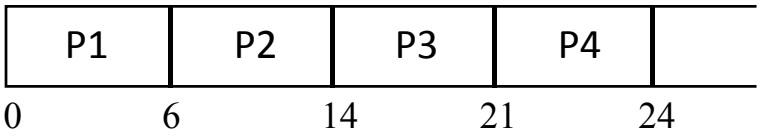
پردازش	P1	P2	P3	P4
Time	6	8	7	3

گانت SJF



میانگین زمان انتظار $\frac{3+16+9+0}{4} = 7$ ← زمان انتظار SJF

گانت FIFO

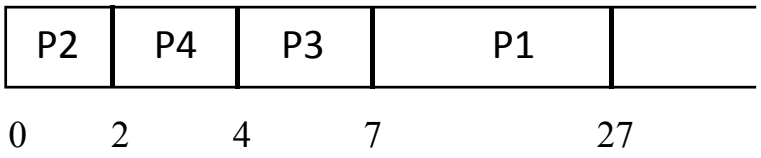


میانگین زمان انتظار $\frac{0+6+14+21}{4} = 10.25$ ← زمان انتظار FIFO

طبق محاسبات، روش SJF بهتر است.

مثال) به روش SJF میانگین زمان انتظار و بازگشت را محاسبه کنید.

پردازش	P1	P2	P3	P4
Time	20	2	3	2



میانگین زمان انتظار $\frac{7+0+4+2}{4} = 3.25$ ← زمان انتظار

میانگین زمان بازگشت $\frac{27+2+7+4}{4} = 10$ ← زمان بازگشت

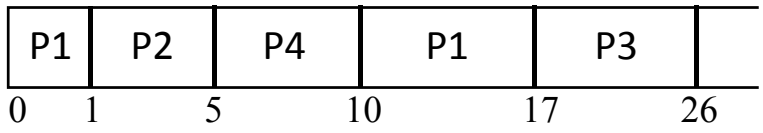
۳-۸-۴. SRT (Shortest Remaining Time)

زمان‌بندی SRT غیرانحصاری است و در آن از صف آماده، فرآیندی انتخاب می‌شود که کمترین زمان باقیمانده برای خاتمه اجرا را داشته باشد. در این حالت سایر فرآیندهای جدیدالوقوع نیز به صف آماده در نظر گرفته می‌شوند.

مثال) چهار فرآیند زیر را در نظر بگیرید. میانگین زمان انتظار را در حالت SRT و SJF محاسبه کنید.

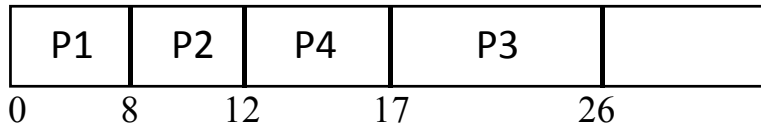
پردازش	P1	P2	P3	P4
Time	8	4	9	5
زمان ورود	0	1	2	3

گانت SRT



میانگین انتظار SRT $6.5 = \frac{[0+(10-1)]+(1-1)+(17-2)+(5-3)}{4}$ ← زمان انتظار SRT

گانت SJF



میانگین زمان SJF $7.75 = \frac{(0-0)+(8-1)+(17-2)+(12-3)}{4}$ ← زمان انتظار SJF

۳-۸-۵. HRRN (Highest Response Ratio Next)

این زمان‌بندی انحصاری است و اولویت با بالاترین نرخ پاسخ می‌باشد. این زمان‌بندی برخی از مشکلات SJF مانند قحطی را تاحدی حل می‌کند. اولویت در این زمان‌بندی داینامیک است و از فرمول زیر محاسبه می‌شود:

$$\text{اولویت} = \frac{\text{زمان انتظار} + \text{زمان سرویس}}{\text{زمان سرویس}}$$

نکته) چون زمان سرویس در مخرج کسر است لذا کارهایی با زمان کوتاه‌تر اولویت بیشتری خواهند یافت و این خوب است.

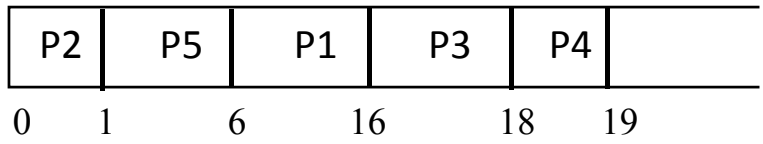
نکته) چون زمان انتظار در صورت کسر است، لذا کارهایی با زمان طولانی‌تر نیز در بازه زمانی اولویت اجرا پیدا می‌کنند.

۳-۸-۶. اولویت (Priority)

این زمان‌بندی هم انحصاری و هم غیرانحصاری است. در این زمان‌بندی اجرای فرآیندها با توجه به اولویت انجام می‌شود. در انحصاری پردازش با اولویت بیشتر جلوی صف Ready قرار می‌گیرد. در غیرانحصاری پردازشی که وارد صف Ready می‌شود اگر اولویت بیشتری داشته باشد، پردازنده را در اختیار می‌گیرد.

مثال)

پردازش	P1	P2	P3	P4	P5
Time	10	1	2	1	5
اولویت	3	1	3	4	2



میانگین زمان انتظار $\rightarrow$ زمان انتظار $= \frac{6+0+16+18+1}{5} = 8.2$

۳-۸-۷. صف‌های چندگانه (Multiple Queue)

در این الگوریتم برای پردازش‌های مختلف، صف‌های جداگانه‌ای تشکیل می‌شود و هر پردازش وارد یک صف می‌شود. روش زمان‌بندی صف‌های مختلف نیز متفاوت است.

۳-۸-۸. صف‌های چندگانه با فیدبک (Multiple Feedback Queue)

در روش قبل اگر پردازشی وارد صف شود، از آن خارج نمی‌شود ولی در زمان‌بندی MFQ پردازش‌ها می‌توانند بین صف‌ها جابه‌جا شوند.

۳-۸-۹. شانسی (Lottery)

این زمان‌بندی انحصاری است. در این الگوریتم به هر پردازش تعدادی عدد داده می‌شود، تعداد اعداد نسبت داده شده به هر پردازش وابسته به اولویت آن است یعنی هر چه اولویت بیشتر باشد اعداد بیشتری

به آن داده می‌شود سپس به طور اتفاقی یک عدد تولید می‌شود و پردازشی که آن عدد را در اختیار دارد، CPU را میگیرد.

۳-۸-۱۰. LPT (Longest Processing Time)

از بین کارهای باقی‌مانده طولانی‌ترین کار را اجرا می‌کند و انحصاری است.

۳-۹. محاسبه توان عملیاتی و راندمان CPU

$$\text{توان عملیاتی} = \frac{\text{تعداد پردازش‌ها}}{\text{زمان کل}}$$

$$\text{راندمان CPU} = \frac{\text{زمان پردازش}}{\text{زمان کل پردازش}}$$

۳-۱۰. تکنیک Aging (سالمندی)

به منظور جلوگیری از قحطی از آن استفاده می‌شود که در آن به تدریج زمان اولویت پردازش‌هایی که منتظر پردازنده بودند، افزایش داده می‌شود. مثلا هر n دقیقه یکبار یک واحد به اولویت پردازش‌های منتظر در صف اضافه می‌شود.

فصـــل چهارم

بن بست (Dead Lock)

۴-۱. مفهوم بن بست

سیستمی را با یک چاپگر و یک سی‌دی رام در نظر بگیرید. فرض کنید پردازش P1 چاپگر را در اختیار گرفته و پردازش P2 سی‌دی رام را. حال اگر P1 سی‌دی رام را و P2 چاپگر را همزمان تقاضا کنند آنگاه بن بست رخ می‌دهد. به عبارت دیگر اگر هر پردازش در انتظار حادثه‌ای است که توسط پردازش دیگری می‌تواند رخ دهد، حالت بن بست اتفاق افتاده است.

۴-۲. شرایط لازم برای ایجاد بن بست (شروط کافمن)

شروط کافمن به شرح زیر است:

۴-۲-۱. انحصار متقابل (بالقوه)

منظور از انحصار متقابل این است که اگر فرآیندهایی به صورت همزمان با هم اجرا شدند، منابع مشترک در هر زمان تنها در اختیار یک فرآیند قرار خواهد گرفت. علاوه بر این اگر تعدادی فرآیند برای تصاحب منبعی با هم رقابت کنند، سیستم عامل تضمین می‌کند که این منبع را تنها در اختیار یکی از آنها قرار دهد.

۴-۲-۲. گرفتن و منتظر ماندن (بالقوه) (Hold & Wait)

باید پردازشی وجود داشته باشد که حداقل یک منبع را گرفته و در حال انتظار برای کسب منابع دیگر باشد.

۴-۲-۳. انحصاری بودن (بالقوه)

منبع را نمی‌توان بالاجبار از پردازش گرفت و پردازش بعد از اتمام کار داوطلبانه آن را رها کند.

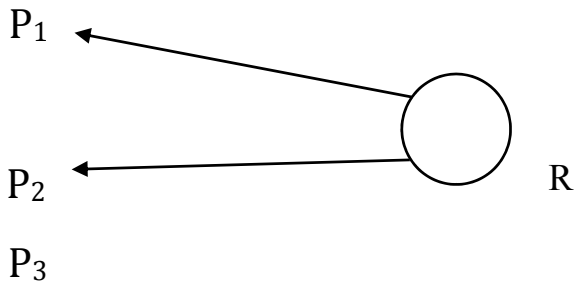
۴-۲-۴. انتظار چرخشی (بالفعل) (Circular Wait)

بایستی مجموعه‌ای از پردازش‌ها وجود داشته باشد {P0,P1,…,Pn} به طوری که P0 منتظر منبعی از P1 و P1 منتظر منبعی از P2 و ... و Pn منتظر منبعی از P0 باشد.

۴-۳. منبع بحرانی (Critical Resource) و بخش بحرانی (Critical Area)

منبعی است که بر سر تصاحب آن رقابت وجود دارد را منبع بحرانی گویند.
بخش بحرانی، بخشی از فرآیند است که با منبع بحرانی در ارتباط باشد.

نکته) اگر منبعی هنگام درخواست موجود نباشد، آن پردازش برای دریافت آن منبع در صف پردازش‌های منتظر قرار خواهد گرفت.
نکته) گرسنگی (قحطی) منبعی بیش از حد متعارف به فرآیند تخصیص داده شده و فرآیند برای گرفتن آن منتظر می‌ماند.

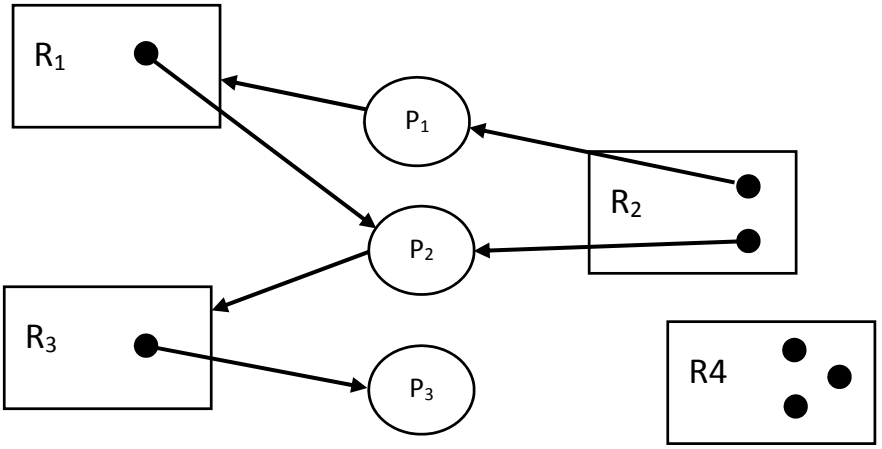


۴-۴. گراف تخصیص منبع

بن‌بست را می‌توان توسط گراف جهت‌داری به نام گراف تخصیص منبع رسم کرد. در این گراف گره‌ها از پردازش و منبع تشکیل شده‌اند و پردازش به صورت دایره و منبع به صورت مستطیل رسم می‌شود. تعداد نمونه‌های منبع را با نقطه درون مستطیل نشان می‌دهیم. هنگامی که پردازش Pi نمونه‌ای از منبع Rj را درخواست کند، یک کمان جهت‌دار از پردازش Pi به مستطیل Rj (Pi → Rj) رسم می‌شود و هنگامی که نمونه‌ای از منبع Rj به پردازش Pi تخصیص یابد یک کمان جهت‌دار از مستطیل Rj به دایره Pi رسم می‌شود. (Rj → Pi)

مثال) گراف تخصیص منبع زیر را رسم کنید.

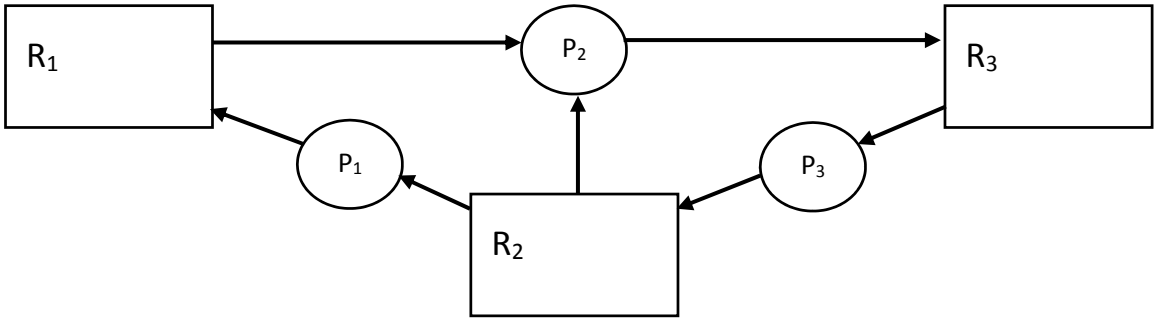
پردازش $P = \{P_1, P_2, P_3\}$
منبع $R = \{R_1 \text{ (یک نمونه)}, R_2 \text{ (دو نمونه)}, R_3 \text{ (یک نمونه)}, R_4 \text{(سه نمونه)}\}$
کمان $E = \{P_1 \to \mathrm{R}_1, \mathrm{P}_2 \to \mathrm{R}_3, R_1 \to \mathrm{P}_2, \mathrm{R}_2 \to \mathrm{P}_2, \mathrm{R}_2 \to \mathrm{P}_1, \mathrm{R}_3 \to \mathrm{P}_3\}$



اگر در گراف حلقه وجود نداشته باشد، آنگاه سیستم در حالت بن‌بست نیست.

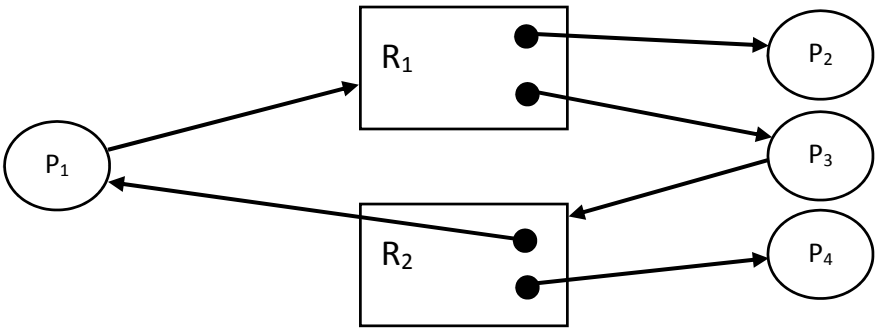
مثال) آیا گراف زیر در بن‌بست قرار دارد؟

$$R_2 \to P_1 \to R_1 \to P_2 \to R_3 \to P_3 \to R_2 \to P_2 \to R_3 \to P_3 \to R_2$$



در شکل هر سه پردازش دربن‌بست هستند، چون پردازش P2 منتظر R3 است و از طرفی R3 در اختیار P3 می‌باشد و از طرف دیگر P3 منتظر پردازش‌های P1 یا P2 می‌باشد که منبع R2 را رها کند. همچنین پردازش P1 منتظر P2 است تا R3 را رها کند.

مثال) آیا گراف زیر در بن‌بست قرار دارد؟



با این که در شکل حلقه وجود دارد، ولی از آنجا که ممکن است پردازش P4 تمام شود و منبع R2 را آزاد کند و منبع R2 به P3 تخصیص داده شود، در نتیجه حلقه شکسته می‌شود پس بن‌بستی وجود ندارد. همین روند ممکن است برای P2 نیز اتفاق بیفتد.

۴-۵. استراتژی‌های مدیریت بن‌بست

۴-۵-۱. صرف نظر کردن از بن‌بست

در این استراتژی در واقع هیچ عملی برای مقابله یا رفع بن‌بست استفاده نشده و در صورت وقوع بن‌بست سیستم به صورت دستی ریست (Reset) می‌شود.

نکته) در سیستم‌های امروزی چون بن‌بست به ندرت رخ می‌دهد، از این روش استفاده می‌کنند. این روش نسبت به روش‌های دیگر ارزان‌تر است.

۴-۵-۲. جلوگیری از بن‌بست

در این استراتژی تدبیری در نظر گرفته می‌شود که طی آن از یکی از ۴ شرط وقوع بن‌بست نقض شود.

الف) جلوگیری از انحصار متقابل: در این روش سعی می‌شود تا منابع تا جای ممکن به صورت اشتراکی استفاده شوند مثلا استفاده اشتراکی از فایل‌های Read-Only. البته این روش برای بعضی از منابع مانند چاپگر مناسب نیست.

ب) جلوگیری از Hold & Wait: در این روش دو دستورالعمل وجود دارد. در دستورالعمل اول فرآیند تنها در صورتی آغاز می‌شود که تمام مورد نظر آزاد باشند، در روش دوم که به آن روش جریمه گفته می‌شود اگر فرآیندی منابعی را در دست داشت و درخواست منبع جدیدی را کرد بایستی منابع قدیم خود را آزاد کنند و بعد منبع جدید را اخذ کنند.

ج) جلوگیری از انحصاری بودن: در این روش اگر پردازش منتظر منبعی باشد که در حال حاضر امکان تخصیص آن وجود ندارد. کلیه منابع تخصیص داده شده خود را پس می‌دهد و مجدداً وقتی آزاد می‌شود که همه منابع مورد نظرش آزاد باشد.

د) جلوگیری از انتظار چرخشی: برای این روش هر منبع یک شماره یکتا دارد. در این حالت هر پردازش فقط می‌تواند منابع را در جهت صعودی شماره‌هایشان درخواست کند مثلاً اگر CD-ROM شماره ۱ باشد و HDD شماره ۱۰ و Printer شماره ۱۵ ، اگر پردازشی هارد را داشته باشد، نمی‌تواند CD-ROM را تقاضا کند بلکه فقط Printer را می‌تواند بگیرد.

نکته) به طور کلی روش‌های جلوگیری از بن‌بست باعث بهره‌وری پایین و کاهش توان عملیاتی سیستم می‌شوند.

۴-۵-۳. اجتناب از بن‌بست

در این روش پویا، تخصیص منبع آتی مورد بررسی قرار می‌گیرد و در صورت وقوع بن‌بست فعلا در تخصیص آن منبع صرف نظر می‌شود.

حالت امن (Safe Mode): حالت تخصیص منبع در صورتی امن است که با توالی خاصی منابع به فرآیندها اختصاص داده شده و بن‌بست ایجاد نشود.

حالت ناامن (Unsafe Mode): حالتی که در آن سیستم حتی یک ترتیب امن برای تخصیص منابع به پردازش‌ها نتواند اتخاذ کند.

مثال) سیستمی را با ۱۲ منبع CD و ۳ پردازش در نظر بگیرید. نیازهای جاری سیستم در لحظه T0 مشخص شده است.

پردازش‌ها	حداکثر نیازها	نیازهای جاری
P0	10	5
P1	4	2
P2	0	2

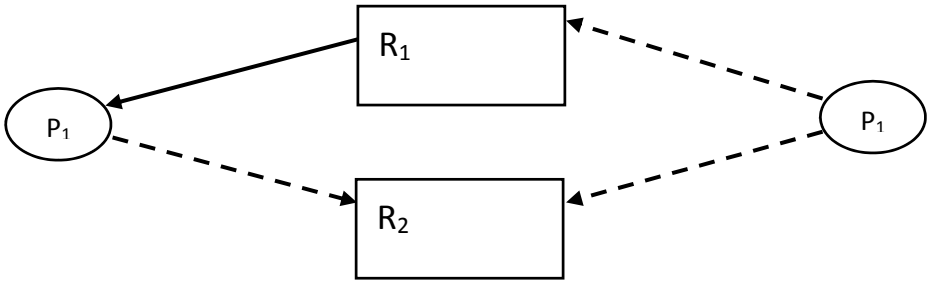
منابع آزاد $12 - 9 = 3$ ←

در این سیستم اگر ابتدا پردازش P1 و در انتها P2 منابع خود را دریافت کنند، بن‌بست رخ نمی‌دهد و سیستم به حالت امن می‌رود.

۴-۵-۳-۱. اجتناب از بن‌بست با گراف تخصیص منبع

اگر در سیستمی از هر نوع منبع فقط یک نمونه داشته باشیم، می‌توانیم از این روش استفاده کنیم. در این حالت علاوه بر کمان درخواست و تخصیص کمان، ادعا نیز داریم که با نقطه‌چین نمایش می‌دهیم. این کمان بیانگر این است که پردازش ممکن است در آینده منبعی را درخواست کند.

مثال) در گراف زیر اگر P2 تقاضا R2 را بکند، نباید آن را تخصیص دهیم هر چند R2 آزاد است چون کمان‌های ادعا را بررسی کردیم ممکن است باعث ایجاد حلقه در گراف شود یعنی سیستم به حالت ناامن برود.



۴-۵-۳-۲. اجتناب از بن‌بست با الگوریتم بانکدار

دو مفهوم ماتریس Claim و ماتریس Available

ماتریس Claim (درخواست‌ها):

	R_1	R_2	R_3
P_1	0	1	3
P_2	4	1	2
P_3	6	2	3

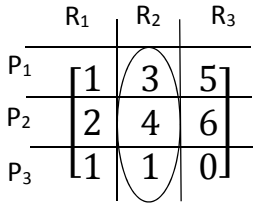
C_{ij} → شماره منبع

شماره پردازش

$1 + 1 + 2 = 4$

ماتریس Available: این که چه تعداد منابع اولیه وجود دارد.

ماتریس Allocation: این که چه تعداد از منابع به کدام پردازش‌ها تخصیص داده شده‌اند.



مثال) سیستمی دارای چهار پردازش و سه منبع می‌باشد. اگر ماتریس منابع اولیه به شکل زیر باشد، آیا سیستم در حالت امن قرار دارد؟

	R1	R2	R3
P1	3	2	2
P2	6	1	3
P3	3	1	2
P4	4	2	2

Claim

	R1	R2	R3
P1	1	0	0
P2	6	1	2
P3	2	1	1
P4	0	0	2

Allocation

Resources	9	3	6
	R1	R2	R3

مرحله ۱: محاسبه ماتریس Available:

ابتدا جمع ستون های ماتریس Allocation ⟵

9	2	5
R1	R2	R3

Resources - Allocation ⟵

0	1	1

مرحله ۲: ترتیب اجرا

اجرای P2	0	1	0	⟶	خاتمه P2	6	2	3
اجرای P1	4	0	1	⟶	خاتمه P1	7	2	3
اجرای P3	6	2	2	⟶	خاتمه P3	9	3	4
اجرای P4	5	1	4	⟶	خاتمه P4	9	3	6

سیستم با این ترتیب اجرا به حالت امن می‌رود.

مثال) در شرایط کنونی اگر P1 یک منبع R1 و یک منبع R3 درخواست کند، آیا بن‌بست رخ می‌دهد؟

3	2	2
6	1	3
3	1	2
4	2	2

Claim

1	0	0
5	1	1
2	1	1
0	0	2

Allocation

R_1	R_2	R_3
9	3	6

جمع ستون های Allocation

8	2	4

Available

1	1	2

Available جدید طبق درخواست جدید

0	1	1

با تخصیص منابع درخواستی P1 تعداد منابع R1 در ماتریس Available جدید صفر می‌شود. در صورتی که هر یک از فرآیندها به R1 برای اجرا احتیاج دارند. مثلاً P2 یک منبع یا P4 چهار منبع و ... احتیاج دارند. چون این تخصیص باعث بن‌بست می‌شود پس صورت نخواهد گرفت.

مثال)

	R1	R2	R3
P1	7	5	3
P2	3	2	2
P3	9	0	2
P4	2	2	2
P5	4	3	3

Claim

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

Resource

10	5	7

3	2	3

Available

اجرای P2	2	1	0	⟶	خاتمه P2	5	3	2
اجرای P4	5	2	1	⟶	خاتمه P4	7	4	3
اجرای P1	0	0	0	⟶	خاتمه P1	7	5	3
اجرای P3	1	5	3	⟶	خاتمه P3	10	5	5
اجرای P5	6	2	4	⟶	خاتمه P5	10	5	7

۴-۵-۴. کشف یا بازیابی بن‌بست

در این سیستم نه الگوریتم‌های پیشگیری و نه الگوریتم‌های اجتناب به کار گرفته می‌شود. لذا ممکن است بن بست رخ دهد که در این صورت سیستم بررسی شده و در صورت وقوع، بن‌بست تشخیص داده شده و الگوریتم دیگری سیستم را از بن‌بست خارج می‌کند.

۴-۵-۴-۱. کشف برای حالتی که چند نمونه از منبع وجود داشته باشد

در این حالت

۱- به ماتریس Allocation نگاه کنید. اگر سطری برابر صفر وجود داشت، آن سطر را علامت بزنید چون مطمئن هستیم فرآیند در بن‌بست نیست.

۲- مقداری که در Available قرار دارد را در یک متغیر (x) قرار می‌دهیم. در این مرحله سطری در ماتریس Claim می‌یابیم که مقدار آن نظیر به نظیر یا کمتر مساوی x باشد، اگر دو سطر این خاصیت را داشت، سطر بالاتر را انتخاب می‌کنیم و آن را در ماتریس Allocation علامت میزنیم چون قطعاً در بن‌بست نیست.

۳- در این مرحله مقدار x را با سطر ماتریس Allocation جمع می‌کنیم و به مرحله ۲ برمیگردیم، در نهایت فرآیندهایی که علامتی نمی‌خورند، قطعاً در بن‌بست هستند.

مثال)

	R1	R2	R3	R4	R5
P1	0	1	0	0	1
P2	0	0	1	0	1
P3	0	0	0	0	1
P4	1	0	1	0	1

Claim

	R1	R2	R3	R4	R5	
P1	1	0	1	1	0	✗
P2	1	1	0	0	0	✗
P3	0	0	0	1	0	✓
P4	0	0	0	0	0	✓

Allocation

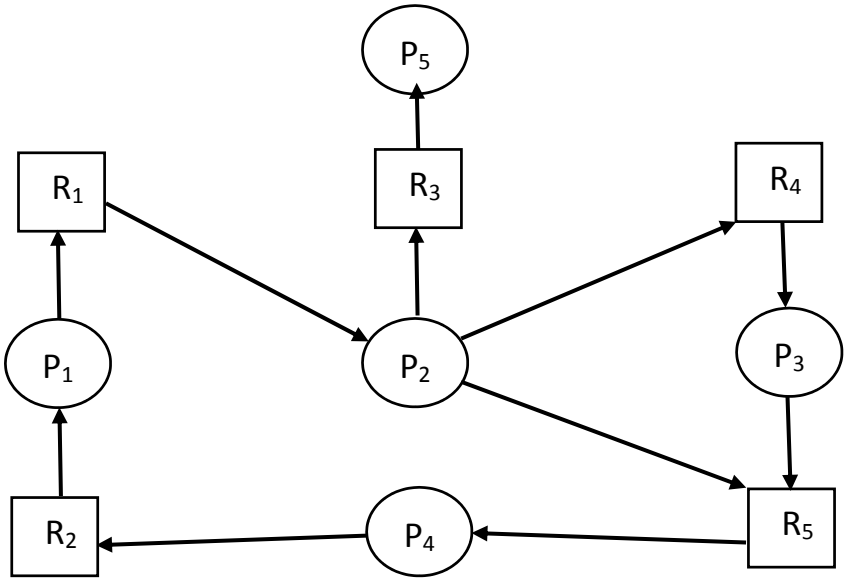
	R1	R2	R3	R4	R5
Resources	2	1	1	2	1
Available	0	0	0	0	1

$$X = \begin{array}{r} 00001 \\ +\ 00010 \\ \hline 00011 \end{array}$$

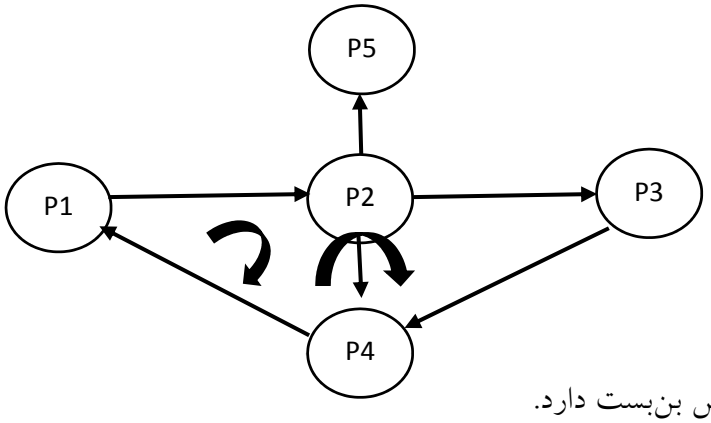
۴-۵-۴-۲. کشف برای حالتی که یک نمونه از منبع وجود داشته باشد

در این حالت بایستی از گراف انتظار استفاده کرد. اگر در این گراف دور وجود داشته باشد، فرآیند مذکور در بن‌بست خواهد بود.

مثال)



حل:



دو دور دارد، پس بن‌بست دارد.

۴-۵-۴-۳. خاتمه پردازش (بازیابی)

در این حالت با خاتمه پردازش‌های درگیر بن‌بست منابع آن‌ها آزاد شده و به سیستم بازگردانده می‌شود. این روش هزینه زیادی به دنبال دارد مثلاً ممکن است بخش عمده پردازش انجام شده و با خاتمه آن مجبوریم فرآیند زا مجدداً از ابتدا آغاز کنیم.

نکته) بازپس‌گیری منابع: در این روش اجرای فرآیندی به صورت موقت متوقف شده و منابع آن در اختیار فرآیندهای بن‌بست شده قرار می‌گیرد.

فصـــل پنجم

مدیریت حافظه

۵-۱. مدیریت حافظه

یکی از وظایف سطح بالا سیستم عامل مدیریت حافظه بوده و بعد از CPU، حافظه بحرانی‌ترین منبع در کامپیوتر محسوب می‌شود.

تعریف مدیریت حافظه (Memory Management): به معنی استفاده بهینه از حافظه در سیستم است که توسط مدیر حافظه در سیستم عامل انجام می‌شود.

۵-۲. مدیریت حافظه در سیستم‌های قدیمی

۱- سیستم‌های تک برنامه‌ای: کل برنامه به فضای حافظه آورده می‌شود و اگر حجم برنامه بیشتر از فضای حافظه بود برنامه اجرا نمی‌شد.

۱- سیستم‌های تک برنامه‌ای با تکنیک Overlay: در این مورد برنامه به بخش‌های مجزا تقسیم شده و در هر زمان، بخش مورد نیاز به حافظه منتقل شده و سایر بخش‌های برنامه در حافظه جانبی باقی می‌ماند.

۲- سیستم‌های چند برنامه‌ای: در این حالت حافظه به بخش‌های مجزا تقسیم شده و برنامه‌ها در این بخش‌های مجزا قرار می‌گیرند.

تکنیک مبادله (Swapping): در این حالت هر برنامه به صورت کامل به رم آورده شده، مدتی در فضای رم اجرا می‌شود و سپس به حافظه جانبی بازگردانده می‌شود و پردازش دیگری جایگزین آن می‌شود.

۵-۳. تقسیم‌بندی فیزیکی و منطقی حافظه

در مدیریت حافظه چندین موضوع کلی وجود دارد:
تقسیم‌بندی فیزیکی و منطقی حافظه: در واقع هارد دیسک به صورت فیزیکی تقسیم می‌شود اما برنامه‌ها به صورت منطقی تقسیم‌بندی می‌شوند.

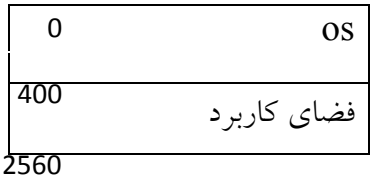
آدرس فیزیکی و منطقی: در واقع آدرس یک خط از برنامه هنگام بارگذاری و اجرا در رم آدرس فیزیکی است و آدرس تولید شده توسط CPU را آدرس منطقی می‌گوییم.

۵-۳-۱. عمل نگاشت (Binding)

نگاشت یا تبدیل یک فضای آدرس به فضای دیگر را Binding می‌گوییم. هنگامی که عمل Binding در حین اجرا صورت گیرد به آدرس منطقی اصطلاحاً آدرس مجازی یا Virtual Address گفته می‌شود. تبدیل آدرس مجازی به فیزیکی توسط واحد MMU (Management Unit Memory) صورت میگیرد.

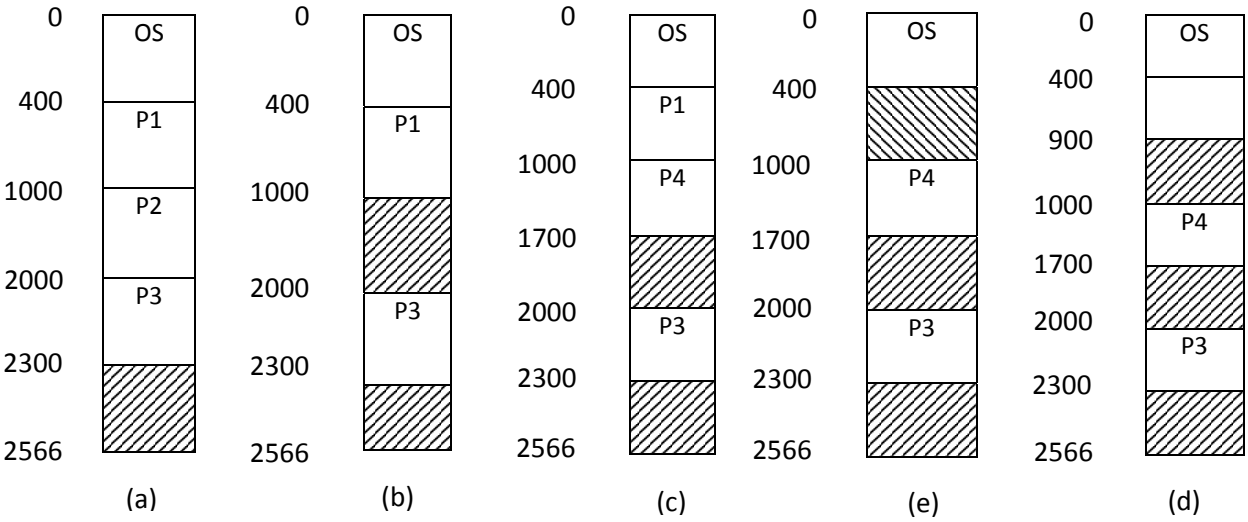
۵-۴. روش‌های تخصیص حافظه

مثال) حافظه‌ای 2560k است که 400k سیستم عامل گرفته و باقی مانده 2160k.



- زمان بندی انتخاب کار از نوع FIFO
- زمان بندی RR با کوانتوم ۱

زمان مورد نیاز	حافظه مورد نیاز	پردازش
10	600k	P1
5	1000k	P2
20	300k	P3
8	700k	P4
15	500k	P5



۵-۵. روش‌های تخصیص حافظه به پردازش

۱-۵-۵. First Fit

در این حالت سیستم عامل در فضاهای خالی حافظه اولین حفره‌ای که گنجایش فرآیند را داشته باشد برای آن فرایند در نظرمی‌گیرد. در این روش تراکم خانه‌های اشغال شده در ابتدای رم بیشتر است، همچنین این روش سرعت بالایی دارد.

۲-۵-۵. Next Fit

مانند First Fit است با این تفاوت که جستجو از محلی در لیست آغاز می‌شود که آخرین بار تخصیص از آن محل صورت گرفته است. به این ترتیب توزیع برنامه‌ها در سطح حافظه یکنواخت خواهد بود پس بهینه است ولی نسبت به First Fit زمان جستجوی زیادی نیاز دارد.

۳-۵-۵. Best Fit

در این روش کل فضای آزاد جستجو شده و کوچکترین حفره‌ای که به اندازه‌ی کافی بزرگ است به پردازش تخصیص داده می‌شود چون تمام فضاهای آزاد باید بررسی شود. این تکنیک زمان‌بر است ولی بهینه است چون فضاهای بزرگتر برای تقاضاهای بیشتر حفظ می‌شوند.

۴-۵-۵. Worst Fit

در این روش بزرگترین خانه برای یک پردازش در نظر گرفته می‌شود. ایراد این تکنیک تقاضاهایی است که ناحیه‌ی بزرگی می‌خواهند که شاید دیگر نتوانند برآورده شوند ولی این تکنیک نسبت به Best Fit باتوجه به این که در Best Fit حفره‌های باقی مانده آنقدر کوچک هستند که دیگر قابل استفاده نباشند، شاید مناسب‌تر باشد چون حفره‌های باقی مانده بزرگتری باقی می‌ماند.

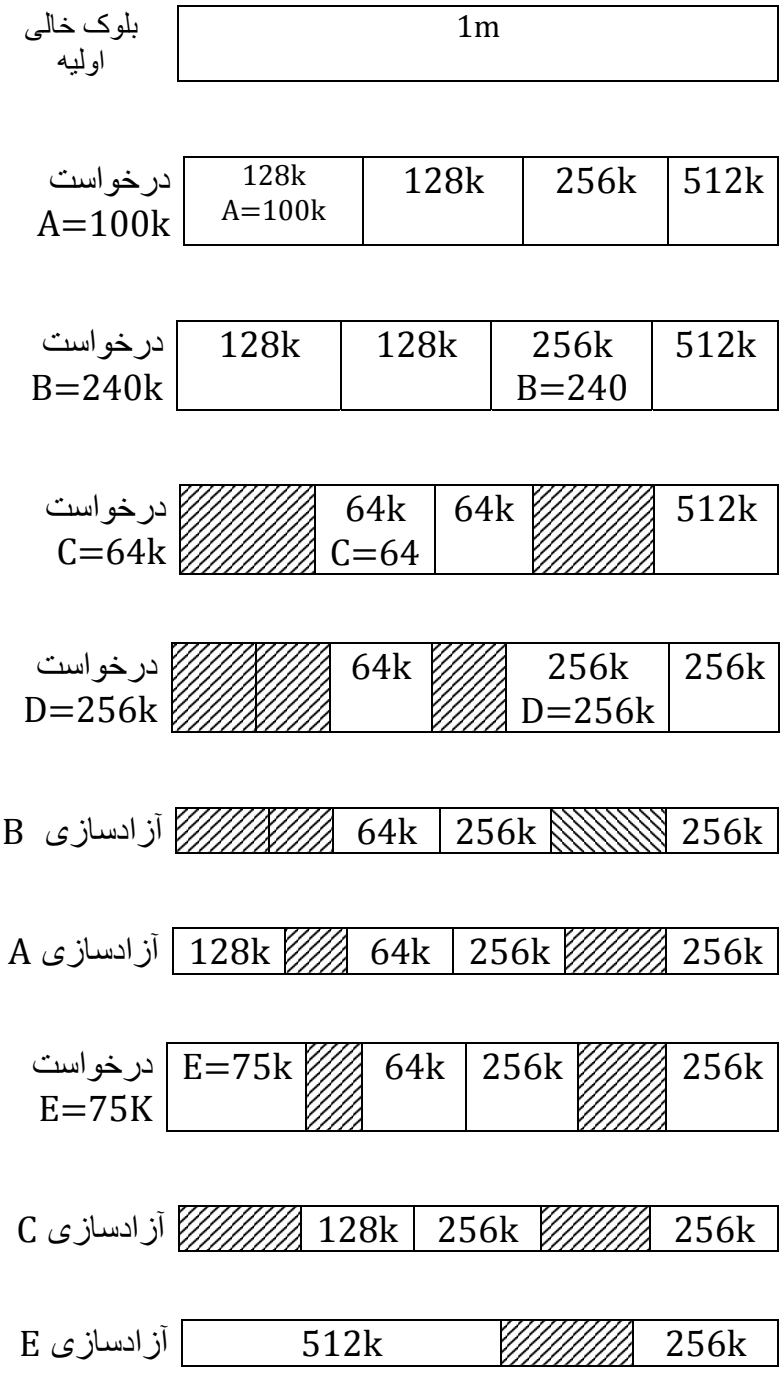
۵-۵-۵. Quick Fit

در این روش برای هر دسته از پردازش‌ها یک لیست تهیه می‌شود مثلاً این الگوریتم دارای جدولی با n خانه است که خانه‌ی اول اشاره‌گری به ابتدای لیستی شامل حفره‌های 4k است، خانه‌ی دوم به لیست حفره‌های 8k اشاره می‌کند و غیره. با این الگوریتم پیدا کردن حفره‌ای با اندازه‌ی مناسب سریع است ولی

عیب آن این است که اگر پردازشی خاتمه یابد، باید فضای آزاد شده‌ی آن به لیست مناسب اضافه شود که این زمان‌بر است.

Buddy ۶-۵-۵

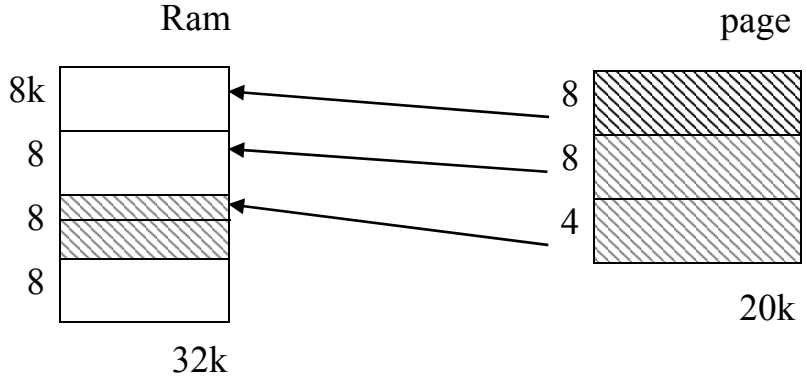
در این روش همه‌ی بخش‌های آزاد حافظه به قطعاتی که همگی توانی از ۲ هستند تقسیم می‌شوند. در این روش بلوک‌های کنار همدیگر ترکیب می‌شوند و بخش‌های بزرگتری را پدید می‌آورند.
مثال)



۵-۶. روش‌های تقسیم‌بندی منطقی حافظه

۵-۶-۱. روش صفحه‌بندی (Paging)

در این روش حافظه فیزیکی به بلوک‌هایی با اندازه‌ی ثابت به نام قاب‌ها یا Frames تقسیم بندی می‌شوند. حافظه‌ی منطقی نیز به بلوک‌هایی با اندازه‌ی یکسان به نام صفحه یا Page به اندازه‌ی Frames بوده و در آن جایگذاری می‌شود.

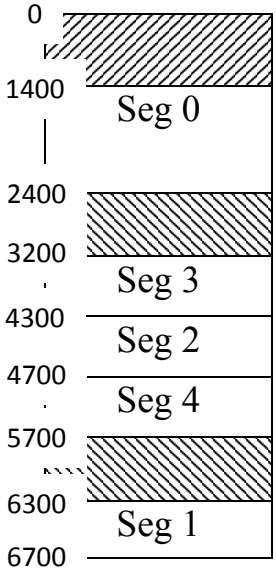


۵-۶-۲. روش قطعه‌بندی (Segmentation)

در این روش به جای آن که حافظه تقسیم شود، فرآیند به بخش‌هایی تقسیم می‌شود که این بخش‌ها لزوماً مساوی نیستند. به هر بخش سگمنت گفته می شود و سگمنت‌ها درحفره های حافظه‌ی رم جایگزین می شوند.

مثال) فرض کنید برنامه‌ای شامل ۵ سگمنت است. با توجه به جدول SMT فضای رم را ترسیم کنید (تبدیل آدرس منطقی به فیزیکی)

	limit	Base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700



نکته) جدول سگمنت مشخص می‌کند که هر قطعه بایستی به کدام آدرس در فضای رم نگاشته شود. علاوه بر آن آدرس شروع هر قطعه در ستون Base و طول مجاز هر قطعه در ستون Limit مشخص شده است.

۵-۷. مقایسه صفحه‌بندی و قطعه‌بندی

۱- در صفحه بندی اندازه هر Page ثابت بوده است و RAM به فریم‌های مساوی تقسیم می‌شود اما در قطعه‌بندی اندازه‌ی هر قطعه متغییر بوده و RAM تقسیم نمی‌شود.

۲- اندازه‌ی هر سگمنت را برنامه نویس به صورت غیرمستقیم و کامپایلر به صورت مستقیم تعیین می‌کند.

۳- در صفحه‌بندی، سیستم عامل همیشه باید لیست فریم‌های خالی در RAM را نگهداری کند. در قطعه بندی سیستم عامل بایستی لیست حفره‌های خالی RAM را نگهداری کند.

۴- در صفحه‌بندی مشکل قطعه قطعه شدن داخلی وجود دارد ولی در قطعه‌بندی این مشکل وجود ندارد. در صفحه‌بندی مشکل قطعه قطعه شدن خارجی وجود ندارد ولی در قطعه‌بندی وجود دارد چون پس از اتمام کار سگمنت‌های برنامه حفره‌های خالی در جای جای RAM وجود خواهد داشت، این باعث می‌شود سیستم عامل زمان خود را صرف Defrag کردن بکند و کارایی کاهش پیدا کند.

فصـــل ششم

همروندی فرایند ها و نواحی بحرانی

۶-۱. دسته بندی محاوره و ارتباط بین فرایندها

محاوره و ارتباط بین فرایند ها بر اساس میزان اطلاع فرایند ها از هم به سه دسته تقسیم می شوند:

۱- همکاری به وسیله اشتراک

در این حالت فرایندها به صورت کاملا غیر مستقیم مثلا از طریق یک شیء از وجود هم مطلع می شوند. در همچنین حالتی احتمال دارد نتایج حاصل از اجرای یک فرایند به نتایج فرایند دیگر وابستگی داشته باشد. مواردی که سیستم عامل باید کنترل کند: بن بست، گرسنگی، انحصار متقابل.

۲- همکاری به وسیله ارتباط

در این حالت فرایندها به صورت مستقیم از وجود همدیگر مطلع هستند و وابستگی نتایج بین فرایند ها وجود دارد. مواردی که سیستم عامل باید کنترل کند: بن بست، گرسنگی.

۳- رقابت

در این حالت فرایندها از وجود همدیگر بی اطلاع هستند. و باهم کار نمی کنند. در نتیجه نتایج حاصل از اجرای یک فرایند مستقل از عملکرد فرایند دیگر است. مواردی که سیستم عامل باید کنترل کند: بن بست، گرسنگی، انحصار متقابل.

نکته) شرایط رقابتی: شرایطی است که چند فرایند در حال نوشتن یا خواندن داده ها به صورت اشتراکی باشند. نتیجه نهایی بستگی به اینکه کدام فرایند چه موقع اجرا شود، دارد.

۶-۲. ناحیه بحرانی

وقتی فرایندها برای دسترسی به داده ها یا منابع مشترک با یکدیگر رقابت می کنند، در هر فرایند قطعه کدی وجود دارد که با اجرای آن به ناحیه بحرانی دسترسی پیدا می کند. وقتی فرایند ناحیه بحرانی خود را اجرا می کند شرایط رقابتی ایجاد می شود. ترتیب اجرای فرایند می تواند در نتیجه تاثیر گذار باشد. اگر بتوانیم ترتیبی ایجاد کنیم که هیچ دو پردازشی در یک زمان وارد ناحیه بحرانی نشوند از شرایط رقابتی جلوگیری کردیم.

۶-۳. شرایط ارائه شده برای حل مساله ناحیه بحرانی

۱- شرط پیشرفت: هیچ فرایندی نباید از بیرون ناحیه بحرانی خود بر روی دیگر فرایند ها تاثیر بگذارد و در واقع نباید امکان بلوکه کردن دیگر فرایندها را داشته باشد.
۲- انحصار متقابل وجود داشته باشد: تنها یک فرایند بتواند ناحیه بحرانی خود را اجرا کند.
۳- شرط انتظار مقید: هیچ فرایندی نباید بی نهایت منتظر ورود به ناحیه بحرانی اش باشد.
۴- هر فرایند تنها برای زمان محدودی می تواند در داخل بخش بحرانی خودش باقی بماند.

۶-۴. راه حل های ارائه شده برای مساله ناحیه بحرانی

راه حل ها شامل موارد زیر هستند.

۶-۴-۱. راه حل نرم افزاری

۶-۴-۱-۱. استفاده از متغیر قفل (Lock)

در این راه حل یک متغیر قفل به صورت مشترک بین فرایند ها مورد استفاده قرار می گیرد. مقدار اولیه این قفل صفر است. هر فرایند برای ورود به ناحیه بحرانی خودش ابتدا مقدار قفل را بررسی می کند. اگر مقدار برابر با صفر باشد پس می تواند وارد ناحیه بحرانی شود، قبل از ورود به ناحیه بحرانی، مقدار قفل را یک می کند که فرایند ها دیگر با بررسی مقدار قفل (چون مقدار یک شده است) نتوانند وارد ناحیه بحرانی بشوند.

```
int lock = 0;
```

P1:	P2:	P3:
…	…	…
While (lock == 1) do;	While (lock == 1) do;	While (lock == 1) do;
Lock = 1 ;	Lock = 1 ;	Lock = 1 ;
ناحیه بحرانی	ناحیه بحرانی	ناحیه بحرانی
Lock = 0;	Lock = 0;	Lock = 0;
…	…	…

در این روش انحصار متقابل برآورده نمی شود. چون ممکن است قبل از اینکه فرایندی lock را یک کند، برش زمانی اش پایان یابد و در همین لحظه فرایند دیگر با دیدن مقدار صفر برای lock وارد ناحیه بحرانی شود. پس دو فرایند وارد ناحیه بحرانی می شوند. پس راه حل خوبی نیست.

۶-۴-۱-۲. استفاده از متغیر تناوبی (نوبتی)

در این روش یک متغیر نوبت (turn) وجود دارد که مقدار آن یا می تواند صفر باشد یا می تواند یک باشد. و تنها دو فرایند با نام های p0 و p1 می توانند برای رسیدن به ناحیه بحرانی تلاش کنند. اگر مقدار متغیر نوبت برابر با شماره فرایند باشد، فرایند می تواند ناحیه بحرانی خود را اجرا کند، در غیر این صورت نوبت برای فرایند دیگر خواهد بود.

در این روش انحصار متقابل برقرار است. شرط انتظار مقید هم برقرار است. شرط پیشرفت برقرار نیست.

```
int turn = 0;
```

```
P0:
...
While (1) {
While (turn != 0) do;
ناحیه بحرانی
turn = 1;
}
...
```

```
P1:
...
While (1) {
While (turn != 1) do;
ناحیه بحرانی
turn = 0;
}
...
```

۶-۴-۱-۳. استفاده از متغیر Flag (روش اول)

در این روش یک متغیر آرایه منطقی به اسم فلگ دارای دو خانه در نظر گرفته می شود. مقدار اولیه خانه ها false است. هر خانه از آرایه مربوط به یک فرایند است. یعنی flag[0] مربوط به فرایند p0 و flag[1] مربوط به فرایند p1 است. هر فرایند برای ورود به ناحیه بحرانی فلگ رقیب را بررسی می کند. اگر true باشد نمی تواند وارد ناحیه بحرانی شود. اگر false باشد، ابتدا فلگ خود را true می کند و وارد ناحیه بحرانی می شود.

boolean flag[2] = false;

P0:	P1:
…	…
While (flag[1]==true) do	While (flag[0]==true) do
Flag[0]=true;	Flag[1]=true;
ناحیه بحرانی	ناحیه بحرانی
Flag[0]=false;	Flag[1]=false;
…	…

در این روش انحصار متقابل برآورده نمی شود. چون ممکن است قبل از true شدن فلگ، برش زمانی پردازش تمام شود، در نتیجه فلگ تغییر نمی کند و فرایند دیگر می تواند وارد ناحیه بحرانی شود، بنابراین هردو وارد ناحیه بحرانی می شوند.

شرط پیشرفت برآورده می شود. چون اگر فرایندی نخواهد وارد ناحیه بحرانی شود مقدار فلگ را برابر true نمی کند و تاثیری بر روی فرایند دیگر ندارد.

شرط انتظار مقید برآورده نیست. چون اگر فرایندی در مثلا حلقه خود گرفتار شود فلگ خود را تغییر نداده و در نتیجه فرایند دیگر باید زمان نامشخصی منتظر بماند.

۶-۴-۱-۴. استفاده از متغیر Flag (روش دوم)

مثل روش قبل است فقط جای دستور حلقه و مقداردهی فلگ با هم عوض شده است.

boolean flag[2] = false;

P0:	P1:
…	…
Flag[0]=true;	Flag[1]=true;
While (flag[1]==true) do	While (flag[0]==true) do
ناحیه بحرانی	ناحیه بحرانی
Flag[0]=false;	Flag[1]=false;
…	…

این روش انحصار متقابل را برآورده می کند. چون فرایندی که وارد ناحیه بحرانی می شود اول فلگ رقیب را false دیده است، یعنی رقیب در ناحیه بحرانی نبوده است و دوم فلگ خودش را true می کند. با این کار مانع ورود رقیب به ناحیه بحرانی می شود.

شرط پیشرفت برقرار است. چون هر فرایند بعد از اتمام کار خودش فلگ خودش را false می کند پس بعد از آن، رقیب می تواند وارد ناحیه بحرانی شود.
شرط انتظار مقید وجود ندارد به خاطر وجود برش زمانی.

۶-۴-۱-۵. استفاده از روش پیترسون

این روش ترکیب روش های قبلی است. یعنی به صورت همزمان از متغیر نوبت و فلگ استفاده می کند.

```
boolean flag[2] = false;
turn = 0;
```

P0:	P1:
…	…
Flag[0]=true;	Flag[1]=true;
Turn = 1;	Turn = 0;
While (flag[1]==true && turn==1) do	While (flag[0]==true && turn==0) do
ناحیه بحرانی	ناحیه بحرانی
Flag[0]=false;	Flag[1]=false;
…	…

این روش انحصار متقابل را برقرار می سازد. چون فرایند در صورتی می تواند وارد ناحیه بحرانی شود که فلگ رقیب fasle باشد و turn به آن اجازه ورود بدهد. چون turn یعنی در لحظه فقط یک فرایند می تواند وارد ناحیه بحرانی شود، پس انحصار متقابل برقرار است.
شرط انتظار مقید به دلیل اینکه دائما نوبت به رقیب داده می شود، کاملا برقرار است.
شرط پیشرفت نیز برقرار است. چون هر فرایند هنگام خروج از ناحیه بحرانی فلگ خود را false می کند پس به رقیب اجازه ورود به ناحیه بحرانی می دهد و مانع اجرای آن نمی شود.

نکته) راه حل پیترسون چون هر سه شرط را برقرار دارد، راه حل قابل قبولی در نظر گرفته می شود. ولی این راه حل دو پردازشی است.

۶-۴-۲. راه های سخت افزاری

در این نوع راه حل ها به کمک دستورات زبان ماشین و اسمبلی و یا از کار انداختن وقفه ها می توانیم ناحیه بحرانی را کنترل کنیم.

۶-۴-۲-۱. از کار انداختن وقفه ها

در این روش فرایند هنگام ورود به ناحیه بحرانی همه وقفه ها را از کار می اندازد و پس از خروج از ناحیه بحرانی دوباره وقفه های را فعال می کند. در این حالت چون وقفه مربوط به برش زمانی از کار افتاده است، بنابراین وقتی فرایندی در حال اجرا است، قطع نمی شود و همچنین پردازنده نمی تواند بین فرایند ها سوییچ کند.

pi = i i ∈ {1,2,3, ... }
...
Disable INT
ناحیه بحرانی
Enable INT
...

شرط انحصار متقابل برقرار است. چون اگر فرایندی بتواند وقفه ها را از کار بی اندازد و وارد ناحیه بحرانی شود، فرایندهای دیگر نمی توانند وارد ناحیه بحرانی شوند.
شرط پیشرفت برقرار است. چون هنگام خروج از ناحیه بحرانی، تمام وقفه های غیر فعال شده، دوباره فعال می شوند، بنابراین فرایند های دیگر می توانند با غیر فعال کردن وقفه ها وارد ناحیه بحرانی شوند.
شرط انتظار مقید برقرار نیست چون وقتی وقفه ها غیر فعال می شوند هیچ مکانیزمی برای کنترل فرایند های دیگر جهت استفاده از ناحیه بحرانی نداریم.

۶-۴-۲-۲. استفاده از دستورات TSL (Test and Set Lock)

برای استفاده از این راه حل دو مورد از طرف سخت افزار باید برقرار باشد: مورد اول اینکه عملیات خواندن کلمه از حافظه و ذخیره سازی مقدار آن باشد اتمیک (تجزیه ناپذیر) باشد. مورد دوم اینکه در هنگام انجام این عمل هیچ فرایند دیگری یا حتی پردازنده دیگری حق دسترسی به این کلمه از حافظه را ندارد. هنگام استفاده از این روش گذرگاه مربوط به حافظه قفل می شود بنابراین از دسترسی دیگر فرایندها خارج می شود. دستور TSL محتویات یک کلمه را از حافظه خوانده و در یک ثبات قرار می دهد و سپس مقدار آن خانه حافظه را یک می کند.
هنگامی که فرایند می خواهد وارد ناحیه بحرانی شود تابع enter و هنگامی که می خواهد از ناحیه بحرانی خارج شود تابع exit صدا زده می شود.

int lock = 0; متغیر مشترک میان فرایندها

Enter:	Exit:	pi = i i ∈
TSL Reg,Lock	mov lock,0	{1,2,3, … }
cmp Reg, 0		…
jne Enter		Enter
		ناحیه بحرانی
		Exit
		…

شرط انحصار متقابل برقرار است. زیرا اولین فرایند که enter را فراخوانی کند مقدار صفر را از lock به reg منتقل می کند و مقدار lock را برابر با یک می کند. چون مقدار reg برابر صفر شده است این فرایند وارد ناحیه بحرانی می شود ولی سایر فرایند ها نمی توانند وارد ناحیه بحرانی شوند.

شرط پیشرفت برقرار است چون فرایند هنگام خروج از ناحیه بحرانی مقدار lock را به وسیله تابع exit برابر صفر قرار می دهد یعنی دوباره قفل باز شده است بنابراین دیگر فرایندها می توانند با دیدن قفل باز وارد ناحیه بحرانی شوند.

شرط انتظار مقید بر قرار نیست. چون هیچ مکانیزمی برای نوبت دهی یا کنترل فرایندهایی که می خواهند وارد ناحیه بحرانی شوند، وجود ندارد.

۶-۴-۲-۳. استفاده از دستورات مربوط به تعویض یا swap یا exchange

این روش به صورت دستورالعمل های زبان ماشین انجام می شود. به طوری که محتوای یک ثبات را با محتوای محلی از حافظه عوض می کند. عمل تعویض به صورت اتمیک است. در طی عملیات تعویض هیچ فرایند یا پردازنده دیگری نمی تواند به همان محل حافظه مراجعه کند. گذرگاه حافظه قفل می شود.

int lock = 0; متغیر مشترک میان فرایندها

P1:	P2:	P3:
…	…	…
Key1 = 1	Key2 = 1	Key3 = 1
Do	Do	Do
Exch(key1, lock)	Exch(key2, lock)	Exch(key3, lock)
While (key1==1)	While (key2==1)	While (key3==1)
ناحیه بحرانی	ناحیه بحرانی	ناحیه بحرانی
Lock=0	Lock=0	Lock=0
…	…	…

در این روش lock بین همه فرایندها مشترک است اما هر فرایند key مربوط به خود را دارد.
انحصار متقابل برقرار است. در این روش اولین فرایندی که دستور exch را اجرا کند مقدار lock را برابر با یک کرده ولی مقدار key خودش را برابر صفر کرده است و وارد ناحیه بحرانی می شود و در نتیجه دیگر فرایند ها نمی توانند وارد ناحیه بحرانی شوند چون با اجرای exch مقدار key آن ها برابر با یک می شود.
مانند روش TSL شرط پیشرفت برقرار است ولی شرط انتظار مقید برقرار نیست.

۶-۴-۳. راه حل هایی که توسط سیستم عامل ارائه می شوند

۶-۴-۳-۱. روش سمافور (Semaphore)

یک سمافور کی ساختار سطح بالای برنامه نویسی دارای شیء و کلاس است که دارای داده و تابع نیز می باشد. داده های یک سمافور شامل متغیر صحیح value و یک صف Q برای قرار دادن نام فرایندها هست. توابع عبارتند از wait (Down) و signal (up) به صورت اتمیک هستند. یعنی موقع فراخوانی برش زمانی پایان نمی یابد.
تابع wait: از مقدار value یک واحد کم می کند و اگر مقدار منفی شود فرایند فراخواننده را مسدود کرده و به صف Q اضافه می کند.
تابع signal: به مقدار value یک واحد اضافه می کند و اگر این مقدار کوچکتر یا مساوی صفر باشد یکی از فرایندهای مسدود را از صف Q به صف Ready اضافه می کند.

پیاده سازی اول)

```
Wait(s) {
s.value = s.value – 1;
   if (s.value < 0) {
      block process P;
      add P to s.Q;
   }
}
```

```
signal(s) {
s.value = s.value + 1;
   if (s.value < 0) {
      wakeup process P from s.Q;
      add P to s.Q;
   }
}
```

پیاده سازی دوم)

```
Wait(s) {
if (s.value > 0)
    s.value = s.value – 1;
else {
      block process P;
      add P to s.Q;
   }
}
```

```
signal(s) {
if (s.Q is empty)
    s.value = s.value + 1;
else {
      wakeup process P from s.Q;
      add P to Ready;
   }
}
```

یک سمافور S با مقدار اولیه "یک" به صورت مشترک میام تمام فرایندهای مد نظر، در نظر گرفته می شود. هر فرایند جهت ورود به ناحیه بحرانی wait(s) فراخوانی می کند. اولین فرایندی که wait(s) را فراخوانی کند مقدار Value را از یک به صفر تغییر می دهد و بدون اینکه مسدود شود وارد ناحیه بحرانی می شود. اما فرایندهای بعدی با فراخوانی wait(s) مقدار Value را منفی خواهند کرد و مسدود خواهند شد و نام آن ها در صف Q نوشته می شود پس در نتیجه شرط انحصار متقابل فراهم می شود.

هر فرایند هنگام خروج از ناحیه بحرانی signal(s) فراخوانی می کند که باعث می شود یک واحد به s.value اضافه شود. اگر s.value بزرگتر مساوی صفر باشد به این معنی است که فرایندی مسدود وجود دارد، پس فرایند ابتدای صف Q وارد ناحیه بحرانی می شود. بنابراین شرط انتظار مقید وجود دارد. چون فرایندهایی که نمی توانند وارد ناحیه بحرانی شوند در صف قرار می گیرند و از صف برداشته شده و وارد ناحیه بحرانی می شوند.

شرط پیشرفت برقرار است چون هر فرایند با خروج از ناحیه بحرانی signal(s) را اجرا می کند و با این کار آثار مربوط به wait(s) را از بین می برد پس بقیه فرایندها می توانند با فراخوانی wait(s) وارد ناحیه بحرانی شوند.

```
Semaphore S(1);      // سمافور مشترک بین فرایندها با مقدار اولیه یک
Pi       i ∈ {1,2,3, … }
…
Wait(s);
ناحیه بحرانی
Signal(s);
…
```

نکته ۱) اگر s.value کوچکتر از صفر باشد، قدرمطلق آن تعداد فرایندهای مسدود شده را نشان می دهد.
نکته ۲) اگر s.value بزرگتر از صفر باشد، نشان دهنده تعداد فرایندهایی است که می توانند وارد ناحیه بحرانی شوند.

۶-۴-۴. راه حل هایی که توسط زبان های برنامه نویسی ارائه می شوند

۶-۴-۴-۱. روش مانیتور (Monitor)

یک مانیتور یک ساختار سطح بالای برنامه نویسی یا یک شیء یا کلاس است که دارای تابع و داده است. داده های یک مانیتور عبارتند از هر داده دلخواه برنامه نویسی از هر نوع دلخواه. و علاوه بر این یک نوع داده جدید به نام نوع داده شرطی در مانیتور وجود دارد. متغیرهای شرطی فاقد مقدار هستند و فقط دارای یک صف Q هستند. یعنی هر متغیر شرطی یک صف Q است. توابع مانیتور می توانند هر تابع دلخواه کاربر باشند. همچنین دو تابع wait و signal بر روی متغیرهای شرطی درون مانیتور قابل اعمال است.

نکته ۱) متغیرهای درون مانیتور از نوع private هستند.
نکته ۲) هرگاه فرایندی از طریق فراخوانی یک تابع از یک مانیتور وارد مانیتور شود تا زمانیکه اجرای این تابع تمام نشود و فرایند از مانیتور خارج نشود، کامپایلر تضمین می کند که هیچ فرایند دیگری حق فراخوانی هیچ تابع دیگری را از مانیتور ندارد و نمی تواند وارد مانیتور بشود.

وجود نکته ۱ و نکته ۲ برای هر داده باعث می شود که شرط انحصار متقابل وجود داشته باشد.
تابع wait روی متغیر شرطی فراخوانی می شود و فرایند فراخواننده را مسدود می کند و نام آن را وارد صف Q می کند.
تابع signal روی متغیر شرطی فراخوانی می شود و فرایند موجود در صف Q را بیدار کرده (wakeup) و برای ارسال به ناحیه بحرانی آن را آماده می کند.

۶-۴-۵. راه حل تبادل پیام (Message Passing)

تمام راه حل های بیان شده در بالا، برای حالتی است که حافظه مشترک وجود داشته باشد. اگر سیستم های توزیع شده را داشته باشیم می توانیم از راه حل تبادل پیام استفاده کنیم.

در این مکانیزم، دو تابع send و receive برای ارسال و دریافت پیام بین فرایندها استفاده می شود. تابع send (mes, dest) پیام mes را به مقصد dest ارسال می کند. تابع receive (mes, sor) پیام mes را از مبدا sor دریافت می کند.

به طور کلی تابع receive مسدود شونده است یعنی وقتی فرایندی این تابع را فراخوانی می کند اگر پیامی از سوی مبدا دریافت نکند مسدود می شود. تابع send مسدود شونده نیست.

```
P1:
Send(mes, p2);
While (1){
Receive(mes, p2);
ناحیه بحرانی
Send(mes, p2);
}
…
```

```
P2:
…
While (1){
Receive(mes, p1);
ناحیه بحرانی
Send(mes,p1);
}
…
```

این راه حل، انحصار متقابل را برقرار می سازد. شرط پیشرفت را ندارد چون فرایندها به صورت یکی در میان وارد ناحیه بحرانی می شوند. انتظار مقید را نیز برقرار می کند. این راه حل برای دو پردازش استفاده می شود.

اگر بخواهیم راه حل تبادل پیام را برای چند فرایند اعمال کنیم، می توانیم از صندوق پستی مشترک میان فرایندها استفاده کنیم. در ابتدا تنها یک پیام خالی در این صندوق قرار داده می شود. هر فرایندی که بتواند پیام خالی را دریافت کند می تواند ناحیه بحرانی خود را اجرا کند. بعد از خاتمه ناحیه بحرانی، دوباره پیامی خالی را به صندوق ارسال می کند که فرایندهای دیگر بتوانند با دریافت آن پیام، وارد ناحیه بحرانی شوند.

```
Pi        i ∈ {1,2,3, … }
…
While (1) {
Receive(mes, mailbox);      // دریافت پیام از صندوق پستی
ناحیه بحرانی
Send(mes, mailbox);         // ارسال پیام به صندوق پستی
بقیه قسمت ها
```

…
}

Main:
Create mailbox; // ساخت صندوق مشترک بین فرایندها
Send(null, mailbox); قرار دادن یک پیام خالی در صندوق
Start Pi $i \in \{1,2,3, \dots\}$ // آغاز فرایندها

–

–