

سیستم‌های عامل

مؤلف: فرشید شیرافکن

دانشجوی دکتری بیوانفورماتیک دانشگاه تهران



ناشر: سازمان علمی آموزش فرادرس

بزرگترین پلتفرم آموزش آنلاین ایران

وب: www.faradars.org

تقدیم به:

روح پاک پدرم

- فرشید شیر افکن



سخن ناشر

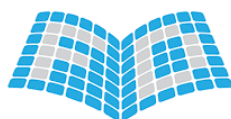
در عین تمام نقدهای وارد شده به کنکور، هنوز راه حلی عملی که در جمیع جوانب، بهتر از سبک کوتاه و چند گزینه‌ای سوالات باشد؛ ارائه نشده است. همین موضوع، کنکور را به ویژه کنکور کارشناسی ارشد به عنوان یک آزمون متمرکز و سراسری، از اهمیت دو چندانی برخوردار می‌کند.

یکی از آسیب‌های همراه با این آزمون سراسری این است که فضای رقابتی آن با ایجاد موسسات گوناگون، به سرعت از فضای یک رقابت علمی تبدیل به فضای رقابت اقتصادی می‌شود؛ به گونه‌ای که هزینه سرسام آور کلاس‌ها، دوره‌ها و منابع مرتبط با آزمون، از عهده بسیاری از دانشجویان خارج می‌شود. دانشجویانی که در عین استعداد تحصیلی بالا، در میدان رقابت مالی تحمیلی، در عین تمام شایستگی‌های خود، قدرت ادامه مسیر را از دست می‌دهند یا به نتیجه‌ای که در فضای مساوی مالی برای همه باید به آن می‌رسیدند، دست نمی‌یابند.

یکی از اهداف و آرمان‌های فرادرس به عنوان بزرگترین پروژه آموزش دانشگاهی اجرا شده بر بستر وب کشور، ایجاد دسترسی همگانی و یکسان به آموزش و دانش؛ مستقل از جغرافیا، زمان و سطح مالی دانشجویان بوده است. سیاست کاری فرادرس در راستای این آرمان، انتشار آموزش‌های ویدئویی تخصصی و دانشگاهی رایگان و یا بسیار کم هزینه، با تدریس مجرب‌ترین اساتید داخل و خارج کشور بوده است.

ما با انتشار رایگان این کتاب (به همراه نزدیک به ده کتاب رایگان دیگر) یکی از گام‌های دیگر خود را در راستای آرمان فرادرس برداشتیم. کتاب حاضر که حاصل نزدیک به دو دهه تدریس و پژوهش و تألیف مؤلف و مدرس فرادرس می‌باشد؛ در عین هزینه‌های بالای تألیف و آماده‌سازی، به جای انتشار و فروش؛ با تأمین مالی و سرمایه‌گذاری فرادرس به عنوان ناشر، به صورت کاملاً رایگان منتشر می‌شود. ما در گام‌های بعدی نیز تلاش خواهیم کرد تا هر جا بتوانیم، حتی شده یک کتاب مرجع دیگر و بیشتر را با پرداخت هزینه، آزادسازی کرده و به صورت رایگان منتشر کنیم.

مؤلفین و ناشرینی که تمایل به واگذاری حق انتشار کتاب خود به فرادرس را دارند، می‌توانند با ایمیل ebooks@faradars.org مکاتبه نمایند. ما این کتاب‌ها را با پرداخت هزینه تألیف به مدرس و ناشر، به صورت رایگان منتشر خواهیم کرد تا همه دانشجویان مستقل از سطح مالی، به منابع مفید آزمون دسترسی داشته باشند. همچنین اگر ایده و نظری در خصوص کتاب‌های رایگان فرادرس داشته باشید، خوشحال می‌شویم که آن را با ایمیل ebooks@faradars.org مطرح نمایید.



فرادرس

فراتر از یک کلاس درس
www.faradars.org

سازمان علمی آموزش فرادرس

بزرگترین پلتفرم آموزش آنلاین ایران

وب: www.faradars.org

منبع مطالعاتی تکمیلی مرتبط با این کتاب

آموزش ویدئویی سیستم‌های عامل

سیستم عامل یا سامانه عامل (Operating System) بدون شک مهمترین نرم افزار در کامپیوتر است. سیستم های عامل از اولین نرم افزارهایی است که پس از روشن کردن کامپیوتر مشاهده می شود، و همچنین آخرین نرم افزاری خواهد بود که قبل از خاموش کردن کامپیوتر مشاهده می شود.

سیستم عامل نرم افزاری است که مدیریت برنامه ها را به عهده گرفته و با کنترل، مدیریت و سازماندهی منابع سخت افزاری امکان استفاده بهینه و هدفمند آن ها را فراهم کرده و بستری را برای اجرای نرم افزارهای کاربردی فراهم می کند.

اگر کامپیوتری دارای عملیات محدود و ساده باشد به وجود سیستم عامل نیازی نخواهد بود. در این گونه سیستم ها اطلاعات ورودی و خروجی از طریق صفحه کلید، نمایشگر و ... در اختیار سیستم قرار می گیرد و لذا در این سیستم ها همواره یک برنامه در تمام حالت اجرا خواهد شد.

مدرس: مهندس فرشید شیرافکن

مدت زمان: ۱۱ ساعت

faradars.org/fvsft103

جهت مشاهده آموزش ویدئویی این آموزش [کلیک](#) کنید.

درباره مدرس

مهندس فرشید شیرافکن کارشناس ارشد مهندسی کامپیوتر گرایش نرم افزار است و در حال حاضر دانشجوی دکتری بیوانفورماتیک دانشگاه تهران هستند. ایشان از مدرسین نمونه در زمینه ارائه و آموزش دروس دانشگاهی انتخاب شده اند. ایشان مشاور کنکور هستند و بیش از ۳۰ کتاب در زمینه کنکور رشته کامپیوتر تألیف نموده اند. ایشان در حال حاضر به عنوان یکی از برترین مدرسین فرادرس از جهت کمیت و کیفیت دروس ارائه شده، نزدیک به ۲۰ عنوان درسی را در قالب آموزش ویدئویی از طریق فرادرس منتشر کرده اند. این مجموعه دروس تا کنون مورد استفاده دهها هزار دانشجوی سراسر کشور قرار گرفته اند.



مشاهده همه آموزش های تدریسی و تألیفی توسط مؤلف کتاب [کلیک](#) کنید.

Email: fshirafkan51@gmail.com

شماره همراه مدرس: ۰۹۱۲۱۹۷۲۰۲۸

<http://faradars.org/computer-engineering-exam>

دانلود رایگان مجموعه کتب کارشناسی و کنکور ارشد کامپیوتر

کتاب رایگان دیگر از این مجموعه آموزشی

۱. آموزش نظریه زبان ها و ماشین - [کلیک کنید \(+\)](#)
۲. آموزش برنامه‌نویسی سی پلاس پلاس - [کلیک کنید \(+\)](#)
۳. آموزش پایگاه داده ها - [کلیک کنید \(+\)](#)
۴. آموزش ساختمان داده ها - [کلیک کنید \(+\)](#)
۵. آموزش شیء‌گرایی در سی پلاس پلاس - [کلیک کنید \(+\)](#)
۶. آموزش ذخیره و بازیابی اطلاعات - [کلیک کنید \(+\)](#)

برای دانلود رایگان این مجموعه کتب، به لینک زیر مراجعه کنید:

<http://faradars.org/computer-engineering-exam>



دسته بندی موضوعی آموزش های فرادرس، در ادامه آمده است:

 <p>مهندسی برق الکترونیک و رباتیک</p>	 <p>هوش مصنوعی</p>	 <p>آموزش‌های دانشگاهی و تخصصی</p>	 <p>برنامه‌نویسی</p>
<p>مهندسی برق الکترونیک و رباتیک - کلیک (+)</p>	<p>هوش مصنوعی - کلیک (+)</p>	<p>آموزش‌های دانشگاهی و تخصصی - کلیک (+)</p>	<p>برنامه‌نویسی - کلیک (+)</p>
 <p>مهارت‌های دانشگاهی</p>	 <p>مباحث مشترک</p>	 <p>دروس دانشگاهی</p>	 <p>مهندسی نرم‌افزار</p>
<p>مهارت‌های دانشگاهی - کلیک (+)</p>	<p>مباحث مشترک - کلیک (+)</p>	<p>دروس دانشگاهی - کلیک (+)</p>	<p>مهندسی نرم‌افزار - کلیک (+)</p>
 <p>آموزش‌های عمومی</p>	 <p>طراحی و توسعه وب</p>	 <p>نرم‌افزارهای عمومی</p>	 <p>نرم‌افزارهای تخصصی</p>
<p>آموزش‌های عمومی - کلیک (+)</p>	<p>طراحی و توسعه وب - کلیک (+)</p>	<p>نرم‌افزارهای عمومی - کلیک (+)</p>	<p>نرم‌افزارهای تخصصی - کلیک (+)</p>

فهرست مطالب

فصل ۱ : مفاهیم اولیه

پردازنده
وقفه
فراخوانی های سیستم
حفاظت
سلسله مراتب حافظه
روشهای انتقال ورودی/خروجی
نگاه کلی به سیستم عامل
تاریخچه سیستم عامل
انواع سیستم عامل از نظر ساختاری
آزمون

فصل ۲ : فرآیند

فرایند و حالات آن
فرایند معلق
انواع زمانبندیها
نخ (thread)
پیاده سازی نخ (سطح کاربر، سطح هسته و ترکیبی)

فصل ۳ : زمان بندی پردازنده

معیارهای زمان بندی
الگوریتم های زمانبندی
الگوریتم FCFS
الگوریتم RR
الگوریتم SPN (SJF)
الگوریتم SRT
الگوریتم HRRN
الگوریتم FB
الگوریتم MLFQ
الگوریتم MLQ
زمان بندی اولویت (Priority)

زمان بندی FCFS

زمان بندی در سیستم چند پردازنده ای (LPT, RPT, SPT)

آزمون

فصل ۴ : همروندی

انحصار متقابل و همگام سازی

مباحث مطرح در ارتباط بین فرایندها

رویکردهای نرم افزاری انحصار متقابل

الگوریتم Decker (پنج تلاش دگر)

الگوریتم Peterson

رویکردهای انحصار متقابل با حمایت سخت افزار

راهکارهای سیستم عامل و زبان برنامه سازی برای تدارک همزمانی

سمافور

پیاده سازی انحصار متقابل توسط سمافور

همگام سازی با استفاده از سمافور

مسئله تولید کننده و مصرف کننده

مسئله غذا خوردن فیلسوف ها

مسئله خوانندگان و نویسندگان

مانیتور (ناظر)

مسئله تولید کننده و مصرف کننده با مانیتور

تبادل پیام

همگام سازی به کمک تبادل پیام

پیاده سازی انحصار متقابل توسط تبادل پیام

حل مسئله تولید کننده و مصرف کننده توسط تبادل پیام

آزمون

فصل ۵ : بن بست

شرایط بن بست
 گراف تخصیص منابع
 روش های رفع بن بست
 ترمیم
 روش های پیشگیری از بن بست
 روش های اجتناب از بن بست
 الگوریتم بانکداران
 خلاصه رویکردها
 آزمون

فصل ۶ : مدیریت حافظه

مدیریت حافظه ابتدایی
 جا به جایی و حفاظت
 مبادله
 الگوریتم های مکان یابی و تخصیص حافظه
 مدیریت حافظه با سیستم رفاقتی
 روی هم گذاری (Overlay)
 صفحه بندی (Paging)
 حافظه مجازی
 صفحه بندی درخواستی
 صفحه بندی چند سطحی
 جدول صفحه وارونه (معکوس)
 بافرهای کناری ترجمه (TLB)
 زمان موثر دسترسی
 آزمون
 الگوریتم‌های جایگزینی صفحه
 الگوریتم بهینه (optimal)
 الگوریتم NRU
 الگوریتم FIFO
 الگوریتم دومین شانس
 الگوریتم ساعت

الگوریتم LRU

پیاده سازی سخت افزاری LRU

شبیه سازی LRU در نرم افزار (الگوریتم سالمندی)

الگوریتم بافر کردن صفحه

نکات طراحی سیستم های صفحه بندی

پیش صفحه بندی (prepaging)

مدل مجموعه کاری (working sets)

الگوریتم فرکانس نقص صفحه (PFF)

تناقض بلیدی (Belady's anomaly)

الگوریتم های پشته (Stack Algorithms)

اندازه صفحه

ساختار برنامه

قطعه بندی

قطعه بندی درخواستی

قطعه بندی صفحه بندی (Segmentation with paging)

مقایسه روشهای مدیریت حافظه

آزمون

فصل ۷ : مدیریت I/O و دیسک

نرم افزار I/O

مدیریت دیسک

الگوریتم های زمانبندی بازوی دیسک (FCFS , SSTF , SCAN , CSCAN)

روشهای تخصیص فضای دیسک به فایل

سطوح در یک حافظه سه سطحی

آزمون

فصل ۱:

مفاهیم اولیه

سیستم عامل از منابع سخت افزاری پردازنده برای ارائه خدمات به کاربران استفاده می کند. بنابراین آشنایی با سخت افزار کامپیوتر، برای بررسی سیستم عامل ضروری است. اجزای سخت افزاری تشکیل دهنده کامپیوتر عبارتند از:

- ۱- پردازنده
- ۲- حافظه اصلی
- ۳- مولفه های ورودی و خروجی
- ۴- اتصالات داخلی سیستم.

پردازنده

پردازنده از واحد محاسبه و منطق (ALU)، واحد کنترل و رجیسترها (ثبات ها) تشکیل می شود و سه گام "واکشی، رمز گشایی و اجرا" را به طور مداوم انجام می دهد. پردازنده دائما در حال کار است و فقط در موارد خاصی به مدت کوتاهی به حالت Hold می رود و کنترل گذرگاه را به کنترل کننده DMA می دهد.

ثباتهای پردازنده

در داخل پردازنده مجموعه ای از ثباتها وجود دارد. این ثباتها سطحی از حافظه که سریعتر و کوچکتر از حافظه اصلی است را فراهم می کنند. ثباتهای داخل پردازنده عبارتند از:

۱- ثباتهای داده (AX, BX, CX, DX)

۲- ثبات شاخص (SI, DI)

۳- اشاره گر قطعه (CS, DS, SS, ES)

۴- اشاره گر پشته (SP)

۵- شمارنده برنامه (PC یا IP)

۶- ثبات دستورالعمل (IR)

تذکر: تمام پردازنده ها شامل یک یا مجموعه ای از ثباتها هستند به نام کلمه وضعیت (PSW) که حاوی اطلاعات وضعیت هستند. این ثبات، نوعا علاوه بر کدهای وضعیت، شامل اطلاعات دیگری، مثل بیت فعال/غیر فعال کردن وقفه و بیت حالت کاربر/سرپرست، نیز می باشد.

حالت‌های اجرای پردازنده

پردازنده دارای دو حالت اجرا (dual-mode) می باشد:

- ۱- مد کاربر: حالت کم امتیازتری که برنامه های کاربران در این حالت اجرا می شوند.
- ۲- مد هسته (سیستم، کنترل، کرنل): حالت ممتازتری که دستورالعملهایی مانند تغییر ثباتهای کنترل، I/O و مدیریت حافظه، در این حالت اجرا می شوند. در این حالت نرم افزار کنترل کامل پردازنده، دستورالعملها، ثباتها و حافظه را در اختیار دارد. این سطح از کنترل برای برنامه های کاربران غیرضروری، ناامن و نامطلوب است.

هدف اصلی از عملیات dual-mode، محافظت سیستم عامل از دیگر نرم افزارها می باشد.

فرایند یا پردازش (process)، یک برنامه در حال اجرا می باشد. تفاوت برنامه و فرایند در این است که برنامه یک نهاد غیرفعال و فرایند یک نهاد فعال می باشد. معمولاً برنامه بر روی دیسک به صورت یک فایل اجرایی دودویی ذخیره می شود. برنامه باید به حافظه لود شود و در داخل فرایندی قرار بگیرد تا اجرا شود. فرایند می تواند هم در حافظه اصلی و هم در حافظه جانبی قرار بگیرد.

تنظیم زمان سیستم در مد کرنل و خواندن ساعت سیستم در مد کاربر انجام می شود.

پردازنده به کمک بیتی که در PSW وجود دارد متوجه می شود که باید در کدام حالت (کاربر یا هسته) اجرا کند.

وقتی کاری برای اجرا موجود نباشد، پردازنده یک برنامه شامل یک حلقه انتظار مشغول را اجرا خواهد کرد.

عمل تغییر وضعیت از مد کرنل به مد کاربر و بلعکس از امکانات سخت افزاری است که برای کمک مستقیم به سیستم عامل طراحی شده است.

وقفه (interrupt)

وقفه سیگنالی است که روند عادی اجرا را تغییر می دهد. وقتی وقفه‌ای به CPU ارسال می شود، کار CPU متوقف شده و روال پاسخگو به وقفه (interrupt routine) اجرا می شود. بعد از پایان اجرای این روتین، CPU کار قبلی اش را ادامه می دهد. آدرس دستوری که هنگام اجرای آن وقفه صادر شده (آدرس برگشت) در پشته ذخیره می شود. بعد از پایان پاسخگویی به وقفه، آدرس برگشت در شمارنده برنامه (PC) قرار می گیرد و محاسباتی که اجرای آنها به تعویق افتاده از سر گرفته می شود.

- با وجود وقفه‌ها، پردازنده می‌تواند در هنگامی که عمل ورودی/خروجی در جریان است، مشغول اجرای دستور عملهای دیگر باشد.
- سخت افزار در هر زمان می‌تواند با ارسال سیگنالی به CPU، وقفه‌ای را صادر کند. نرم افزار نیز با اجرای عملیات خاصی به نام فراخوانی سیستم (System call)، می‌تواند وقفه‌ای را صادر کند.
- وقتی وقفه‌ای رخ می‌دهد، سخت افزار کنترل را به سیستم عامل بر گردانده و ثبات‌ها و شمارنده برنامه را ذخیره می‌کند. همچنین نوع وقفه را نیز تعیین می‌کند.
- بردار وقفه، جدولی از اشاره گرها می‌باشد که هر اشاره گر به یک روال وقفه اشاره می‌کند.

انواع وقفه‌ها

وقفه‌ها بر چهار نوع می‌باشند:

۱- برنامه

وقفه‌هایی که به دلیل بعضی شرایط حاصل از اجرای یک دستورالعمل بروز می‌کند:

- الف- سرریز شدن محاسباتی
- ب- تقسیم بر صفر
- ج- تلاش برای اجرای یک دستورالعمل ماشین غیرمجاز
- د- مراجعه به آدرس خارج از فضای مجاز کاربر

۲- زمان سنج

وقفه‌ای که توسط زمان سنج داخلی پردازنده تولید می‌شود. این وقفه به سیستم عامل اجازه می‌دهد، بعضی اعمال (مانند تست حافظه، چک کردن سخت افزار و یا تعیین زمان اجرای پردازنده در هر برش در سیستم استراک زمانی) را به طور مرتب انجام دهد.

۳- ورودی/خروجی

وقفه‌هایی که به وسیله کنترل کننده I/O تولید می‌شود، تا کامل شدن طبیعی یک عمل یا شرایط خطا را اعمال کند.

۴- نقص سخت افزار

وقفه‌هایی که با نقص سخت افزاری تولید می‌شود، مثل نقص برق یا خطای توازن حافظه.

پردازش وقفه

هنگامی که یک دستگاه I/O، یک عمل I/O را کامل می‌کند، دنباله حوادث زیر اتفاق می‌افتد:

- ۱- دستگاه یک علامت وقفه برای پردازنده می‌فرستد.
- ۲- پردازنده اجرای دستورالعمل جاری را قبل از پاسخ به این وقفه، به پایان می‌رساند.
- ۳- پردازنده بروز وقفه را بررسی کرده، در می‌یابد که وقفه ای آمده است و علامتی مبنی بر دریافت وقفه برای دستگاه وقفه دهنده می‌فرستد.
- ۴- پردازنده برای انتقال کنترل به روال خدماتی مربوط آماده می‌شود. در ابتدا اطلاعات مورد نیاز برای از سرگیری برنامه جاری را ذخیره می‌کند. حداقل اطلاعات، محتویات ثبات وضعیت برنامه (PSW) و محل دستورالعمل بعدی (PC) می‌باشد. این اطلاعات می‌تواند در بالای پشته کنترل سیستم گذاشته شود.
- ۵- پردازنده شمارنده برنامه (PC) را با آدرس شروع برنامه گرداننده وقفه ای که قرار است به این وقفه پاسخ دهد، بار می‌کند. در این حالت پردازنده به چرخه واکنشی رفته و کنترل به برنامه گرداننده وقفه منتقل می‌شود.
- ۶- در این لحظه محتوای PC و PSW مربوط به برنامه وقفه داده شده، در پشته سیستم ذخیره است.
- ۷- روال وقفه را پردازش می‌کند.
- ۸- بعد از کامل شدن پردازش وقفه، مقادیر ثباتها از پشته POP شده و در ثباتها گذاشته می‌شود.
- ۹- بار کردن مجدد مقادیر PSW و PC از پشته، که موجب اجرای دستورالعمل بعدی از برنامه وقفه داده شده می‌شود.

تعویض متن (Context Switch)

هنگامیکه وقفه ای رخ می‌دهد، قبل از اینکه سیستم عامل کنترل را به یک روال وقفه گیر بخصوص رد کند، وضعیت پردازش جاری را در محلی حفظ می‌کند، تا بتواند بعداً آنرا ادامه دهد و سپس به طرف روال وقفه گیر می‌رود که به این جریان تعویض متن می‌گویند.

فراخوانی های سیستم (System Calls)

فراخوانیهای سیستم واسطی بین فرایند و سیستم عامل فراهم می‌کند که به صورت دستورات زبان اسمبلی می‌باشند. در بعضی از سیستم‌ها، فراخوانی‌ها مستقیماً از برنامه‌های زبان سطح بالا ساخته شده و مانند زیر برنامه می‌باشند. وقتی برنامه ای اجرا می‌شود از فراخوانی‌های سیستمی به وفور استفاده می‌کند و کاربر جزئیات آن را نمی‌بیند.

انواع فراخوانی های سیستم عبارتند از:

- ۱- کنترل فرایند
- ۲- دستکاری فایلها
- ۳- دستکاری دستگاه
- ۴- دستکاری اطلاعات
- ۵- ارتباطات

وقتی یک وقفه، تله و یا فراخوانی سرپرست رخ می دهد، پردازنده در حالت هسته گذاشته شده و کنترل به سیستم عامل منتقل می گردد و یک تعویض حالت به یک روال سیستم عامل انجام شده، ولی چون اجرا در داخل فرایند جاری ادامه می یابد، تعویض فرایند انجام نمی شود.

حفاظت

مسئله حفاظت از سه دیدگاه مورد بررسی است:

۱- حفاظت از I/O

برای حفاظت از I/O می توان تمام دستورات I/O را به عنوان دستورات ممتاز در نظر گرفت تا کاربران فقط از طریق سیستم عامل بتوانند آن دستورات را اجرا کنند.

۲- حفاظت از حافظه

حفاظت حافظه را حداقل برای بردار وقفه و روال وقفه باید فراهم کرد. در واقع می خواهیم سیستم عامل را از دستیابی برنامه کاربر و همچنین برنامه های کاربر را از یکدیگر محافظت کنیم. یکی از روشهای ممکن استفاده از ثباتهای پایه و حد می باشد.

۳- حفاظت از CPU

باید کاری کرد که برنامه کاربر در حلقه گیر نکند و کنترل را به سیستم عامل برگرداند. برای این منظور از یک تایمر استفاده می کنیم.

سلسله مراتب حافظه

بین سه ویژگی کلیدی حافظه، یعنی "هزینه، ظرفیت و زمان دسترسی" باید سبک و سنگین کرد. برای این کار نمی توان بر یک حافظه یا فن آوری خاصی تکیه کرد و باید از سلسله مراتب حافظه استفاده کرد. یک سلسله مراتب متداول حافظه در زیر آورده شده است:

- ۱- ثباتها
- ۲- حافظه پنهان
- ۳- حافظه اصلی
- ۴- حافظه پنهان دیسک
- ۵- دیسک مغناطیسی
- ۶- رسانه جا به جایی پذیر

با حرکت به سطوح پایین تر این سلسله مراتب، شرایط زیر رخ می دهد:

- ۱- کاهش هزینه در هر بیت
 - ۲- افزایش ظرفیت
 - ۳- افزایش زمان دسترسی
 - ۴- کاهش تعداد دفعات دسترسی پردازنده به حافظه
- مدیریت منابع هر یک از حافظه های زیر در مقابل آن نوشته شده است:

ثبات: کامپایلر

حافظه پنهان: خودکار (سخت افزاری)

حافظه اصلی و دیسک: سیستم عامل

حافظه پنهان

حافظه پنهان (cache)، یک حافظه کوچک و سریع بین پردازنده و حافظه اصلی است. حافظه پنهان، حاوی بخشی از حافظه اصلی است. وقتی پردازنده می خواهد کلمه ای از حافظه را بخواند، وجود آن را در حافظه پنهان بررسی می کند. اگر وجود داشته باشد، به پردازنده تحویل می شود، در غیر اینصورت یک بلوک از حافظه اصلی، شامل تعداد ثابتی از خانه های حافظه، به حافظه پنهان منتقل شده و سپس کلمه مورد نظر به پردازنده تحویل داده می شود. هنگامی که یک بلوک از داده ها به حافظه پنهان آورده می شود تا یک مراجعه به حافظه انجام شود، به دلیل پدیده محلی بودن مراجعات، احتمالاً بزودی به دیگر کلمات آن بلوک نیز مراجعه خواهد شد.

روشهای انتقال ورودی/خروجی

برای عملیات I/O روشهای زیر وجود دارد:

۱- I/O برنامه سازی شده (سرکشی)

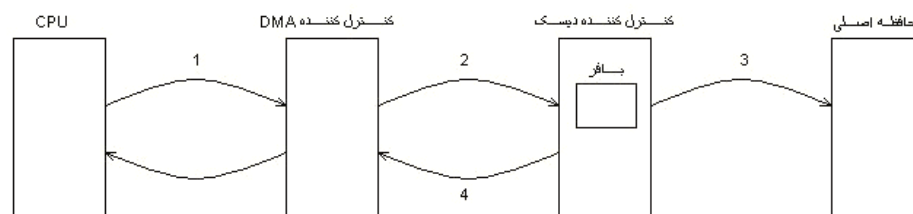
پردازنده یک فرمان I/O را از جانب فرایندی به یک مولفه I/O صادر می کند. سپس آن فرایند قبل از ادامه، تا کامل شدن عمل I/O، به انتظار مشغولی می گذراند.

۲- I/O مبتنی بر وقفه

پردازنده یک فرمان I/O را از جانب فرایند صادر می کند، سپس به اجرای دستورالعملهای بعدی ادامه می دهد و با کامل شدن عمل I/O، با وقفه مولفه I/O مواجه می شود.

۳- دسترسی مستقیم به حافظه (DMA)

مولفه DMA تبادل داده‌ها بین حافظه اصلی و مولفه I/O را کنترل می‌کند. پردازنده تقاضایی برای انتقال یک بلوک از داده‌ها را به مولفه DMA می‌فرستد و فقط پس از کنترل کل بلوک، مورد وقفه قرار می‌گیرد.



تذکر: در روش ورودی/خروجی مبتنی بر وقفه، دستورالعمل‌های بعدی می‌توانند از همان فرایند باشند، البته به شرطی که فرایند نیازی به انتظار برای تکمیل I/O نداشته باشد. در غیر اینصورت فرایند در انتظار وقفه معلق می‌گردد و کار دیگری انجام می‌گیرد.

روش دوم کارآمدتر از روش اول است.

برای انتقال I/O چند کلمه‌ای، روش DMA بسیار کارآمدتر از روش‌های اول و دوم است.

در روش اول، مولفه I/O به پردازنده وقفه نمی‌دهد و این مسئولیت پردازنده است که وضعیت مولفه I/O را متناوباً بررسی کند تا اتمام آن عمل را دریابد. بنابراین روش اول را می‌توان ورودی/خروجی بر اساس سرکشی (polling) نامید.

مشکل ورودی/خروجی برنامه‌سازی شده این است که پردازنده باید برای مدت طولانی منتظر بماند تا مولفه I/O برای دریافت یا ارسال آماده شود. در مدت انتظار، پردازنده به علت اینکه مکرراً وضعیت را سؤال می‌کند، کارایی سیستم به شدت پایین می‌آید.

روش Programed I/O و Interrupt I/O پردازنده اصلی را درگیر عملیات I/O می‌کند.

نگاهی کلی به سیستم عامل

سیستم عامل برنامه ای است که اجرای برنامه های کاربردی را کنترل و به صورت رابط کاربر و سخت افزار کامپیوتر عمل می کند. برای سیستم عامل انجام سه وظیفه را می توان در نظر گرفت:

۱- سهولت

سیستم عامل استفاده از کامپیوتر را برای کاربر ساده تر می کند.

۲- کارآمدی

سیستم عامل موجب استفاده کارآمد و بهینه از منابع سیستم کامپیوتری می شود.

۳- قابلیت رشد

سیستم عامل باید به نحوی ساخته شده باشد که توسعه آن میسر باشد.

سیستم عامل چیزی جز یک برنامه کامپیوتری نیست. این برنامه پردازنده را برای استفاده از سایر منابع سیستم هدایت می کند.

جایگاه سیستم عامل

می توان به سخت افزار و نرم افزاری که کاربردها را برای کاربر ارائه می کند به صورت لایه ای (شکل زیر) نگاه کرد. در این نگاه، سیستم عامل یک میانجی برای تسهیل دسترسی برنامه ساز و برنامه های کاربردی از امکانات و خدمات می باشد

برنامه های کاربردی
برنامه های سیستمی
سیستم عامل
سخت افزار

سخت افزار، پایین ترین سطح است که شامل سه قسمت است:

۱- دستگاه های فیزیکی (مانند سیم ها، منبع تغذیه، تراشه های مدار مجتمع)

۲- ریز معماری (شامل ثبات های درون CPU و یک مسیر داده شامل واحد محاسبه و منطق)

۳- زبان ماشین

تذکر: برنامه های سیستمی که در بالای سیستم عامل قرار دارند (مانند کامپایلرها، ادیتورها و ..)، بخشی از سیستم عامل نمی باشند، اگر چه توسط بعضی از سازندگان سیستم عامل ها عرضه می شوند.

سیستم عامل در زمینه های زیر خدمات خود را ارائه می دهد:

- ۱- ایجاد برنامه
- ۲- اجرای برنامه
- ۳- دسترسی به دستگاههای ورودی/خروجی
- ۴- کنترل دسترسی به پرونده ها
- ۵- دسترسی به سیستم عامل
- ۶- کشف و پاسخ به خطاها
- ۷- حسابداری.

سیستم عامل از طریق هسته (Kernel) ، با منابع سخت افزاری و نرم افزاری و از طریق پوسته (shell) با کاربران، ارتباط برقرار می کند.

بخشی از سیستم عامل که در حافظه اصلی قرار دارد را هسته سیستم عامل می گویند.

یک سیستم عامل ممکن است به دلایل زیر در طول زمان تغییر کند:

- ۱- ارتقاء و انواع جدید سخت افزار
- ۲- خدمات جدید
- ۳- رفع خطا

مولفه های سیستم عامل

سیستم بزرگی چون سیستم عامل را باید به مولفه های (components) کوچکتری تقسیم کرد. اکثر سیستم عامل ها دارای مولفه های زیر می باشند:

۱- مدیریت فرایند

برخورد با بن بست، ایجاد و حذف فرایندها، تعویق و از سرگیری فرایندها، هماهنگی فرایندها.

۲- مدیریت حافظه اصلی

تعیین بخشهای پر حافظه، تعیین فرایندی که باید لود شود، تخصیص حافظه و آزاد سازی حافظه.

۳- مدیریت حافظه ثانویه

مدیریت فضای آزاد، تخصیص حافظه، زمانبندی دیسک.

۴- مدیریت فایل

ایجاد و حذف فایلها و دایرکتوری ها، نگاشت فایلها در حافظه ثانویه و تهیه پشتیبان.

۵- مدیریت سیستم I/O

مدیریت بافرها، تخصیص کانالهای I/O و دستگاهها به فرایندها

تاریخچه سیستم‌های عامل

نسل اول: سیستم‌های ردیفی

در اولین کامپیوترها (حدوداً ۶۰ سال قبل)، برنامه ساز مستقیماً با سخت افزار در تراکنش بود و سیستم عاملی در کار نبود. این ماشینها از طریق چراغهای نمایش، کلیدها و نوعی دستگاه ورودی و چاپگر اجرا می شدند. مساله اصلی این سیستمها، زمانبندی و زمان نصب بود. چون در این عملیات، کاربران به صورت ردیفی و یکی پس از دیگری به کامپیوتر دسترسی داشتند، آن را پردازش ردیفی می خوانیم.

نسل دوم: سیستم‌های دسته‌ای ساده

زمانی که به خاطر زمانبندی و نصب در سیستمهای ردیفی به هدر می رفت قابل قبول نبود. برای افزایش استفاده از ماشین، مفهوم سیستم عامل دسته ای به وجود آمد. اولین سیستم عامل دسته ای در اوسط دهه ۱۹۵۰ به وجود آمد. ایده اصلی، استفاده از نرم افزاری به نام ناظر بود. با استفاده از سیستم عامل دسته ای، دیگر کاربر دسترسی مستقیم به ماشین ندارد و کار خود را روی کارت یا نوار به متصدی کامپیوتر می دهد. اپراتور کارها را به صورت ردیفی دسته کرده و همگی را روی یک دستگاه ورودی می گذارد تا مورد استفاده ناظر قرار گیرد. مساله زمانبندی و زمان تنظیم شرایط اولیه کارها به عهده ناظر است.

با هر کار، دستورالعملهایی از زبان کنترل کار (JCL) نیز وجود دارد. زبان کنترل کار، نوعی زبان برنامه نویسی برای فرمان دادن به ناظر می باشد.

JCL : Job Control Language

اسپولینگ

در سیستم‌های اولیه، CPU گرانترین جزء کامپیوتر بود و افزایش راندمان CPU مهمترین محرک در تکامل سیستم‌های عامل بوده است. برای همین منظور، تکنیک Spooling مطرح شد. اسپولینگ I/O یک کار را با محاسبات کار دیگر به طور همزمان انجام می دهد. حتی می تواند در حین خواندن ورودی‌های یک کار، خروجی‌های کار دیگر را انجام دهد. اسپولینگ موجب می شود تا CPU و دستگاههای I/O با سرعت بیشتری کار کنند. استفاده از spooling در سیستم عامل دسته ای باعث افزایش سرعت می شود.

سیستم‌های دسته ای Offline Spooling

در این سیستم‌ها، کارها توسط دستگاه کارت خوان یک کامپیوتر آهسته و ارزان خوانده می شد و به کمک یک نوار گردان بر روی یک نوار مغناطیسی ذخیره می شدند. بعد از تشکیل یک دسته از کارها، نوار توسط اپراتور به CPU اصلی منتقل شده تا اجرا شروع شود. نتیجه اجرا بر روی نوار دیگری نوشته می شد. بعد از

اجرای تمام دسته کارها، اپراتور نوار خروجی را به یک پردازنده آهسته منتقل می‌کرد تا عمل چاپ انجام شود. در این روش ارتباط پردازنده اصلی محاسباتی با کارت خوان و چاپگر، غیر مستقیم بود.

فواید سیستم‌های دسته ای **offline Spooling** نسبت به سیستم‌های قبلی

- ۱- افزایش راندمان پردازنده گران قیمت
- ۲- عملیات ساده‌تر
- ۳- ساده شدن استفاده از سیستم از راه دور.

معایب سیستم‌های دسته ای **Offline Spooling**

- ۱- تاخیر بین زمان تحویل کار و زمان تکمیل کار
- ۲- نیاز به سخت افزار اضافی
- ۳- عدم وجود اولویت
- ۴- استفاده زیاد از دستگاههای جانبی

سیستم‌های دسته ای **Online Spooling**

با ظهور دیسک‌های سخت، سیستم‌های دسته ای متحول شدند. در این سیستم‌ها بلافاصله بعد از ورود کارها، کارت‌ها خوانده شده و به دیسک منتقل می‌شدند. زمانی که اجرای یک کار تمام می‌شود، سیستم عامل با سرعت بیشتری یک کار جدید را از روی دیسک برداشته و به حافظه اصلی لود کرده و سپس آن را اجرا می‌کند.

در خروجی نیز از اسپولینگ استفاده می‌شود. اسناد خروجی ابتدا بر روی دیسک ذخیره شده و زمانی که خروجی قبلی چاپگر تمام شد، این اسناد از دیسک به بافر چاپگر منتقل می‌شوند. در این تکنیک که **Spooling** (عملیات پیوسته، مستقیم و همزمان دستگاه‌های جانبی) نامیده می‌شود، ارتباط CPU با دستگاه‌های جانبی مستقیم است و نیازی به پردازنده‌های آهسته و نوار گردان‌های اضافی و حمل نوارها توسط اپراتورها نمی‌باشد.

SPOOL : Simultaneous Peripheral Operation On-Line

مزایای اسپولینگ **online** نسبت به **offline** عبارتند از :

- ۱- دسترسی با اولویت
- ۲- گردش سریع کارها
- ۳- بالا بودن راندمان CPU و دستگاههای I/O
- ۴- امکان داشتن همزمان چند مدرک ورودی/خروجی

از اسپولینگ برای پردازش داده‌های راه دور می‌توان استفاده کرد. CPU داده‌ها را از طریق مسیرهای ارتباطی به چاپگر راه دور می‌فرستد یا ورودی‌ها را از یک کارت خوان راه دور دریافت می‌کند. در پردازش راه دور CPU دخالت ندارد و فقط مواظب است که چه هنگام کار تمام شود تا دسته دیگری از کارها را جمع‌آوری کند.

نرم افزار spooler کارها را دسته بندی می کند. این نرم افزار اجزاء یک کار(داده، برنامه و JCL) را از هم جدا می کند. سپس JCL ها دستور به دستور اجرا شده و توسط سیستم اسپولینگ روی دیسک اصلی قرار می گیرند و جدولی به نام ISPT (Input Spool Table) ساخته می شود. هر سطر این جدول در برگیرنده اطلاعات کنترلی یک Job می باشد.

نسل سوم: سیستم های دسته‌ای چند برنامه‌گی

حتی با ردیف کردن خودکار کارها به وسیله سیستم عامل ساده دسته ای، به علت کند بودن دستگاههای I/O در مقایسه با پردازنده، پردازنده اکثرا بی کار است. تمام کارهایی که وارد سیستم می‌شوند، در انبار کار (job pool) نگهداری می‌شوند. این کارها شامل فرایندهایی می‌باشند که بر روی دیسک قرار دارند و منتظر تخصیص حافظه می‌باشند. اگر چند کار آماده ورود به حافظه وجود داشته باشند، سیستم یکی را انتخاب کرده (زمانبندی کار) و آنرا به حافظه می‌آورد تا اجرا کند. همچنین اگر چند کار به طور همزمان آماده اجرا باشند، سیستم یکی از آنها را انتخاب می‌کند و CPU را به آن تخصیص می‌دهد. (زمانبندی CPU). ایده چند برنامه‌ای به این صورت است که سیستم عامل در هر زمان چند کار را در حافظه نگه می‌دارد و یکی از کارهای موجود در حافظه را انتخاب کرده و اجرای آن را شروع می‌کند. اگر این کار منتظر عمل دیگری مانند آماده شدن نوار یا تکمیل یک عمل I/O باشد، CPU به اجرای کار دیگری مشغول می‌شود و اگر آن کار نیز نیاز به انتظار داشته باشد، به کار دیگری می‌پردازد و این روند ادامه می‌یابد. با پایان مدت انتظار کار اول، دوباره CPU در اختیار آن قرار می‌گیرد. این عملیات باعث می‌شود که CPU بیکار نماند.

سیستم های دسته‌ای برای اجرای برنامه‌های بزرگ که نیاز به محاوره کمی دارند، مناسب می‌باشند.

مشکلات سیستم دسته‌ای از نظر کاربر

- ۱- کاربر برای به دست آوردن نتایج مجبور است از کارتهای کنترلی استفاده کند.
- ۲- مراجعه بعدی در کارهای چند مرحله‌ای ممکن است به نتایج مراحل قبلی وابسته باشد.
- ۳- برنامه نویس نمی‌تواند برنامه در حال اجرا را تغییر دهد. (دیباگ برنامه‌ها ایستا می‌باشد)

ممکن است حافظه را برای نگهداری ۳،۴ یا تعداد بیشتری از برنامه‌ها گسترش دهیم و پردازنده به تمام آنها پردازد. این عمل را چند برنامه‌ای یا چند وظیفه‌ای می‌گویند که موضوع اصلی سیستم‌های عامل امروزی است.

نسل چهارم : سیستم‌های اشتراک زمانی (Time sharing)

سیستم‌های اشتراک زمانی (چندبرنامگی تعاملی) توسعه منطقی سیستم‌های چند برنامه‌ای می‌باشند. در سیستم عامل اشتراک زمانی، وقت پردازنده بین کارها به اشتراک گذاشته می‌شود. برای اجرای چند کار، پردازنده بین آنها سوئیچ می‌کند و کاربران می‌توانند با برنامه‌های در حال اجرا تعامل داشته باشند. در سیستم‌های اشتراک زمانی ارتباط کاربر با سیستم به صورت محاوره‌ای (interactive یا online) است. سیستم‌های اشتراک زمانی، هزینه استفاده محاوره‌ای از سیستم‌های کامپیوتری را معقول کرده‌اند و با استفاده از زمانبندی CPU و چند برنامه‌ای، به هر کاربر زمان محدودی را تخصیص داده و هر کاربر حداکثر یک برنامه در حافظه دارد.

در سیستم‌های اشتراک زمانی، چون سیستم به سرعت از برنامه‌ای به برنامه دیگر می‌رود، هر کاربر فکر می‌کند که یک کامپیوتر اختصاصی در اختیار اوست، در حالی که یک کامپیوتر بین همه کاربران مشترک است.

خصوصیات سیستم‌های اشتراک زمانی عبارتند از:

- ۱- پیچیده‌تر از چند برنامه‌ای هستند.
- ۲- برای مواردی که نیاز به زمان پاسخ کوتاه است، استفاده می‌شوند.
- ۳- امکان چند برنامه‌ای را فراهم می‌سازد.
- ۴- وجود یک سیستم فایل ضروری است.
- ۵- در main frame قابلیت چند کاربره و چند برنامه‌ای را فراهم می‌کند.

چندبرنامه‌ای دسته‌ای و اشتراک زمانی هر دو از چندبرنامگی استفاده می‌کنند.

تفاوت‌های اصلی چندبرنامه‌ای دسته‌ای با اشتراک زمانی

اشتراک زمانی	چندبرنامه‌ای دسته‌ای	
حداقل زمان پاسخ	حداکثر استفاده از پردازنده	هدف اصلی
فرمانهایی که از پایانه وارد می‌شود.	دستورالعمل‌های JCL که همراه کار ارائه شده است.	منبع دستورات به سیستم عامل

تذکر: سیستم‌های زیر در کتاب سیلبرشاتس آورده شده است:

۱- سیستم‌های موازی (یا چند پردازنده ای)

۲- سیستم‌های عامل توزیع شده (گسترده)

۳- سیستم‌های عامل بی‌درنگ (Real time)

سیستم‌های موازی (یا چند پردازنده ای)

سیستم‌هایی که بیش از یک پردازنده در آنها وجود دارد را چند پردازنده ای (Multi Processor) می‌نامند. پردازنده‌ها در این سیستم با یکدیگر ارتباط نزدیکی دارند و از گذرگاه آدرس، ساعت و گاهی حافظه و دستگاه‌های جانبی به طور اشتراکی استفاده می‌کنند. این سیستم‌ها را اتصال محکم (tightly coupled) نیز می‌نامند.

انواع سیستم‌های چند پردازنده‌ای

الف - چند پردازنده‌ای متقارن

در سیستم چند پردازنده‌ای متقارن (Symmetric)، هر پردازنده از کپی یکسانی از سیستم عامل استفاده می‌کند که این کپی‌ها در صورت لزوم با یکدیگر ارتباط برقرار می‌کنند.

ب - چند پردازنده‌ای نامتقارن

در سیستم چند پردازنده‌ای نامتقارن، هر پردازنده کار خاصی را انجام می‌دهد. کنترل سیستم به عهده پردازنده اصلی می‌باشد و پردازنده‌های دیگر منتظر دستور پردازنده اصلی هستند یا کار از قبل تعیین شده‌ای دارند. این طرح رئیس / مرئوس (master/slave) را بیان می‌کند، که پردازنده اصلی (master)، کارها را برای پردازنده‌های دیگر (slave)، زمانبندی کرده و به آنها تخصیص می‌دهد.

مزایای سیستم متقارن نسبت به نامتقارن عبارتند از :

۱- قابل حمل بودن روی سیستم‌های سخت افزاری مختلف

۲- تعادل بار سیستم به علت اجرای سیستم عامل روی چند پردازنده

دلایل ساخت سیستم‌های چند پردازنده‌ای

۱- افزایش توان عملیاتی (throughput) : تعداد کارهای انجام شده در یک واحد زمانی.

۲- افزایش قابلیت اعتماد

(خرابی یک پردازنده منجر به از کار افتادن سیستم نمی‌شود بلکه از سرعت آن می‌کاهد).

۳- مقرون به صرفه بودن از نظر اقتصادی

سیستم‌های عامل توزیع شده (گسترده)

در این سیستم‌ها، محاسبات بین چند پردازنده توزیع می‌شود. هر پردازنده، حافظه و ساعت مخصوص به خود را دارد و از طریق خطوط ارتباطی با یکدیگر مرتبط‌اند. همچنین پردازنده‌ها از نظر اندازه و عملکرد با

یکدیگر فرق دارند. سیستم‌های توزیعی را سیستم‌های ارتباط ضعیف (Loselycoupled) نیز می‌گویند.

دلایل ساخت سیستم‌های توزیعی

الف- اشتراک منابع : کاربری در یک سایت می‌تواند از چاپگری در سایت دیگر استفاده کند.

ب- قابلیت اعتماد: خرابی یک کامپیوتر، دیگری را تحت تاثیر قرار نمی‌دهد.

ج- افزایش سرعت محاسبات: توزیع یک محاسبه در بین چند سایت

د- ارتباطات

سیستم‌های عامل بی‌درنگ (Real time)

سیستم عامل بی‌درنگ (بلادرنگ)، نوعی از سیستم‌های عامل‌های همه منظوره می‌باشد و در صورتی به کار گرفته می‌شود که برای عملکرد یک پردازنده نیاز به زمان دقیقی باشد. یک سیستم بی‌درنگ وقتی درست کار می‌کند که در محدودیت زمانی مشخص، نتایج مورد انتظار را تولید کند. یعنی پردازش باید در محدودیت زمانی خاص انجام شود وگرنه سیستم از کار می‌افتد.

سیستم‌های نظامی، تزریق سوخت اتمی، کنترل کننده‌های لوازم خانگی، کنترل صنعتی و تصویرسازی پزشکی نمونه‌هایی از سیستم‌های بی‌درنگ می‌باشند. سیستم‌های بی‌درنگ بر دو نوع نرم و سخت می‌باشند.

ویژگی‌های سیستم‌های بی‌درنگ نرم (Soft Real Time)

- ۱- دارای محدودیت زمانی نسبتاً دقیقی می‌باشند.
- ۲- در پروژه‌های علمی پیشرفته استفاده می‌شوند.
- ۳- با سیستم‌های دیگر قابل ترکیب می‌باشند.
- ۴- مهلت زمانی را پشتیبانی نمی‌کنند. (از آنها در کنترل صنعتی و روبات‌ها استفاده نمی‌شود).

ویژگی‌های سیستم‌های بی‌درنگ سخت (Hard Real Time)

- ۱- دارای محدودیت زمانی دقیقی می‌باشند و کارهای بحرانی به موقع انجام می‌شوند.
- ۲- تمام تاخیرهای موجود در سیستم باید از بین بروند.
- ۳- از ویژگیهای پیشرفته سیستم عامل استفاده نمی‌شود.
- ۴- با سیستم‌های اشتراک زمانی برخورد (Conflict) دارند و با یکدیگر ترکیب نمی‌شوند.
- ۵- هیچکدام از سیستم‌های همه منظوره موجود از عملکرد بی‌درنگ سخت پشتیبانی نمی‌کنند.

انواع سیستم عامل از نظر ساختاری

انواع سیستم عامل از نظر ساختاری عبارتند از:

- ۱- یکپارچه (Monolithic)
- ۲- لایه ای (Layered)
- ۳- ماشین مجازی (Virtual Machine)
- ۴- مشتری-خدمتگزار (Client /Server)

ساختار یکپارچه

ساده ترین ساختار برای سیستم عامل است که در DOS از آن استفاده می شود. در این ساختار واسط ها و سطوح عملکرد به خوبی از هم تفکیک نشده اند و برنامه های کاربردی می توانند به روالهای I/O دستیابی داشته باشند و مستقیماً بر روی مونیتر یا دیسک بنویسند.

ساختار لایه ای

با اعمال خاصیت پیمانه ای بودن به سیستم عامل، سیستم عامل می تواند کنترل بیشتری بر روی کامپیوتر و برنامه های کاربردی داشته باشد. یکی از بهترین روشها برای این کار، روش لایه ای است که سیستم عامل را به تعدادی لایه تقسیم می کند و هر کدام بر روی یکدیگر قرار می گیرند. لایه پایینی (شماره صفر)، سخت افزار است و لایه بالایی واسط کاربر است. طراحی و پیاده سازی ساختار لایه ای ساده است.

ویژگی های ساختار لایه ای

- ۱- هر لایه فقط از توابع و خدمات لایه های پایین تر استفاده می کند که باعث ساده شدن خطایابی می شود.
- ۲- خطایابی از لایه پایین شروع می شود و سپس لایه بعدی خطایابی می شود. اگر در حین خطایابی لایه ای، خطایی یافت شود، می دانیم که خطا در همان لایه است، زیرا لایه های پایینی قبلاً خطایابی شده اند.
- ۳- هر لایه می تواند عملیات، ساختمان داده ها و سخت افزار را از لایه های بالاتر مخفی کند.

مشکلات روش لایه ای عبارتند از:

- ۱- هر لایه نیاز به برنامه ریزی دقیقی دارد، چون فقط می تواند از لایه های زیرین استفاده کند.
- ۲- پایین بودن کارایی نسبت به روشهای دیگر. (تولید سربار زیاد)

مزایای ساختار لایه ای نسبت به یکپارچه عبارتند از:

- ۱- قابلیت گسترش بیشتر
- ۲- خطایابی ساده تر
- ۳- مدیریت ساده تر

ساختار ماشین مجازی (Virtual Machine)

یک سیستم کامپیوتری دارای لایه های سخت افزار، هسته و برنامه های سیستم می باشد. برنامه های سیستم که در بالای هسته قرار دارند، می توانند از فراخوانیهای سیستم و دستورات سخت افزاری استفاده کنند. بعضی سیستمها از این الگو تبعیت می کنند و حتی برنامه های سیستم می توانند از طریق برنامه های کاربردی فراخوانی شوند و برنامه های سیستم می توانند آنچه را که در زیر آنها قرار دارد را مشاهده نمایند، به طوری که گویی برنامه های کاربردی بخشی از خود ماشین هستند. این شیوه لایه ای را ماشین مجازی می نامند که یک کپی از کامپیوتر در اختیار هر فرایند قرار می دهد. اجرا شدن سیستم عامل DOS تحت محیط ویندوز با استفاده از ایده ماشین مجازی انجام گرفته است.

مزایای ساختار ماشین مجازی

- ۱- امکان اجرای هم زمان چند سیستم عامل مختلف.
- ۲- شبیه سازی دستورات ماشینی که کامپیوتر فاقد آن است.
- ۳- امکان برنامه نویسی به زبان های غیر اسمبلی.
- ۴- تست سیستم عامل جدید.

با تغییراتی در روش ماشین مجازی، ساختاری به نام **Exokernels** به وجود آمده است. در این ساختار نیازی به لایه نگاشت نمی باشد و همچنین برنامه ای در پایین ترین لایه در مد هسته اجرا شده که منابع را به ماشین های مجازی تخصیص داده و بررسی می کند که ماشینی از منابع ماشین دیگر استفاده نکند.

ساختار مشتری - خدمتگزار (Client - Server)

در این ساختار، اکثر وظایف سیستم عامل در سطح کاربر انجام می شود و هسته از طریق پیام ها بین مشتری/خدمتگزار ارتباط برقرار می سازد. ایده طراحی این ساختار، کمینه کردن هسته و انتقال کدها به لایه های بالاتر می باشد. مزایای ساختارهای مشتری - خدمتگزار عبارتند از:

- ۱- طراحی ساده تر سیستم عامل
- ۲- استفاده در سیستم های توزیعی
- ۳- عدم خرابی کل سیستم در صورت خرابی یک سرور

کنکور ارشد

(مهندسی کامپیوتر - دولتی ۸۶)

۱- کدام یک از دستور العمل های زیر، فقط قادر به اجرا در مد کرنل (Kernel Mode) است؟

- (۱) خواندن ساعت سیستم
(۲) خواندن PSW
(۳) تنظیم زمان سیستم
(۴) نوشتن در ثبات دستور العمل

پاسخ: گزینه ۳ درست است. ■

(مهندسی IT - دولتی ۸۶)

۲- مدیریت منابع حافظه ای زیر با کدام عامل است؟

- ۱- ثبات ها ۲- حافظه پنهان (cache) ۳- حافظه اصلی ۴- فضای دیسک

(تذکر: منظور از خودکار، سخت افزاری است)

- | | | | |
|----------------|---------------|---------------|-------------------------|
| ۱) ۱- کاربر | ۲- سیستم عامل | ۳- سیستم اصلی | ۴- سیستم عامل |
| ۲) ۱- کامپایلر | ۲- خودکار | ۳- سیستم عامل | ۴- سیستم عامل |
| ۳) ۱- کاربر | ۲- سیستم عامل | ۳- کامپایلر | ۴- خودکار |
| ۴) ۱- خودکار | ۲- خودکار | ۳- سیستم عامل | ۴- سیستم عامل یا خودکار |

پاسخ: گزینه ۲ درست است.

حافظه اصلی و فضای دیسک توسط سیستم عامل مدیریت می شود.

(مهندسی IT - دولتی ۸۸)

۳- فرض کنید زمان محاسبات یک فرآیند 200 سیکل CPU باشد. در ضمن عملیات I/O در حال انجام برای یک فرآیند دیگر از طریق DMA در حال انجام بوده و پس از 100 سیکل CPU، پایان عملیات I/O توسط یک وقفه به سیستم اطلاع داده شود. اگر زمان اجرای ISR را 10 سیکل CPU فرض کنیم، کل عملیات مذکور چه مقدار از زمان سیستم را به خود

اختصاص می دهند؟ (زمان هر سیکل CPU را معادل یک واحد زمانی فرض کنید.)

- (۱) 210 واحد زمانی
(۲) کمتر از 210 واحد زمانی
(۳) 310 واحد زمانی
(۴) بیشتر از 210 واحد زمانی

پاسخ: گزینه ۴ درست است.

فرآیند اول برای انجام محاسباتش به اندازه 200 سیکل به CPU نیاز دارد و فرآیند دوم معادل 100 سیکل مشغول انجام عملیات I/O می باشد و چون از DMA استفاده می شود، این دو عمل همزمان انجام شده و به 200 سیکل نیاز است. همچنین 10 سیکل نیز صرف اجرای ISR فرآیند دوم می شود. البته در هنگام کار با DMA و انجام I/O، گاهی گذرگاه در اختیار DMA قرار می گیرد و زمان اجرای فرآیند اول بیشتر از 200 سیکل خواهد شد. بنابراین کل عملیات به بیش از 210 سیکل نیاز دارد. ■

(مهندسی کامپیوتر - آزاد ۷۹)

۴- جدول زیر زمان های لازم برای ورود، محاسبه و خروج 3 کار را در یک سیستم دسته ای با Spooling نشان می دهد. حداقل کل زمان مصرفی برای اجرای هر 3 کار به شرط آنکه ترتیب ورود کارها تعیین کننده ترتیب پردازش و ترتیب خروجی آن ها باشد، چقدر است؟

	زمان ورود	زمان پردازش	زمان خروج
Job1	5	4	1
Job2	2	2	3
Job3	5	3	2

17 (۴)

20 (۳)

10 (۲)

27 (۱)

پاسخ: گزینه ۴ درست است.

در سیستم دسته ای که از تکنیک Online Spooling استفاده می کند، امکان پردازش یک کار، همزمان با I/O سایر کارها وجود دارد. طبق جدول، ورود Job1، 5 واحد زمانی طول می کشد. در این زمان پردازشی انجام نمی شود. با پایان ورود Job1، پردازش آن در لحظه 5 شروع شده و تا لحظه 9 ادامه می یابد. در زمان 5 تا 7 نیز Job2 از ورودی دریافت می شود. اجرای Job1 در لحظه 9 تمام شده و به خروجی فرستاده می شود و ... بنابراین داریم:

	ورود کارها	پردازش	خروج
Job1	0..5	5..9	9..10
Job2	5..7	9..11	11..14
Job3	7..12	12..15	15..17

در لحظه 11 تا 12 پردازشی انجام نمی شود، چون اجرای Job2 تمام شده و ورود Job3 کامل نشده است. در لحظه 10 تا 11 خروجی نداریم، چون خروجی Job1 تمام شده ولی پردازش Job2 تمام نشده است. در لحظه 14 تا 15 خروجی نداریم، چون خروجی Job2 تمام شده ولی پردازش Job3 تمام نشده است. در لحظه 17 آخرین کار خارج می شود.

(مهندسی کامپیوتر - آزاد ۷۱)

۵- در یک سیستم عامل گسترده (Distributed Operating System)، کدام یک از موارد زیر درست نیست؟

(۱) چندین CPU مستقل از نظر جغرافیایی با هم فاصله دارند و تحت یک سیستم عامل کار می کنند.

(۲) در تبادل پیغام کاربران می بایست آدرس ماشین های یکدیگر را بدانند.

(۳) محل استقرار فایل ها در کنترل کاربران نمی باشد.

(۴) قابلیت اطمینان یک سیستم عامل گسترده از یک سیستم عامل متمرکز بیش تر است.

پاسخ: گزینه ۲ جواب است.

در سیستم عامل های توزیع شده، می توان منابع را با نام آنها (بدون نیاز به دانستن آدرس آن ها)، فراخوانی کرد. به این خاصیت "شفافیت" می گویند.



(مهندسی IT - آزاد ۹۰)

۶- امکان انتقال یک کلمه توسط واحد DMA در کدام نقطه از یک چرخه دستورالعمل وجود ندارد؟

(۱) پس از اجرای دستورالعمل

(۲) پس از کدگشایی دستورالعمل

(۳) پس از واکنشی عملوندها

(۴) پس از ذخیره نتایج

پاسخ: گزینه ۳ جواب است.

امکان انتقال یک کلمه توسط واحد DMA پس از واکنشی عملوندها وجود ندارد.



فصل ۲: فرآیند

در این فصل مفهوم فرآیند و حالات فرآیند بررسی می‌شوند. همچنین نخ نیز توضیح داده می‌شود.

فرآیند و حالات آن

به برنامه در حال اجرا، فرآیند (process) می‌گویند. یک فرآیند می‌تواند در یکی از حالت‌های زیر باشد:

۱- آماده (READY): فرآیندی که وقتی به آن فرصت داده شود، برای اجرا آماده باشد.

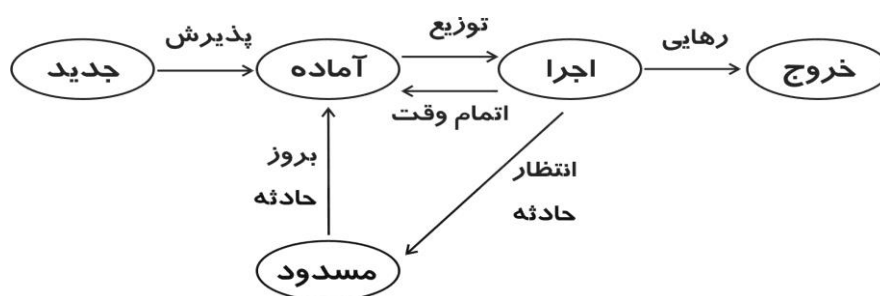
۲- اجرا (Running): فرآیندی که هم اکنون در حال اجر می‌باشد.

۳- مسدود (Blocked): فرآیندی که تا بروز حادثه‌ای (مثل اتمام یک عمل I/O)، نمی‌تواند اجرا شود.

۴- جدید (New): فرآیندی که هم اکنون گرفته شده است، اما هنوز جزء فرآیندهای قابل اجرای سیستم عامل پذیرفته نشده باشد.

۵- خروج (Terminated): فرآیندی که اجرای آن پایان یافته است و یا اجرای آن قطع شده و از مجموعه فرآیندهای قابل اجرای سیستم عامل خارج شده است.
تذکر: به حالت مسدود، حالت بسته یا انتظار نیز می‌گویند.

در شکل زیر این حالات، نشان داده شده است:



تغییر حالات ممکن

۱- جدید به آماده

اگر سیستم عامل آمادگی گرفتن یک فرآیند دیگر را داشته باشد، فرآیند موجود در حالت جدید را به حالت آماده می‌برد.

۲- آماده به اجرا

سیستم عامل یکی از فرایندهای موجود در حالت آماده که وقت اجرای آن فرا رسیده است را انتخاب کرده و از حالت آماده به اجرا می‌برد. به این عمل توزیع (Dispatch) می‌گویند.

۳- اجرا به خروج

وقتی که فرایند جاری اعلام پایان کند، سیستم عامل آن را از حالت اجرا به خروج می‌برد.

۴- اجرا به مسدود

وقتی فرایندی چیزی را خواهد که به خاطر آن باید منتظر بماند، سیستم عامل آن فرایند را از حالت اجرا به مسدود می‌برد. به این عمل بلوکه شدن می‌گویند. مثلاً فرایندی یک عمل I/O را شروع کند که قبل از تکمیل نیاز به بخش مشترکی از حافظه مجازی را داشته باشد که فوراً فراهم نباشد یا منتظر دریافت ورودی از یک فرایند دیگر باشد.

۵- مسدود به آماده

وقتی حادثه‌ای که فرایند منتظر آن بوده است رخ دهد، از حالت مسدود به آماده می‌رود. به این عمل wake up می‌گویند.

۶- اجرا به آماده

متداول ترین دلیل انتقال یک فرایند از حالت اجرا به آماده، اتمام زمان مجاز برای اجرای فرایند جاری در سیستم عامل‌های چند برنامه‌ای می‌باشد.

۷- آماده به خروج

در بعضی سیستم‌ها، یک پدر می‌تواند هر لحظه که بخواهد، فرایند فرزند خود را پایان دهد و یا با پایان یافتن فرایند پدر، ممکن است همه فرایندهای فرزند آن نیز پایان یابند.

۸- مسدود به خروج

مانند توضیحات آماده به خروج است.

۹- تهی به آماده

فرایند جدیدی را برای اجرای یک برنامه ایجاد می‌کند.

وجود وضعیت مسدود باعث افزایش بهره‌وری پردازنده می‌شود. چون وقتی فرایند در حال اجرا نیاز به I/O پیدا می‌کند، به حالت Blocking منتقل شده و فرایند آماده دیگری به قسمت اجرا منتقل می‌شود، تا در حد امکان CPU بیکار نماند.

دسته بندی فرایندها

بطور کلی فرایندها به دو دسته تقسیم می‌شوند:

۱- محدود به CPU (CPU Limited)

بیشتر زمان کامپیوتر صرف محاسبات CPU می‌شود.

۲- محدود به ورودی / خروجی (I/O Limited)

بیشتر زمان کامپیوتر صرف ورود داده‌ها و خروج اطلاعات می‌شود.

دلایل ایجاد یک فرایند جدید

۱- برقراری ارتباط محاوره‌ای

کاربر از طریق پایانه با سیستم ارتباط برقرار می‌کند.

۲- ارائه سرویس به وسیله سیستم عامل

سیستم عامل می‌تواند فرایندی را برای ارائه خدمتی از طرف برنامه کاربر ایجاد نماید، بدون اینکه کاربر ناچار به انتظار باشد.

۳- زایش توسط یک فرایند موجود

به منظور بهره‌گیری از توازی با تفکیک، برنامه کاربر می‌تواند ایجاد فرایندهای جدیدی را دیکته کند.

۴- کار دسته‌ای جدید

سیستم عامل با جریانی از کارهای دسته‌ای روبه‌رو می‌باشد. وقتی برای گرفتن یک کار جدید آماده است، دنباله بعدی از فرمانهای کنترل کار را می‌خواند.

عملیات ایجاد فرایند

۱- تخصیص یک شناسه یکتا به فرایند جدید ۲- تخصیص فضا برای فرایند

۳- مقدار دهی اولیه PCB ۴- برقراری پیوندهای لازم

۵- ایجاد و گسترش ساختمان داده‌های ممکن دیگر.

دلایل پایان یک فرایند

فراخوانی یک سرویس سیستم عامل برای بیان تکمیل اجرایش	پایان طبیعی
در فرایند محاوره ای، مقدار زمانی که از آخرین ورودی کاربر گذشته است.	سقف زمانی
انتظار زیادتر از حد برای بروز یک حادثه مشخص	گذشت زمان
نیاز به حافظه ای بیش از آنچه که سیستم می تواند فراهم کند.	نبود حافظه
مراجعه به محلهای غیر مجاز در حافظه	تجاوز از حدود
تلاش برای دسترسی به منبعی که مجاز به استفاده از آن نیست.	خطای حفاظت
مانند تقسیم بر صفر یا تلاش برای ذخیره عددی بزرگتر از ظرفیت سخت افزاری	خطای محاسباتی
مانند پیدا نکردن یک فایل	خطای ورودی/خروجی
تلاش برای اجرای دستورالعملی که وجود ندارد	دستورالعمل نامعتبر
تلاش برای اجرای دستورالعملی که مخصوص سیستم عامل است	دستورالعمل ممتاز
داده با نوع نامناسب یا بدون مقدار اولیه	استفاده نامناسب از داده
به دلایلی مانند بن بست	دخالت سیستم عامل
با پایان یک فرایند، فرایندهای فرزند آن نیز پایان داده شوند	پایان یافتن پدر
فرایند حق پایان دادن به هر یک از فرایندهای فرزند خود را دارد.	درخواست پدر

بلوک کنترل فرایند (PCB)

هر فرایند در سیستم بوسیله یک ساختاری به نام بلوک کنترل فرایند (Process Control Block) مشخص می‌شود. این بلوک مهمترین و محوریتترین ساختمان داده در سیستم عامل است که تمام اطلاعات مورد نیاز سیستم عامل در مورد یک فرایند را در بردارد.

PCB شامل موارد زیر است:

- ۱- حالت فرایند : یک فرایند می تواند در یکی از حالات جدید، آماده، اجرا، انتظار و غیره باشد.
- ۲- شمارنده برنامه : شامل آدرس دستور بعدی که باید اجرا شود.
- ۳- اطلاعات زمانبندی CPU : شامل اولویت فرایند و اشاره گر به صف زمانبندی.
- ۴- اطلاعات مدیریت حافظه : شامل مقدار ثباتهای پایه و حد، جدولهای صفحه یا قطعه.
- ۵- اطلاعات حسابرسی : میزان استفاده از پردازنده، محدودیتهای زمانی، شماره فرایند.
- ۶- اطلاعات وضعیت I/O : شامل لیست دستگاههای I/O تخصیص داده شده به فرایند
- ۷- ثباتهای CPU

تفاوت تعویض حالت با تعویض فرایند

تعویض حالت و تعویض فرایند (تعویض متن) دو مفهوم مجزا هستند. ممکن است تعویض حالت بدون تغییر حالت فرایندی که در حال اجراست، صورت گیرد. تعویض فرایند که متضمن تغییر حالت است، در مقایسه با تعویض حالت، به تلاش بیشتری نیاز دارد.

گام‌های که در یک تعویض فرایند کامل وجود دارد:

- ۱- ذخیره سازی متن پردازنده
- ۲- بهنگام سازی بلوکهای کنترل فرایندی که هم اکنون در حالت اجراست.
- ۳- انتقال بلوک کنترل فرایند مربوط به این فرایند به صف مناسب
- ۴- انتخاب فرایند دیگری برای اجرا
- ۵- بهنگام کردن بلوک کنترل فرایند مربوط به فرایند انتخاب شده
- ۶- بهنگام سازی ساختمان داده های مدیریت حافظه
- ۷- بارگذاری مجدد متن پردازنده

فرایند معلق

فرایند معلق، فرایندی است که فوراً آماده اجرا نیست. فرایند معلق توسط عاملی مانند سیستم عامل، خودش یا فرایند پدر، در حالت معلق قرار گرفته. تا وقتی عامل تعلیق فرمان ندهد، نمی توان فرایند را از حالت معلق خارج کرد.

دلایل معلق کردن یک فرایند

- ۱- **مبادله**: برای آوردن فرایندی که آماده اجراست، سیستم عامل نیاز به آزاد کردن حافظه کافی دارد.
- ۲- **ترتیب زمانی**: اجراء به طور دوره ای و به تعلیق رفتن آن هنگام انتظار برای اجرای بعدی
- ۳- **درخواست کاربر محاوره ای**: به منظور اشکالزدایی یا استفاده از منابع
- ۴- **دلایل دیگر سیستم عامل**: معلق کردن یک فرایند مضمون
- ۵- **درخواست فرایند پدر**: به طور نمونه فرایند A، فرایند B را تولید کرده تا عمل خواندن از یک فایل را انجام دهد و فرایند B در هنگام خواندن با خطایی مواجه شود و آنرا به فرایند A گزارش دهد، فرایند A در این حالت برای رسیدگی به علت خطا، فرایند B را به حالت معلق می برد.

نمودار تغییر حالت فرایند با حالات معلق

وقتی هیچ یک از فرایندهای موجود در حافظه اصلی در حالت آماده نیستند، سیستم عامل یکی از فرایندهای مسدود را از حافظه اصلی خارج و به صف فرایندهای معلق در روی دیسک می برد (مبادله) و فرایند دیگری را از صف معلق به حافظه اصلی می آورد یا درخواست فرایند جدیدی را می پذیرد و اجرا با این فرایند جدید ادامه می یابد.

وجود وضعیت معلق، موجب اجرای فرایند جدید حتی در صورت پر بودن حافظه اصلی می شود.

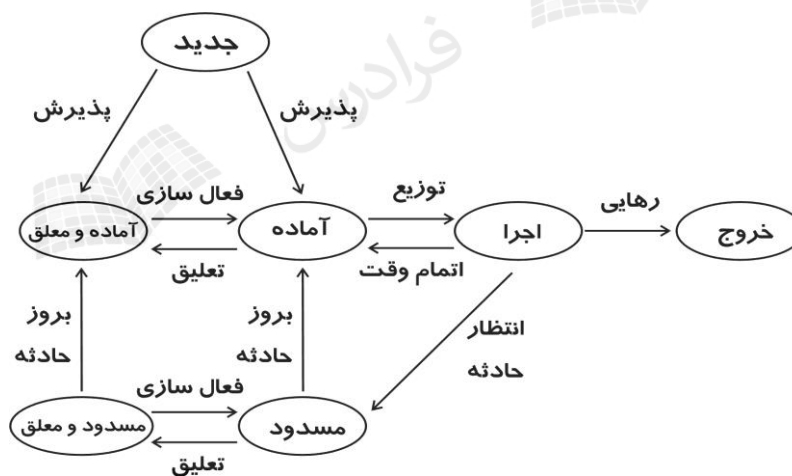
بنابراین حالت معلق را به حالت‌های گفته شده، اضافه می کنیم:

۱- مسدود و معلق (Suspend-Wait)

فرایند مورد نظر در حافظه ثانوی و منتظر حادثه ای است.

۲- آماده و معلق (Suspend-Ready)

فرایند مورد نظر در حافظه ثانوی و به محض لود شدن در حافظه اصلی آماده اجراست.



حالات ممکن در این نمودار

۱- مسدود به مسدود و معلق

وقتی هیچ فرایند آماده ای وجود نداشته باشد، فرایند مسدودی به خارج برده می شود تا جا برای فرایند دیگری که مسدود نیست فراهم شود. این تغییر حالت حتی در مواردی که فرایندهای آماده هم وجود دارد می تواند جهت حفظ کارایی انجام شود.

۲- مسدود و معلق به آماده و معلق

وقتی حادثه ای که یک فرایند مسدود و معلق منتظر آن بوده است رخ دهد به حالت آماده و معلق می‌رود.

۳- آماده و معلق به آماده

وقتی که هیچ فرایند آماده ای در حافظه اصلی نباشد، سیستم عامل یک فرایند آماده و معلق را به حالت آماده می آورد. همچنین ممکن است فرایند موجود در حالت آماده و معلق دارای اولویت بیشتری نسبت به همه فرایندهای آماده باشد که در این حالت فرایند به حالت آماده، آورده می شود.

۴- آماده به آماده و معلق

به طور معمول سیستم عامل ترجیح می دهد یک فرایند مسدود را به جای فرایند آماده، به حال تعلیق در آورد. ولی در صورتی که راهی برای خالی کردن حافظه اصلی نباشد، یک فرایند آماده را به تعلیق در می آورد. سرانجام ممکن است سیستم عامل یک فرایند آماده ولی با اولویت کم را به جای فرایند مسدودی که اولویتش بیشتر است و گمان می کند بزودی آماده می شود به حالت معلق می برد.

۵- مسدود و معلق به مسدود

اگر اولویت فرایندی که در صف مسدود و معلق است از اولویت همه فرایندهای موجود در صف آماده و معلق بیشتر باشد و سیستم عامل گمان کند که حادثه ای که این فرایند مسدود، منتظر آن بوده است به زودی رخ دهد، آن را به حالت مسدود می آورد. البته باید مقداری از حافظه اصلی خالی باشد تا آوردن یک فرایند مسدود به حافظه نسبت به یک فرایند آماده معقول به نظر برسد.

۶- اجرا به آماده و معلق

معمولا با پایان زمان منظور شده برای فرایند جاری، فرایند به حالت آماده منتقل می شود. در اینحال اگر این فرایند به خاطر فرایند با اولویت بیشتری قبضه شود، سیستم عامل می تواند فرایند جاری را مستقیما به صف آماده و معلق منتقل کند تا بخشی از حافظه اصلی آزاد شود.

۷- مختلف به خروج

ممکن است یک فرایند از هر حالتی به حالت خروج منتقل شود.

انواع زمانبندها

کلید چند برنامه‌گی زمانبندی است. زمانبندی بر روی کارایی سیستم اثر می‌گذارد، زیرا مشخص می‌کند کدام فرایندها منتظر مانده و کدام فرایندها به جلو بروند. انواع زمانبندی برای پردازنده عبارت است از:

۱- زمانبند بلند مدت (Long Term Scheduler)

تصمیم‌گیری در مورد افزودن به مجموعه فرایندها برای اجرا.

۲- زمانبند میان مدت (Middle Term Scheduler)

تصمیم‌گیری در مورد افزودن به تعداد فرایندهایی که بخشی یا تمام آنها در حافظه اصلی است.

۳- زمانبند کوتاه مدت (Short Term Scheduler)

تصمیم‌گیری در مورد اینکه کدام یک از فرایندهای موجود در حافظه اصلی، برای اجرا توسط پردازنده انتخاب شود.

۴- زمانبندی ورودی/خروجی

تصمیم‌گیری می‌گیرد که کدام درخواست I/O فرایندها به وسیله یک دستگاه I/O موجود انجام بگیرد.

وظیفه فعال‌سازی و تعلیق فرایندها بر عهده زمانبند میان مدت (Medium-term scheduler) می‌باشد.

زمانبند میان مدت، فرایندی را از حافظه اصلی حذف و به حافظه جانبی می‌برد. این فرایند بعداً می‌تواند به حافظه اصلی لود شود. این الگو را مبادله (swapping) می‌گویند.

ایده اصلی زمانبندی میان مدت، این است که می‌تواند فرایندی را از حافظه حذف کند و درجه چند برنامه‌گی را کاهش دهد.

زمانبند بلند مدت، ترکیب خوبی از فرایندهای I/O limited و CPU Limited، انتخاب می‌کند.

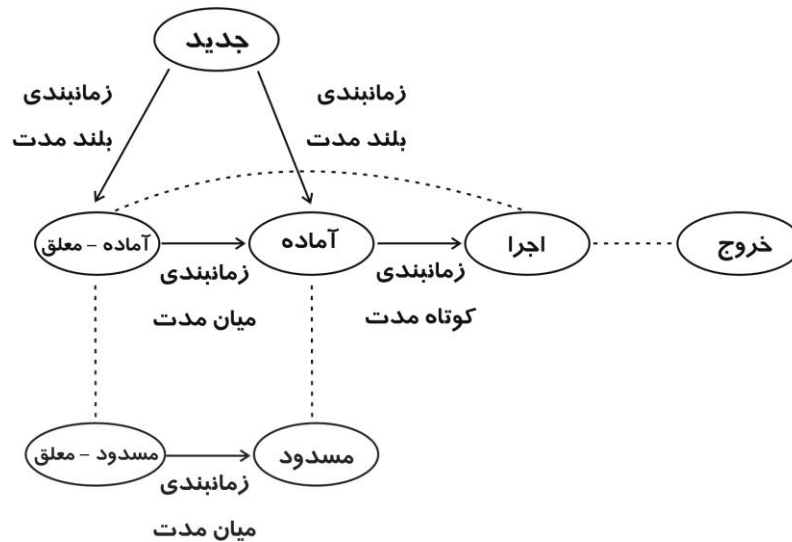
نام دیگر زمانبند بلند مدت، زمانبند کار است.

نام دیگر زمانبند کوتاه مدت، زمانبند پردازنده است.

زمانبند بلند مدت نسبتاً دفعات کمی اجرا می‌شود، زمانبند میان مدت نسبتاً دفعات بیشتری به اجرا در می‌آید و زمانبند کوتاه مدت، بیشترین دفعات اجرا را دارد.

تذکره: اکثر سیستم‌های اشتراک زمانی فاقد زمانبند بلند مدت می‌باشند.

نمودار تغییر حالت فرایند همراه با زمانبندی



مثال

اگر ۱۰ میلی ثانیه طول بکشد تا فرایندی انتخاب گردد که ۱۰۰ میلی ثانیه اجرا شود، آنگاه چند درصد از زمان CPU، صرف زمانبندی کار می شود (به هدر می رود)؟

$$\frac{10}{10+100} \times 100 = 9\%$$



توزیع کننده (Dispatcher)

توزیع کننده، پیمانه ای است که کنترل را به پردازنده ای می دهد که توسط زمانبند کوتاه مدت انتخاب شده است. این عمل شامل موارد زیر است:

- ۱- تعویض بستر (Context Switch)
- ۲- تغییر به حالت کاربر
- ۳- پرش به محل مناسبی در برنامه کاربر و آغاز مجدد آن برنامه.

نخ (Thread)

تا به حال، مفهوم فرایند را به صورت واحدی معرفی کرده ایم که دارای دو خصوصیت تملک منبع و توزیع وقت پردازنده است. این دو خصوصیت می توانند مستقلاً توسط سیستم عامل رسیدگی شوند. برای تمایز این دو خصوصیت، به توزیع وقت پردازنده، نخ و به تملک منبع، فرایند می گویند.

مهم ترین مزیت استفاده از سیستم های چند نخی این است که در فرایند تک نخ، هر گاه فراخوان سیستمی مسدود کننده ای اجرا شود، یا یک نقص صفحه رخ دهد، کل فرایند مسدود می شود. به عنوان مثال یک برنامه صفحه گسترده را در نظر بگیرید و فرض کنید کاربری می خواهد به طور دائمی و تعاملی مقادیر را تغییر دهد. می دانیم که در برنامه صفحه گسترده، وابستگی تابعی بین سلول های مختلف باید حفظ شود. بنابراین اگر سلولی تغییر کند، محاسبات زیادی انجام می گیرد. اگر فقط یک نخ وجود داشته باشد، وقتی برنامه منتظر دریافت ورودی است، محاسبات نمی تواند ادامه یابد، چون فراخوان سیستمی دریافت ورودی، کل فرایند را مسدود کرده است. راه حل این است که از یک نخ برای خواندن ورودی کاربر و از نخ دیگری برای بهنگام سازی صفحه گسترده استفاده شود.

مثال

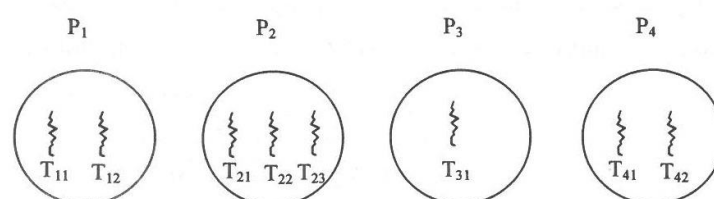
بسیاری از صفحات وب، شامل چند تصویر کوچک هستند که مرورگر باید یک اتصال جداگانه را برای دریافت هر تصویر برقرار کند که زمان زیادی لازم دارد. اگر در مرورگر از روش چند نخی استفاده شود، هر کدام از نخ ها می تواند به طور همزمان با نخ های دیگر تصویر مربوط به خود را درخواست کند. به عنوان مثالی دیگر از کاربرد نخ ها، جهت حفاظت در مقابل قطع برق می توان یک کلمه پرداز را طوری طراحی کرد که هر یک دقیقه بافر RAM را روی دیسک بنویسد. یک نخ می تواند تنها برای گرفتن پشتیبان دوره ای ایجاد شود و خودش را مستقیماً با سیستم عامل زمانبندی کند.

نخها و فرایندها می توانند ۴ حالت را بوجود آورند:

- ۱- یک فرایند- یک نخ
- ۲- یک فرایند - چند نخ
- ۳- چند فرایند- یک نخ در هر فرایند
- ۴- چند فرایند- چند نخ در هر فرایند

مثال

شکل زیر سیستمی با چهار فرایند را نشان می دهد که فرآیند اول شامل دو نخ T_{11} و T_{12} می باشد. فرایند دوم شامل سه نخ است و ...



- حالات اصلی نخ عبارت است از: اجرا، آماده و مسدود. (نخ حالت معلق ندارد)
- تمام نخ‌های یک فرایند، در حالت و منابع آن فرایند شریک هستند.
- اگر نخ یک فرایند در حال اجرا باشد و آن فرایند به حالت خروج برود، نخ نمی‌تواند به اجرا ادامه دهد.
- تعویض متن میان دو نخ متعلق به دو فرایند جداگانه، مثل این است که تعویض متن فرایند رخ داده است. یعنی عمل سوئیچینگ مابین دو نخ متعلق به دو فرایند جداگانه از نوع تعویض متن فرآیندی است.
- نخ در برنامه نویسی، بخشی از یک فرایند یا برنامه بزرگتر می‌باشد.
- ساختار نخ گاهی در محیط تک پردازنده ای، برای ساده کردن ساختار برنامه ای که منطقی‌ا عمل متعددی را انجام می‌دهد، نیز مفید است.
- با تقسیم یک کار به چند نخ، برنامه ساز می‌تواند کنترل زیادی روی مولفه ای بودن آن کاربرد و تنظیم وقت حوادث مربوط به آن داشته باشد.

پیاده سازی نخ

سه روش برای پیاده سازی و مدیریت نخ وجود دارد:

- ۱- نخ‌های سطح کاربر
- ۲- نخ‌های سطح هسته
- ۳- روش‌های ترکیبی

نخ‌های سطح کاربر

در این روش تمام عملیات راهبری در فضای آدرس کاربر انجام می‌شود. در این روش هزینه ایجاد نخ با هزینه تخصیص حافظه برای برپا سازی پشته (stack) نخ تعیین می‌شود. تعویض متن نخ اغلب با تعداد کمی از دستورات العمل‌ها انجام می‌شود و فقط کافی است محتوای ثبات‌های پردازنده برای نخ فعلی، ذخیره شود و سپس مقادیر قبلا ذخیره شده مربوط به نخ‌ی که به آن سوئیچ می‌شود، بار گذاری شوند. در تعویض متن نخ‌ها نیازی به فلاش کردن TLB، تغییر نگاشت‌های حافظه و حسابداری پردازنده نمی‌باشد.

مزایای کتابخانه نخ سطح کاربر

- ۱- ایجاد و حذف سریع نخ‌ها
- ۲- همگام سازی سریع نخ‌ها
- ۳- تعویض متن کم هزینه و سریع

نقاط ضعف نخ‌های سطح کاربر

- ۱- اگر یک نقص صفحه (page fault) برای یک نخ رخ دهد، کل فرایند و در نتیجه همه نخ‌های درون آن به اشتباه مسدود می‌شود.
- ۲- اگر نخ‌ی یک فراخوان سیستمی بلوکه‌کننده را صدا بزند، کل فرایند و همه نخ‌های درون آن نیز به اشتباه مسدود می‌شوند.
- ۳- چون سیستم عامل از وجود نخ‌ها آگاه نمی‌باشد، نخ‌ها را بین چندین پردازنده به خوبی پخش و زمان بندی نمی‌کند.

نخ‌های سطح هسته


اگر نخ‌ها در هسته سیستم عامل پیاده‌سازی شوند، مشکلات روش قبل رخ نمی‌دهند. اما چون هر عملیات نخ باید توسط هسته انجام شود، نیاز به یک فراخوانی سیستمی دارد و هزینه بالا می‌رود. تعویض متن نخ در این روش ممکن است به اندازه تعویض متن فرایند پر هزینه باشد.

تذکر: مزایا و معایب نخ‌های سطح کاربر و نخ‌های سطح هسته، بر عکس یکدیگرند.

روش ترکیبی

برای غلبه بر مشکلات دو روش قبل، نخ‌های سطح کاربر و سطح هسته را را تلفیق می‌کنیم. برای این کار فرایندهای سبک وزن (LWP) ایجاد می‌کنیم. LWP دورن متن یک فرایند اجرا شده و به ازای هر فرایند ممکن است چند LWP وجود داشته باشد. سیستم علاوه بر داشتن LWP ها، یک بسته نخ سطح کاربر برای عملیات ایجاد و حذف نخ‌ها به برنامه‌های کاربردی ارائه می‌دهد. بسته نخ کاملاً در فضای کاربر پیاده‌سازی می‌شود و تمام عملیات روی نخ‌ها بدون دخالت هسته انجام می‌شود. برنامه‌های کاربردی چند نخ‌ی با ایجاد نخ‌ها و سپس انتساب هر نخ به یک LWP ساخته می‌شوند.

در این روش اگر نخ‌ی فراخوان سیستمی مسدودکننده‌ای را اجرا کند، اجرا از مد کاربر به مد هسته تغییر می‌کند ولی همچنان در متن LWP جاری ادامه می‌یابد. در جایی که LWP جاری نتواند ادامه دهد، متن به LWP دیگری تعویض می‌شود که باعث یک تعویض متن برای بازگشت به مد کاربر نیز می‌شود.

در روش ترکیبی، فراخوانی‌های سیستمی از نوع مسدود با تامین هم‌روندی حمایت می‌شوند. 

تغییر متن (context switch) در نخ‌های یک فرایند، اشاره‌گر پشته (SP) را تغییر می‌دهد ولی ثبات‌ها و جداول مدیریت حافظه را تغییر نمی‌دهد و ارتباطی با TLB ندارد.

TLB یک سخت‌افزار جستجوی سریع، کوچک و پنهان است. در صفحه‌بندی تعداد اندکی از ورودی‌های جدول صفحه در TLB قرار می‌گیرد.

(TLB : Translation Lookaside Buffers)



کنکور ارشد

(مهندسی کامپیوتر - دولتی ۸۶)

۱- در مدل انتقال حالت (transition state) یک سیستم عامل که اجازه داده می شود یک فرآیند از حافظه اصلی بیرون کشیده شود و در زمان مناسب دوباره به حافظه اصلی بازگردانده شود (مانند unix)، کدام یک از انتقال ها نمی تواند مجاز باشد؟

(sleep swapped: جابجا شده، به خواب رفته)

(ready to run in memory : آماده برای اجرا در حافظه)

۱) از sleep swapped به ready to run in memory

۲) از ready to run swapped به ready to run in memory

۳) از ready to run in memory به ready to run swapped

۴) از sleep in memory به sleep swapped

پاسخ: گزینه ۱ جواب است.

با توجه به نمودار حالات فرایند، گزینه یک، یعنی انتقال از "مسدود و معلق" به "آماده"، مجاز نمی باشد. به تعاریف زیر توجه کنید:

مسدود و معلق : sleep swapped

مسدود : sleep in memory

آماده : ready to run in memory

آماده و معلق : ready to run swapped

طبق این تعاریف گزینه ها را می توان به صورت زیر نوشت:

۱) از "مسدود و معلق" به "آماده"

۲) از "آماده" به "آماده و معلق"

۳) از "آماده و معلق" به "آماده"

۴) از "مسدود" به "مسدود و معلق"

(مهندسی کامپیوتر - دولتی ۹۲)

۲- کدام گزینه درباره مدل های چند نخه درست نیست؟

۱) مدل های یک به یک و چند به چند، توانایی بکارگیری بهتر از پردازنده ها/هسته ها را دارند.

۲) مدل چند به یک نسبت به مدل یک به یک، از کارایی کمتری برخوردار است.

۳) در مدل های یک به یک و چند به یک، امکان همزمانی کامل بین نخ ها وجود دارد.

۴) مدل یک به یک نسبت به مدل چند به یک، از همزمانی بیشتری برخوردار است.

پاسخ: گزینه ۳ نادرست است.

چون در مدل چند به یک، امکان همزمانی کامل بین نخ‌ها وجود ندارد. توضیح بیشتر: بین نخ‌ها و فرایندها می‌تواند ۴ رابطه زیر برقرار باشد:

هر نخ اجراء، یک فرایند یکتا با فضای آدرسها و منابع خویش است.	UNIX	یک به یک
یک فرایند، یک فضای آدرس و مالکیت پویای منابع را تعریف می‌کند. ممکن است نخهای متعدد در داخل آن فرایند ایجاد و اجرا گردد.	NT , Solris, OS/2	چند به یک
ممکن است یک نخ از محیط یک فرایند به محیط فرایند دیگر مهاجرت کند. این موضوع حرکت یک نخ در بین سیستمهای مجزا را میسر می‌سازد.	Clouds	یک به چند
ترکیب خصوصیات موارد چند به یک و یک به چند	TRIX	چند به چند

در رابطه با این مدل‌ها می‌توان گفت که:

الف- توانایی بکارگیری پردازنده‌ها در مدل‌های یک به یک و چند به چند، بیشتر است.

ب- کارایی مدل یک به یک از مدل چند به یک، بیشتر است.

ج- مدل یک به یک همزمانی بیشتری نسبت به مدل چند به یک دارد.

(مهندسی IT - دولتی ۹۱)

۳- در رابطه با مدیریت نخ (thread) کدام یک از جملات زیر صحیح است؟

(توجه: LWP مخفف Light Weight Process است و محیط اجرای نخ می‌باشد.)

(۱) تغییر متن (Context switch) مابین نخ‌ها شامل: (۱) ذخیره ثبات‌های پردازنده مربوط به نخ بیرون رونده و بار کردن ثبات‌های پردازنده مربوط به نخ داخل شونده و (۲) ذخیره لیست فایل‌های باز شده توسط نخ است.

(۲) یک نخ عادی در طول حیات خود ممکن است در LWP‌های متفاوتی، بخش‌هایی از اجرای خود را بگذراند.

(۳) به ازای هر نخ، سیستم عامل یک LWP ایجاد می‌کند و نخ تا پایان حیات خود به آن LWP منتسب است. زمان بندی نخ می‌تواند توسط سیستم عامل یا کاربر انجام پذیرد.

(۴) نخ مستقیماً زیر نظر سیستم عامل اجرا می‌شود و مدیریت آن نمی‌تواند در سطح کاربر باشد.

پاسخ: گزینه ۲ درست است.

فرایند سبک وزن (LWP)، نگاهی بین نخ سطح کاربر و نخ سطح هسته می‌باشد. هر LWP از یک نخ سطح کاربر یا بیشتر حمایت می‌کند و به یک نخ سطح هسته می‌نگارد. LWP‌ها مستقلاً توسط هسته زمانبندی شده و ممکن است به صورت موازی روی پردازنده‌های متعدد اجرا شوند.

یک نخ عادی در طول حیات خود ممکن است در LWP‌های متفاوتی، بخش‌هایی از اجرای خود را بگذراند. بنابراین گزینه ۲ درست است. گزینه ۳ نادرست است، چون تعداد LWP‌ها به اندازه تعداد نخ‌هایی است که هم‌اکنون با هسته درگیر هستند و سیستم عامل به ازای هر نخ، یک LWP ایجاد نمی‌کند. به طور نمونه اگر دو LWP موجود باشد و نخ سوم فرایند، درخواستی از هسته داشته باشد، باید منتظر بماند تا یکی از LWP‌های قبلی کارش با هسته پایان یابد.

فصل ۳: زمان بندی پردازنده

هدف از زمان بندی پردازنده، تخصیص فرایندها به پردازنده در طول زمان است به گونه ای که هدف های سیستم از قبیل زمان پاسخ، توان عملیاتی و کارایی پردازنده را برآورده سازد. زمان بندی پردازنده، اساس سیستم‌های عامل چند برنامه ای است. با حرکت پردازنده بین فرایندها، سیستم عامل می تواند بهره وری کامپیوتر را افزایش دهد.

معیارهای زمان بندی

معیارهای زمان بندی به دو دسته تقسیم می شوند. معیارهای از دید کاربر مانند زمان پاسخ و معیارهای از دید سیستم: مانند توان عملیاتی، بهره وری پردازنده.

توان عملیاتی (throughput): تعداد فرایندهایی که در واحد زمان تکمیل می شوند.

زمان انتظار = " زمان خروج - زمان اجرا - زمان ورود "

زمان برگشت = " زمان خروج - زمان ورود "

زمان برگشت = " زمان انتظار + زمان اجرا "

تذکر: نام های دیگر زمان برگشت (turnaround time) عبارتند از:

زمان اجرای کامل، زمان کل، زمان تکمیل و زمان پاسخ (response time).

استفاده از پردازنده (بهره وری CPU) در سیستم های اشتراکی معیار مهمی است.

به زمان مشخصی از پردازنده که برای اجرای مجموعه دستور العملهای مربوطه به CPU در نظر گرفته می شود، CBT (CPU Bust Time) می گویند.

از دید کاربر زمان پاسخ و از دید سیستم بهره وری پردازنده، مهم است.

معیارهای زمان بندی گفته شده به یکدیگر وابسته هستند و بهینه سازی همه آن ممکن نمی باشد.

دسته بندی سیاست های زمان بندی

۱- بدون قبضه کردن (non preemptive)

در حالت بدون قبضه کردن (انحصاری)، همین که یک فرایند در حالت اجرا قرار گرفت، آنقدر به اجرا ادامه می‌دهد تا خاتمه یابد یا اینکه خودش (داوطلبانه)، برای انتظار I/O مسدود شود.

۲- با قبضه کردن (preemptive)

در حالت با قبضه کردن (غیر انحصاری)، فرایند در حال اجرا می‌تواند توسط سیستم عامل متوقف شود و به حالت آماده منتقل شود.

تصمیم به قبضه کردن می‌تواند در یکی از حالات زیر انجام گیرد:

۱- به صورت دوره ای براساس وقفه ساعت

۲- در زمان ورود یک فرایند جدید

۳- در زمانی که فرایند مسدودی به حالت آماده برود.

سیاست هایی که با قبضه کردن همراه است سربر بیشتری را به همراه دارند ولی خدمت بهتری را ارائه می‌دهند.

الگوریتم های زمان بندی

الگوریتم های زمان بندی پردازنده عبارتند از:

۱- سرویس به ترتیب ورود (FCFS)

۲- نوبت گردشی (RR)

۳- کوتاهترین فرایند (SJF یا SPN)

۴- کوتاهترین زمان باقی مانده (SRT)

۵- بالاترین نسبت پاسخ (HRRN)

۶- فیدبک (FB)

و الگوریتم های زیر که در کتاب سیلبرشاتز آورده شده است.

Priority - ۳ MLQ - ۲ MLFQ - ۱

FCFS : First-Come First-Served
 RR : Round Robin
 SPN : Shortest Process Next
 SRT : Shortest Remaining Time
 HRRN : Highest Response Ratio Next
 FB : Feed Back
 MLFQ : Multi Level Feedback Queue
 MLQ : Multi Level Queue

سرویس به ترتیب ورود (FCFS)

در این الگوریتم فرایندی انتخاب می‌شود که بیشتر منتظر بوده است، یعنی زودتر CPU را درخواست کرده است. پیاده سازی این الگوریتم با یک صف (FIFO) انجام می‌شود. وقتی فرایندی وارد صف آماده می‌شود، PCB آن در انتهای صف قرار می‌گیرد و با آزاد شدن CPU، فرایند موجود در اول صف، CPU را در اختیار خواهد گرفت و از صف حذف می‌شود.

ویژگی های زمان بندی FCFS

- ۱- گرسنگی ندارد.
- ۲- برای فرایندهای طولانی بسیار بهتر از فرایندهای کوتاه عمل می‌کند.
- ۳- سربار حداقل است، چون نیازی به اطلاعات قبلی در مورد فرایندها نمی‌باشد.
- ۴- تابع انتخاب برابر Max (wait) است.
- ۵- میانگین زمان انتظار بسیار بالا می‌باشد.
- ۶- در یک سیستم تک پردازنده ای، روش خوبی نیست.
- ۷- انحصاری است.
- ۸- به فرایندهای I/O bound ضربه می‌زند. (چون وقتی فرایند CPU bound در حال اجراست، فرایندهای I/O bound باید منتظر باشند).
- ۹- می‌تواند باعث استفاده ناکارآمد از CPU و دستگامهای I/O شود. وقتی فرایند جاری حالت اجرا را ترک می‌کند، فرایندهای آماده I/O bound به سرعت حالت اجرا را گذرانده و برای رویدادهای I/O مسدود می‌شوند. در این لحظه، اگر فرایند CPU bound نیز مسدود باشد، CPU بیکار می‌شود.

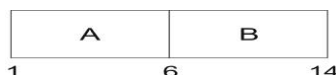
تذکر: برای بدست آوردن زمان پاسخ و زمان انتظار فرایندها از روشی به نام گانت استفاده می‌شود.

مثال

با توجه الگوریتم FCFS، گانت را برای فرایندهای زیر رسم نمایید.

	زمان ورود	زمان اجرا
A	1	5
B	4	8

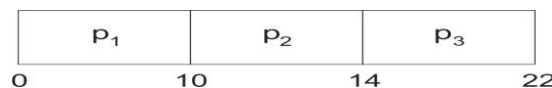
حل:



این نمودار مشخص می کند که در لحظه یک اجرای فرایند A ، بدون هیچ انتظاری شروع شده و بعد از 5 میلی ثانیه در لحظه 6 ، پایان یافته است. سپس اجرای فرایند B با 2 میلی ثانیه تاخیر شروع شده و بعد از 8 ثانیه دیگر در لحظه 14 پایان یافته است. از گانت می توان زمان پاسخ را نیز بدست آورد. کافی است که زمان ورود (که در جدول اولیه داده شده) را از زمان خروج (که در گانت مشخص است) کم کرد. بنابراین زمان پاسخ فرایند A برابر 5 میلی ثانیه (6-1) و زمان پاسخ فرایند B برابر 10 میلی ثانیه (14-4) می باشد. ■

مثال

در صورت استفاده از الگوریتم FCFS ، میانگین زمان انتظار برای سه فرایند P1 و P2 و P3 با زمانهای اجرای 10 و 4 و 8 میلی ثانیه را بدست آورید. (فرایندها در زمان صفر وارد شده اند).
 حل: ابتدا فرایند P1 اجرا شده که 10 میلی ثانیه طول می کشد، سپس فرایند P2 که 10 میلی ثانیه منتظر بوده است اجرا شده و بعد از 4 میلی ثانیه (در زمان 14) اجرای آن تمام می شود. در نهایت فرایند P3 بعد از 14 میلی ثانیه انتظار اجرا می شود. گانت آن به صورت زیر است:



زمان انتظار را به کمک رابطه "زمان خروج - زمان اجرا" محاسبه می کنیم:

$$\frac{(10-0) + (14-4) + (22-8)}{3} = \frac{0+10+14}{3} = 8$$

■

مثال

در الگوریتم FCFS با توجه به فرایندهای داده شده، مطلوب است:

الف- میانگین زمان اجرا

ب- میانگین زمان انتظار

ج- میانگین زمان برگشت (پاسخ)

نام فرایند	زمان ورود	زمان پردازش
A	0	3
B	1	3
C	4	3
D	6	2

حل:

گانت فرایندها به صورت زیر می باشد:



الف- میانگین زمان اجرا:

$$\frac{3+3+3+2}{4} = \frac{11}{4}$$

ب- میانگین زمان انتظار:

$$\frac{(3-3-0) + (6-3-1) + (9-3-4) + (11-2-6)}{4} = \frac{0+2+2+3}{4} = \frac{7}{4}$$

ج- میانگین زمان برگشت:

$$\frac{(3-0) + (6-1) + (9-4) + (11-6)}{4} = \frac{3+5+5+5}{4} = \frac{18}{4}$$

تذکر: البته می توانستیم با توجه به رابطه زیر نیز ، میانگین زمان برگشت را محاسبه کنیم:

میانگین زمان برگشت = میانگین زمان اجرا + میانگین زمان انتظار

نوبت گردشی (RR)

این الگوریتم شبیه به FCFS است، با این تفاوت که زمانبند پردازنده بین فرایندها در یک صف چرخشی حرکت کرده و CPU حداکثر به مدت یک کوانتوم زمانی به هر فرایند تخصیص داده می‌شود. اگر در یک کوانتوم زمانی، اجرای پروسسی تمام نشود به ته صف می‌رود.

ویژگی های روش RR

- ۱- در سیستمهای اشتراک زمانی یا در سیستم پردازش تراکنش بسیار مؤثر می‌باشد.
- ۲- اگر برهه زمانی از زمان اجرای بلندترین فرایند بیشتر باشد، سیاست RR به FCFS تنزل می‌یابد.
- ۳- زمان بندی RR، غیر انحصاری (قبضه شدنی) می‌باشد.
- ۴- گرسنگی ندارد.
- ۵- عملکرد آن عادلانه است.
- ۶- اگر برهه زمانی خیلی کوچک باشد، توان عملیاتی آن کم است.
- ۷- برای فرایندهای کوتاه، زمان برگشت خوبی ارائه می‌کند.
- ۸- از مشکلات آن، رفتار با فرایندهای در تنگنای پردازنده در مقایسه با فرایندهای در تنگنای I/O می‌باشد.

مثال

میانگین زمان انتظار سه پردازش زیر با استفاده از سیاست زمان بندی RR با کوانتوم زمانی 3 میلی ثانیه را بدست آورید؟ (زمان ورود = صفر) (زمان پردازش : P1=9, P2=2, P3=2)
حل: گانت بصورت زیر می باشد:

	p ₁	p ₂	p ₃	p ₁	p ₁
0	3	5	7	10	13

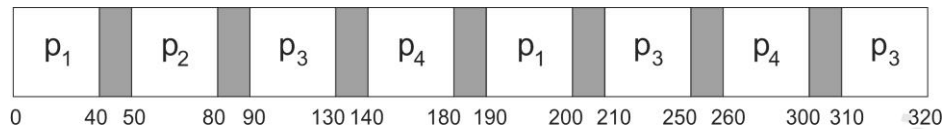
اجرای پروسس P1 در اولین برش زمانی تمام نمی‌شود و به انتهای صف می‌رود و پروسس P2 و P3 قبل از پایان برش زمانی، به پایان می‌رسند. میانگین زمان انتظار برابر است با:

$$\frac{(13-9) + (5-2) + (7-2)}{3} = 4$$



مثال

اگر زمان اجرای پروسسهای P4, P3, P2, P1 به ترتیب 80, 90, 30, 50 باشد و روش RR با کوانتوم زمانی 40 و زمان تعویض متن 10 استفاده شود، میانگین زمان برگشت کدام است؟ (زمان ورود = 0)
حل: گانت فرایندها به صورت زیر است:



میانگین زمان برگشت برابر است با:

$$\frac{200 + 80 + 320 + 300}{4} = 225$$

مثال

پنج فرایند با مشخصات زیر به یک سیستم با زمان بندی RR با برش زمانی q=1 وارد شوند. نمودار گانت را رسم کنید.

	زمان ورود	زمان اجرا
P1	0	2
P2	0	2
P3	1	1
P4	1	1
P5	2	1

فرض: همیشه بین فرایندی که در لحظه t برش زمانی خود را به پایان می‌رساند و فرایند ورودی در لحظه t، اولویت با فرایند قبلی موجود در سیستم است و در شرایط کاملا یکسان بین دو فرایند، اولویت با فرایند با شماره کوچکتر است.

حل:

گانت به صورت زیر می‌باشد:

P ₁	P ₂	P ₁	P ₃	P ₄	P ₂	P ₅
0	2	3	4	5	6	7
		1				

در لحظه 0، P1 و P2 وارد می‌شوند. پردازنده به P1 داده می‌شود و P2 در صف قرار می‌گیرد. در لحظه 1، پردازنده از P1 گرفته شده و بعد از P2 در صف قرار می‌گیرد. همچنین P3 و P4 که در همین لحظه وارد شده‌اند، در صف پشت سر P1 قرار می‌گیرند. پردازنده به P2 داده می‌شود. در لحظه 2، پردازنده از P2 گرفته شده و به انتهای صف بعد از P4 رفته و P5 که در همین لحظه وارد شده در انتهای صف بعد از P2 قرار می‌گیرد. پردازنده هم به P1 (فرایند ابتدای صف) داده می‌شود. در لحظه 3، اجرای P1 تمام شده و پردازنده به P3 که اول صف است داده می‌شود. در لحظه 4، اجرای P3 تمام شده و پردازنده به فرایند اول صف یعنی P4 داده می‌شود. در لحظه 5، اجرای P4 تمام شده و پردازنده به P2 داده می‌شود. در لحظه 6، اجرای P2 تمام شده و پردازنده به P5 داده می‌شود.

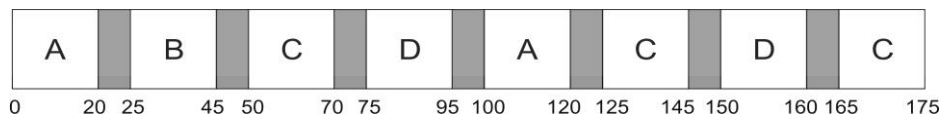


مثال

به کمک روش زمان بندی نوبه‌ای با کوانتوم زمانی 20 میلی ثانیه، متوسط زمان انتظار پردازنده‌ها را محاسبه کنید؟ (زمان Context Switch برابر 5 میلی ثانیه است). (زمان ورود همه فرایندها = 0)

فرایند	زمان اجرا
A	40
B	20
C	50
D	30

حل: گانت آن به صورت زیر است:



و میانگین زمان انتظار برابر است با:

$$\frac{(120 - 40) + (45 - 20) + (175 - 50) + (160 - 30)}{4} = \frac{80 + 25 + 125 + 130}{4} = 90$$

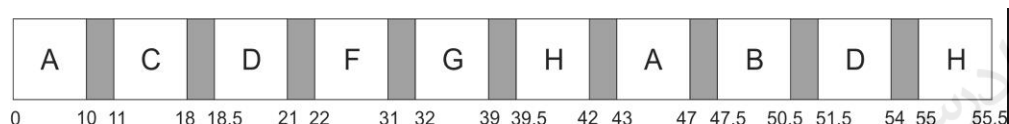


مثال

سیستمی شامل چهار فرایند است که داخل هر فرایند می تواند بیش از یک نخ اجرایی وجود داشته باشد مطابق جدول زیر مفروض است. برای فرایندها از الگوریتم Round-Robin با برش زمانی $q=10$ استفاده کنید. داخل هر فرایند از روش FIFO برای تعویض نخ ها استفاده می شود و تا زمانی که اجرای یک نخ تمام نشده، نوبت به نخ بعدی نمی رسد. برای تعویض فرایند 1ms و برای تعویض نخ در داخل فرایند 0.5ms زمان لازم است. نمودار گانت را رسم کنید.

فرایند	P ₁		P ₂		P ₃	P ₄	
نخ	A	B	C	D	F	G	H
زمان اجرا	14	3	7	5	9	7	3

حل: نمودار گانت به صورت زیر می باشد:

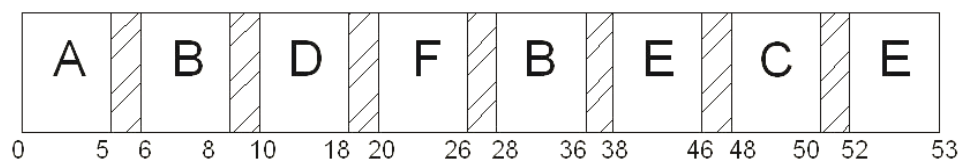


مثال

سیستمی شامل سه فرایند مطابق جدول زیر مفروض است. برای فرایندها از الگوریتم Round-Robin با برش زمانی 8 میلی ثانیه استفاده کنید. داخل هر فرایند از روش FIFO برای تعویض نخ ها استفاده می شود و تا زمانی که اجرای یک نخ تمام نشده، نوبت به نخ بعدی نمی رسد. برای تعویض فرایند 2ms و برای تعویض نخ در داخل فرایند 1ms زمان لازم است. میانگین زمان پاسخ را برای نخ های هر فرایند محاسبه کنید.

فرایند	P ₁			P ₂		P ₃
نخ	A	B	C	D	E	F
زمان اجرا	5	10	2	8	9	6

حل: نمودار گانت به صورت زیر می باشد:



میانگین زمان پاسخ برای نخ‌های هر فرایند برابر است با:

$$A = 5, B = 36, C = 50 \Rightarrow \frac{91}{3}$$

$$D = 18, E = 53 \Rightarrow \frac{71}{2}$$

$$F = 26$$



کوتاهترین فرایند (SJF یا SPN)

در این سیاست فرایندی برای اجرا انتخاب می‌شود که به کوتاهترین زمان پردازش نیاز دارد. یعنی فرایند کوتاه از روی فرایندهای بلند می‌گذرد و به ابتدای صف می‌آید.

ویژگی‌های الگوریتم SPN

۱- از معایب آن، نیاز به دانستن زمان پردازش هر فرایند است.

۲- انحصاری است.

۳- امکان گرسنگی فرایندهای طولانی وجود دارد.

۴- برای محیط‌های اشتراک زمانی، مناسب نیست.

۵- میانگین زمان انتظار، کمینه است.

مثال

میانگین زمان انتظار را برای فرایندهای زیر به روش SJF بدست آورید؟ (ورود همه در لحظه صفر)
($P_1=7, P_2=8, P_3=3, P_4=5$)

حل: ترتیب اجرا برابر است با: (از کوتاهترین کار شروع می‌شود)

P_3	P_4	P_1	P_2
0	3	8	15
			23

میانگین زمان انتظار برابر است با:

$$\frac{(15-7) + (23-8) + (3-3) + (8-5)}{4} = 6.5$$

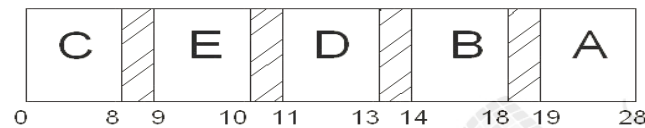


مثال

با توجه به جدول زیر متوسط زمان انتظار را در هر یک از روشهای SJF محاسبه کنید.
(زمان تعویض متن = ۱ میلی ثانیه)

پروسس	زمان ورود به سیستم	زمان موردنیاز پردازش
A	0	9
B	2	4
C	0	8
D	3	2
E	5	1

پاسخ: نمودار گانت به صورت زیر است:



زمان انتظار هر یک از فرایندها برابر است با:

$$A = 19, B = 12, C = 0, D = 8, E = 4 \Rightarrow \frac{19+12+0+8+4}{5} = \frac{43}{5} \blacksquare$$

پیش بینی زمان انفجار محاسباتی بعدی

در الگوریتم SJF راهی وجود ندارد که از انفجار محاسباتی بعدی آگاهی پیدا کنیم. یک روش این است که زمانبندی SJF تخمین زده شود. ممکن است طول بعدی را ندانیم، اما می توانیم اندازه اش را پیش بینی کنیم. با تخمین طول انفجار محاسباتی بعدی، می توانیم فرآیندی را انتخاب کنیم که طول انفجار محاسباتی بعدی آن کوتاهتر است. برای تخمین از رابطه $S_n = \alpha S_{n-1} + (1-\alpha)T_{n-1}$ استفاده می شود. ($0 \leq \alpha \leq 1$)

مثال

سه وظیفه A, B, C را در نظر بگیرید که تاکنون n-1 بار در سیکل آماده-اجرا-مسدود طی مسیر کرده اند. زمان اجرای واقعی سیکل n-1 ام این وظایف بترتیب، 2، 4 و 6 میلی ثانیه و زمان برآورد شده برای اجرای n-1 ام آنها نیز بترتیب 4، 6 و 6 میلی ثانیه می باشد. زمان اجرای واقعی در سیکل n ام به ترتیب 5، 4 و 7 است. با فرض $\alpha = 0.5$ نحوه زمانبندی این وظایف با استفاده از الگوریتم SJF را مشخص کنید.

حل: توسط الگوریتم سالمندی (aging)، می توان زمان اجرای کارها را به کمک رابطه $S_n = \alpha S_{n-1} + (1 - \alpha)T_{n-1}$ تخمین زد. که S_{n-1} زمان برآورد مرحله قبل و T_{n-1} زمان واقعی مرحله قبل می باشد. با فرض $\alpha = 0.5$ داریم:

$$S_n = 0.5 \times S_{n-1} + (1 - 0.5) \times T_{n-1} = 0.5 \times (S_{n-1} + T_{n-1})$$

بنابراین می توان S_n را به صورت زیر محاسبه کرد:

	S_{n-1}	T_{n-1}	S_n
A	4	2	$\frac{1}{2} \times (4 + 2) = 3$
B	6	4	$\frac{1}{2} (6 + 4) = 5$
C	6	6	$\frac{1}{2} (6 + 6) = 6$

با توجه به S_n به دست آمده، مشخص است که ترتیب اجرا برابر است با: $A \rightarrow B \rightarrow C$

(طبق الگوریتم SJF، کارها از کوچک به بزرگ اجرا می شوند)

حال با توجه به زمان اجرای واقعی در سیکل n ام که برابر 5، 4 و 7 می باشد، نمودار گانت را رسم می کنیم:

A	B	C
0 5	9	16

کوتاهترین زمان باقیمانده (SRT)

زمان بندی SRT یک نوع SPN با قبضه کردن است و فرایندی برای اجرا انتخاب می شود که انتظار می رود کوتاهترین زمان پردازش باقیمانده را داشته باشد. اگر فرایند جدیدی وارد صف آماده شود و زمان باقیمانده کمتری نسبت به فرایندی که در حال اجراست داشته باشد، فرایند در حال اجرا قبضه می شود و فرایند جدید اجرا می شود.

ویژگی های الگوریتم SRT

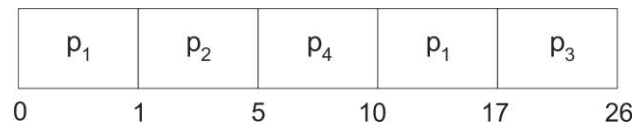
- ۱- غیر انحصاری است.
- ۲- امکان گرسنگی برای کارهای طولانی زیاد است.
- ۳- بر خلاف RR، وقفه های اضافی بوجود نمی آید، بنابراین سربار کاهش می یابد ولی از طرف دیگر، زمان خدمت سپری شده باید ثبت شود، که ایجاد سربار می کند.
- ۴- زمان کل SRT نسبت به SPN بهتر است، چون کار کوتاه اولویت بیشتری نسبت به کار بلند در حال اجرا دارد.

مثال

در صورت استفاده از الگوریتم SRT برای فرایندهای زیر، میانگین زمان انتظار را بدست آورید.

فرایند	زمان ورود	زمان اجرا
P1	0	8
P2	1	4
P3	2	9
P4	3	5

حل: اجرای فرایند P1 در لحظه صفر شروع شده و با ورود P2 در زمان 1، اجرای P1 قطع شده (چون P2 زمان اجرای کمتری نسبت به زمان باقیمانده برای P1 یعنی 7 میلی ثانیه دارد) و اجرای P2 شروع خواهد شد و بعد از اجرای P2، اجرای P4 شروع می‌شود و سپس اجرای P1 ادامه می‌یابد و در نهایت P3 اجرا خواهد شد. بنابراین گانت آن به صورت زیر است:



میانگین زمان انتظار برابر است با:

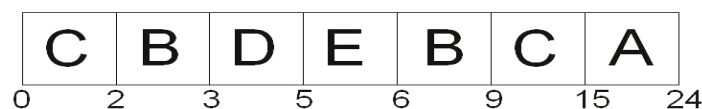
$$\frac{(17-8-0) + (5-4-1) + (26-9-2) + (10-5-3)}{4} = 6.5$$

مثال

با توجه به جدول زیر متوسط زمان انتظار را در هر یک از روشهای SRT محاسبه کنید.

پروسس	زمان ورود به سیستم	زمان موردنیاز پردازش
A	0	9
B	2	4
C	0	8
D	3	2
E	5	1

پاسخ: نمودار گانت به صورت زیر است:



زمان انتظار هر یک از فرایندها برابر است با:

$$A = 24, B = 7, C = 15, D = 2, E = 1 \Rightarrow \frac{24 + 7 + 15 + 2 + 1}{5} = \frac{49}{5}$$

بالاترین نسبت پاسخ (HRRN)

در این روش برای هر فرایند نسبت پاسخی از رابطه $\frac{W+S}{S}$ ، بدست می‌آید، سپس فرایندی اجرا می‌شود که بالاترین نسبت پاسخ را دارد.

($W =$ زمان انتظار برای CPU) و ($S =$ زمان اجرا) و ($W+S =$ زمان پاسخ (P))

ویژگی های الگوریتم HRRN

- ۱- گرسنگی ندارد.
- ۲- از معایب این روش، نیاز به تخمین زمان خدمت مورد نیاز قبل از به کارگیری می‌باشد.
- ۳- تابع انتخاب آن برابر $MAX(\frac{W+S}{S})$ است.
- ۴- سربار می‌تواند زیاد باشد.
- ۵- توان عملیاتی زیاد است.
- ۶- زمان بندی انحصاری (بدون قبضه کردن) است.

فیدبک (FB)

اگر نتوانیم روی زمان باقیمانده برای اجرا تمرکز کنیم، بهتر است روی زمان اجرای سپری شده تمرکز کنیم. زمان بندی FB براساس قبضه کردن صورت می‌گیرد و از یک روش اولویت پویا استفاده می‌شود. فرایندی که ابتدا وارد شود به صف 0 می‌رود و هنگامی که بعد از اولین اجرا به حالت آماده می‌رود در صف 1 با اولویت کمتر قرار می‌گیرد پس از هر اجرا به صف کم اولویت تر بعدی می‌رود. فرایند کوتاه بدون انتقال به صفهای پایین تر به سرعت اجرا می‌شود، اما فرایندهای طولانی به صفهای پایین می‌روند و فرایندهای جدیدتر و کوتاهتر به فرایندهای قدیمی تر و بلند تر ارجحیت دارند. در صف با کمترین اولویت از سیاست RR و در صفهای دیگر از سیاست FCFS استفاده می‌شود. (چون فرایند در صف کمترین اولویت، نمی‌تواند به صفهای پایین تر برود.)

✍ زمان بندی FB غیر انحصاری می‌باشد، چون فرایندی از صفی به صف دیگر می‌تواند منتقل شود.

زمان بندی های زیر در کتاب سیلبرشاتس آورده شده است.

زمان بندی صف بازخوردی چند سطحی (MLFQ)

این زمانبند غیر انحصاری، به فرایندها اجازه می‌دهد تا از صفی به صف دیگر منتقل شوند. فرایندی که زیاد CPU را در اختیار داشته به صف کم اولویت‌تر می‌رود و فرایند I/O bound و محاوره‌ای به صف با اولویت بالاتر می‌رود و فرایندی که زیاد در صف با اولویت پایین بوده به صف با اولویت بالاتر می‌رود، یعنی کهنگی (سالمندی) از ایجاد گرسنگی جلوگیری می‌کند.

مثال

زمانبند صف بازخوردی چند سطحی (MLFQ) با 3 صف (0 تا 2) را در نظر بگیرید. صف اول از روش RR با کوانتوم 8 میلی ثانیه و صف دوم نیز از روش RR با کوانتوم 16 میلی ثانیه و صف آخر از روش FCFS استفاده می‌کند. زمانبند ابتدا تمامی فرایندهای موجود در صف 0 را اجرا می‌کند، اگر صف 0 خالی بود فرایندهای صف 1 اجرا می‌شود و اگر صف 0 و 1 خالی بود فرایندهای صف 2 اجرا می‌شود. فرایند تازه وارد در صف 0 قرار می‌گیرد اگر در مدت 8 میلی ثانیه اجرای آن به پایان نرسد به انتهای صف 1 می‌رود. به فرایند موجود در ابتدای صف 1، کوانتوم زمانی 16 میلی ثانیه داده می‌شود که اگر اجرای آن در این مدت تمام نشود به صف 2 می‌رود. اگر هر دو صف 0 و 1 خالی باشد، فرایندهای صف 2 براساس FCFS اجرا می‌شوند. بنابراین فرایندهای طولانی به صف 2 می‌روند و به ترتیب FCFS اجرا می‌شوند. ■

مثال

یک سیستم تک پردازنده ای با صف بازخورد چند سطحی (MLFQ) را در نظر بگیرید. به صف اول تکه زمانی 8 میکرو ثانیه، به صف دوم، تکه زمانی 16 میکرو ثانیه داده می‌شود. همچنین صف سوم با روش FCFS زمان بندی می‌شود. میانگین زمان پاسخ و میانگین زمان انتظار چقدر خواهد بود؟

فرایند	زمان اجرا
A	4
B	7
C	12
D	20
E	25
F	30

حل:

ابتدا برنامه‌ها وارد صف اول می‌شوند و در صورت نیاز به بیش از ۸ میکروثانیه به صف دوم وارد می‌شوند (فرایندهای F,E,D,C) و در صف دوم در صورت نیاز به بیش از ۱۶ میکروثانیه به صف سوم منتقل می‌شوند (فرایندهای F,E) و در صف سوم به روش FCFS به آنها رسیدگی می‌شود:

A	B	C	D	E	F	C	D	E	F	E	F	
0	4	11	19	27	35	43	47	59	75	91	92	98

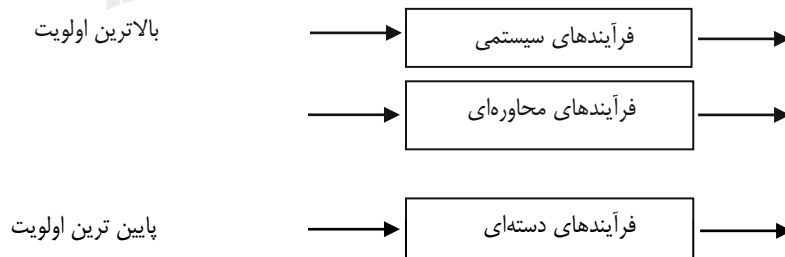
میانگین زمان انتظار برابر است با :

$$\frac{(4-4) + (11-7) + (47-12) + (59-20) + (92-25) + (98-30)}{6} = \frac{213}{6} = 35.5$$

زمان بندی صف چند سطحی (MLQ)

این الگوریتم صف آماده را به چند بخش تقسیم می‌کند. هر فرایند براساس صفات خود (اولویت، نوع) در صفی قرار می‌گیرد، که در هر صف الگوریتم زمان بندی خاصی وجود دارد. به طور معمول فرایندها به دو نوع ForeGround و BackGround تقسیم می‌شوند. اولویت فرایندهای پیش‌زمینه ممکن است از پس‌زمینه بالاتر باشد. در صف پیش‌زمینه از الگوریتم RR و در صف پس‌زمینه از الگوریتم FCFS استفاده می‌شود. همچنین بین صفها نیز باید زمان بندی وجود داشته باشد. که بر اساس الگوریتم غیرانحصاری و با اولویت ثابت پیاده‌سازی می‌شود.

شکل زیر الگوریتم زمان بندی صف چند سطحی با ۳ صف را نشان می‌دهد:



در این مثال، هیچ فرایندی در صف محاوره‌ای نمی‌تواند اجرا شود، مگر اینکه صف مربوط به فرایندهای سیستمی خالی باشد. همچنین اگر در حین اجرای فرایند دسته‌ای، یک فرایند محاوره‌ای وارد صف شود، فرایند دسته‌ای قبضه می‌شود، چون اولویت فرایند محاوره‌ای بالاتر است.

زمان بندی اولویت (Priority)

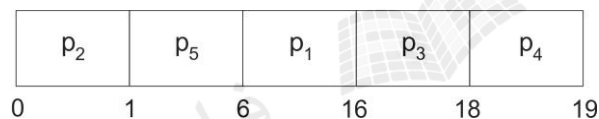
در این الگوریتم به هر فرایند اولییتی داده می‌شود و CPU به فرایندی داده می‌شود که بالاترین اولویت را دارد و فرایندهایی که اولویت آنها یکسان باشد به ترتیب FCFS زمان بندی می‌شوند.

مثال

میانگین زمان انتظار برای پنج فرایند زیر را در روش زمان بندی اولویت بدست آورید. (اعداد کوچک، اولویت بالا را نشان می‌دهد.)

اولویت	زمان اجرا	فرایند
3	10	P1
1	1	P2
3	2	P3
4	1	P4
2	5	P5

حل: گانت فرایندها به صورت زیر است:



میانگین زمان انتظار :

$$\frac{(16-10) + (1-1) + (18-2) + (19-1) + (6-5)}{4} = 8.2$$

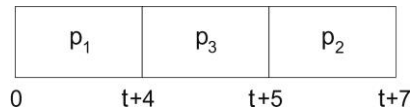


مثال

برای سه پردازنده زیر، زمان متوسط پاسخگویی در روش اولویت کدام است؟ (عدد کمتر در ستون اولویت، نشان دهنده اولویت بیشتر است.)

پردازنده	اولویت	زمان ورود	زمان اجرا
P ₁	1	t	4
P ₂	3	t	2
P ₃	2	t+3	1

حل: ابتدا فرایند P1 اجرا می شود (چون اولویت آن بیشتر است) و سپس فرایند P3 و در نهایت فرایند P2 اجرا می شود. گانت به صورت زیر می باشد:



بنابراین میانگین زمان پاسخگویی برابر است با :

$$= \frac{(t+4-t) + (t+7-t) + (t+5-t-3)}{3} = \frac{13}{3}$$



زمانبندی LCFS

در این روش، آخرین ورودی ابتدا سرویس می گیرد. این روش می تواند قبضه شدنی یا قبضه نشدنی باشد.

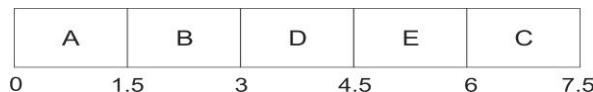
LCFS : Last-Come First-Served

مثال

پنج فرایند زیر به سیستمی وارد شده است. نمودار گانت را در صورت استفاده از الگوریتم LCFS انحصاری (NP-LCFS) را محاسبه کنید.

فرایند	زمان ورود	زمان پردازش
A	0	1.5
B	1	1.5
C	2	1.5
D	3	1.5
E	4	1.5

حل:



پردازنده در لحظه 0 به تنها فرایند موجود یعنی A داده می شود و چون قبضه شدنی نمی باشد، تا پایان اجرای فرایند A ، پردازنده را در اختیار دارد. در زمان 1.5 فقط فرایند B رسیده و پردازنده را می گیرد. در زمان 3 که C و D حاضرند، ابتدا پردازنده به D داده می شود، چون آخرین ورودی است. بعد از اجرای D ، چون E نیز رسیده است، پردازنده به E داده می شود و در نهایت به C داده می شود.



مثال

با توجه به جدول زیر، نمودار گانت را رسم کنید. (برای فرایندها از الگوریتم Round-Robin با برش زمانی $q=2$ و برای نخ های درون هر فرآیند، از الگوریتم LCFS استفاده کنید).

فرایند	نخ	زمان پردازش	زمان ورود
	T11	1.5	0
	T12	1.5	1
	T21	2.5	2
	T22	2	3

حل: نمودار گانت به صورت زیر می باشد:

T ₁₁	T ₁₂	T ₂₁	T ₂₂	T ₁₂	T ₁₁	T ₂₂	T ₂₁	
0	1	2	3	4	4.5	5	6	7.5

متوسط زمان بازگشت نخ های فرایند P1 برابر است با:

$$P1: \frac{(5-0) + (4.5-1)}{2} = 4.25$$



زمانبندی در سیستم چند پردازنده ای

الگوریتم های متداول برای زمانبندی سیستم های چند پردازنده ای عبارتند از:

LPT - ۱ RPT - ۲ SPT - ۳

الگوریتم LPT

این الگوریتم از بین کارهای باقیمانده، طولانی ترین کار را برای اجرا انتخاب می کند. این الگوریتم بهینه نیست ولی معمولاً منجر به زمانبندی هایی با طول معقول می گردد.

مثال

طول زمانبندی را در سیستم تکالیف {8,4,1,7,2,13,2,6} در حالت دو پردازنده محاسبه کنید.

حل: ابتدا کارها را به ترتیب نزولی مرتب می کنیم:

$$\{\tau_i\} = \{13, 8, 7, 6, 4, 2, 2, 1\}$$

سپس کارها را به ترتیب به پردازنده ها داده و هر پردازنده ای که کارش را انجام داد، کار بعدی را اجرا خواهد کرد:

P1	13	6	2	1
P2	8	7	4	2

بنابراین طول زمانبندی برابر ۲۲ خواهد بود. (زمان مشغول بودن پردازنده اول که بیشترین زمان است).

مثال

طول زمانبندی را در سیستم تکالیف $\{8,4,1,7,2,13,2,6\}$ در حالت سه پردازنده محاسبه کنید.

حل: ابتدا کارها را به ترتیب نزولی مرتب می‌کنیم:

$$\{\tau_i\} = \{13,8,7,6,4,2,2,1\}$$

سپس کارها را به ترتیب به پردازنده‌ها داده و هر پردازنده‌ای که کارش را انجام داد، کار بعدی را اجرا خواهد کرد:

P1	13	2	
P2	8	4	2
P3	7	6	1

بنابراین طول زمانبندی برابر ۱۵ خواهد بود. (زمان مشغول بودن پردازنده اول که بیشترین زمان است).

الگوریتم RPT

این الگوریتم از نظر طول زمانبندی مشابه LPT عمل کرده ولی زمان پاسخ بهتری را می‌دهد. در واقع RPT مانند LPT ای است که ترتیب تکالیف هر پردازنده معکوس شده است.

مثال

نتیجه استفاده از RPT برای $\{\tau_i\} = \{13,8,7,6,4,2,2,1\}$ در شرایط دو پردازنده را مشخص نمایید؟

حل: به روش LPT داریم:

P1	13	6	2	1
P2	8	7	4	2

که با معکوس کردن ترتیب تکالیف، خواهیم داشت

P1	1	2	6	13
P2	2	4	7	8

الگوریتم SPT

این الگوریتم چندپردازنده‌ای، ابتدا کارها را بر اساس زمان اجرای افزایشی مرتب کرده و سپس m تا کار اول را برای اجرا در m پردازنده موجود زمانبندی می‌کند (یک کار برای هر یک از پردازنده‌ها)، سپس m تا کار بعدی زمانبندی می‌گردند و بهمین ترتیب تا آخر.

مثال

طول زمانبندی در سیستم تکالیف {8,4,1,7,2,13,2,6} با دو پردازنده به روش SPT را تعیین نمایید.

حل: ابتدا کارها را به ترتیب صعودی مرتب می‌کنیم:

{1,2,2,4,6,7,8,13}

سپس کارها را به ترتیب به پردازنده‌ها داده و هر پردازنده‌ای که کارش را انجام داد، کار بعدی را اجرا خواهد کرد:

P1	1	2	6	8
P2	2	4	7	13

بنابراین طول زمانبندی برابر ۲۶ خواهد بود. (زمان مشغول بودن پردازنده دوم که بیشترین زمان است).

■




مشخصات سیاست‌های زمان بندی

FB	HRRN	SRT	SPN	RR	FCFS	
	$\max(\frac{w+s}{s})$	Min[s-e]	Min[s]	ثابت	MAX[W]	تابع انتخاب
با قبضه کردن (در برهه زمانی)	بدون قبضه کردن	با قبضه کردن در ورود	بدون قبضه کردن	با قبضه کردن (در برهه زمانی)	بدون قبضه کردن	حالت تصمیم گیری
تاکید نشده است.	زیاد	زیاد	زیاد	اگر برهه زمانی خیلی کوچک باشد، کم می شود.	تاکید نشده است	توان عملیاتی
تاکید نشده است	زمان پاسخ خوبی را ارائه می کند.	زمان پاسخ خوبی را ارائه می کند.	برای فرایندهای کوتاه زمان پاسخ خوبی را ارائه می دهد.	برای فرایندهای کوتاه زمان پاسخ خوبی را ارائه می دهد.	می تواند زیاد باشد، به خصوص اگر واریانس زمانهای اجرا خیلی بزرگ باشد.	زمان پاسخ
می تواند زیاد باشد	می تواند زیاد باشد	می تواند زیاد باشد	می تواند زیاد باشد	کم	حداقل	سربرار
می تواند به نفع فرایندهای در تنگنای I/O باشد.	توازن مناسب	به فرایندهای طولانی صدمه می زند.	به فرایندهای طولانی صدمه می زند.	عملکرد عادلانه	به فرایندهای کوتاه و فرایندهای در تنگنای I/O صدمه می زند.	تأثیر بر روی فرایندها
امکان دارد	خیر	امکان دارد	امکان دارد	خیر	خیر	گرسنگی

w = زمان سپری شده در سیستم برای انتظار و اجرا تا به حال

e = زمان سپری شده، برای اجرا تا به حال

S = کل زمان مورد نیاز فرایند، که شامل e نیز هست.

شرط اینکه یک سیستم بلادرنگ قابل زمان بندی باشد این است که : $\sum_{i=1}^m \frac{c_i}{p_i} \leq 1$ 

c_i : زمان اجرای رخداد i p_i : تناوب رخداد i m : تعداد رخدادها

کنکور ارشد

(مهندسی کامپیوتر - دولتی ۸۷)

۱- فرض کنید در سیستمی که از زمان بندی Round-Robin استفاده می کند، s زمان مورد نیاز برای سوئیچ کردن، q زمان برش و r میانگین زمان اجرای پردازش ها قبل از I/O را نشان می دهد. کارایی CPU توسط کدام یک از گزینه های زیر بیان می شود؟ (با فرض به اینکه رابطه $s=q < r$ برقرار باشد).

(از زمان موردنیاز برای سوئیچ کردن بین پردازش ها به دلیل I/O صرفنظر می شود)

(۱) به سمت صد در صد میل می کند. (۲) کمتر از 50 درصد می باشد.

(۳) به سمت صفر میل می کند. (۴) 50 درصد

پاسخ: جواب گزینه ۴ است.

$$\frac{q}{q+s} \times 100 = \frac{q}{q+q} \times 100 = \frac{1}{2} \times 100 = 50\%$$



(مهندسی کامپیوتر - آزاد ۸۷)

۲- در سیستمی 5 فرآیند موجود هستند. اگر الگوریتم زمان بندی فرآیندها، RR با مقدار کوانتوم 10 میلی ثانیه و زمان تعویض متن 1 میلی ثانیه باشد، آن گاه حداکثر زمانی که یک فرآیند منتظر می ماند تا نوبت به اجرای کوانتوم زمانی بعدی اش برسد کدام است؟

(۱) 40 (۲) 50 (۳) 55 (۴) 44

حل: جواب گزینه ۴ است. حداکثر زمان انتظار برای دریافت کوانتوم بعدی برابر است با:

$$(n-1)(s+q) = (5-1)(10+1) = 44$$

(مهندسی IT - آزاد ۸۹)

۳- سیستمی از روش زمانبندی نوبتی چرخشی استفاده می کند. اگر c زمان مورد نیاز برای تعویض متن، q برش زمانی (کوانتوم)، r میانگین زمان اجرای فرایندها قبل از I/O و $q > r$ باشد، کارایی CPU برابر است با:

$$\frac{q}{r+c} \quad (۴) \quad \frac{q}{q+c} \quad (۳) \quad \frac{r}{r+c} \quad (۲) \quad \frac{r}{q+c} \quad (۱)$$

حل: جواب گزینه ۲ است. چون میانگین زمان اجرای فرایندها قبل از ورودی/خروجی کمتر از زمان برش زمانی است، اجرای فرایندها قبل از پایان کوانتوم زمانی، به اتمام می رسد. بنابراین کارایی پردازنده، یعنی نسبت زمان مفید (r) به کل زمان

$$\frac{r}{r+c} \quad \text{برابر است با:}$$



(مهندسی کامپیوتر - دولتی ۷۴)

۴- پنج کار در وضعیت آماده، در انتظار اجرا شدن هستند. زمان تخمین زده شده برای اجرای آنها برابر است با 10, 5, 6, 8, x میکرو ثانیه. (x مجهول است). از کدام روش زمان بندی استفاده شود تا متوسط زمان پاسخگوئی حداقل شود؟

FCFS (۱) SJF (۲) SRT (۳) RR (۴)

حل: جواب گزینه ۲ است.

الگوریتم SJF همواره کمترین زمان پاسخ را نتیجه می‌دهد و به زمان اجراهای داده شده بستگی ندارد.

(مهندسی کامپیوتر - آزاد ۸۴)

۵- کدام گزینه در مورد الگوریتم زمان بندی SJF (Shortest Job First) درست نیست؟

(۱) این الگوریتم زمان برگشت را کاهش می‌دهد. (۲) توان عملیاتی (throughput) را بالا می‌برد.

(۳) این الگوریتم بر اساس اولویت عمل می‌کند. (۴) این الگوریتم بهره‌وری CPU را بالا می‌برد.

حل: جواب گزینه ۴ است.

گزینه ۱ درست است. چون الگوریتم SJF، نسبت به سایر الگوریتم‌های زمان بندی انحصاری، دارای میانگین زمان برگشت کمتری است.

گزینه ۲ درست است، چون در این الگوریتم کارهای کوتاه‌تر، زودتر اجرا می‌شوند، بنابراین تعداد کارهای انجام شده در واحد زمان (توان عملیاتی) بیشتر است.

گزینه ۳ درست است، چون الگوریتم SJF اولویت را به کارهای کوتاه‌تر می‌دهد، یک الگوریتم اولویت است.

گزینه ۴ نادرست است، چون SJF سعی به کم کردن تعداد تعویض متن‌ها ندارد، بنابراین بهره‌وری CPU را بالا نمی‌برد.

(مهندسی کامپیوتر - آزاد ۸۳)

۶- یک سیستم تک پردازنده‌ای از الگوریتم زمان بندی کوتاه‌ترین زمان باقی مانده (SRT) استفاده می‌نماید. چهار فرایند با زمان اجرای تخمینی 3,2,3,6 میلی ثانیه به ترتیب در زمانهای 7,3,1,0 میلی ثانیه وارد سیستم می‌شوند. اگر زمان تعویض متن ناچیز باشد و تمامی فرایندها فقط کار پردازشی داشته باشند، آنگاه میانگین زمان انتظار برای اجرای کامل فرایندها چند میلی ثانیه است؟

1.5 (۱) 0.25 (۲) 2.5 (۳) 2.25 (۴)

حل: جواب گزینه ۴ است.

نمودار گانت به صورت زیر می‌باشد:

A	B	C	A	D	A	
0	1	4	6	7	10	14

بنابراین میانگین زمان انتظار برابر است با:

$$\frac{(14-6-0) + (4-3-1) + (6-2-3) + (10-3-7)}{4} = \frac{8+0+1+0}{4} = 2.25$$



(مهندسی کامپیوتر-آزاد ۸۳)

۷- کدامیک از ویژگیهای زیر به عنوان ملاک الگوریتم زمان بندی صف‌های چندگانه (Multiple Queues) نیست؟

- (۱) افزایش گذردهی
 (۲) کاهش تعداد تعویض متن
 (۳) افزایش بهره‌وری از پردازنده
 (۴) اعمال اولویت (ابتدا کوتاه‌ترین فرایند)
- حل: جواب گزینه ۴ است.

در الگوریتم زمان بندی صف‌های چند گانه، اولویت لزوماً به معنای کوتاه‌ترین فرایند نمی باشد. ■

(مهندسی IT - آزاد ۸۹)

۸- چهار فرایند بر اساس جدول زیر وارد سیستم می شوند. در این سیستم از زمان بندی HRRN استفاده می شود. زمان تعویض متن یک میلی ثانیه است. میانگین زمان برگشت (TURNAROUND TIME) برابر است با:

فرایند	زمان ورود (میلی ثانیه)	زمان اجرا (میلی ثانیه)
P1	0	7
P2	2	3
P3	3	6
P4	3	5

13 (۴)

14 (۳)

13.75 (۲)

12.75 (۱)

پاسخ: جواب گزینه ۱ است.

در زمان صفر فقط فرایند P1 در سیستم وجود دارد و پردازنده به آن داده می شود تا اجرائیش به پایان برسد. در این زمان

(یعنی 7)، اولویت فرایندها طبق فرمول الگوریتم HRRN یعنی $\frac{W+S}{S}$ محاسبه می شود:

$$P2: \frac{(7-2)+3}{3} = 2.6$$

$$P3: \frac{(7-3)+6}{6} = 1.6$$

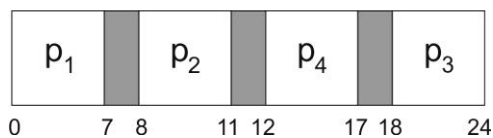
$$P4: \frac{(7-3)+5}{5} = 1.8$$

حال چون اولویت P2 از فرایندهای P3 و P4 بیشتر است، پردازنده در زمان 8 (بعد از یکی میلی ثانیه از پایان اجرای P1 به علت تویض متن) به آن داده می شود. اجرای P2 در لحظه 11 به پایان می رسد. در زمان 11، مجدداً اولویت فرایندهای باقی مانده را حساب می کنیم:

$$P3: \frac{(11-3)+6}{6} = 2.3$$

$$P4: \frac{(11-3)+5}{5} = 2.6$$

بنابراین پردازنده به P4 داده می‌شود. و در نهایت بعد از اجرای P4 به P3 داده می‌شود.
نمودار گانت به صورت زیر است:



بنابراین میانگین زمان برگشت برابر است با:

$$\frac{(7-0) + (11-2) + (17-3) + (24-3)}{4} = 12.75$$



منتخبی از عناوین آموزشی منتشر شده بر روی فرادرس

برنامه نویسی	
مدت زمان تقریبی	عنوان آموزش
۲ ساعت	آموزش اصول و مبانی برنامه نویسی - کلیک کنید (+)
۲۳ ساعت	آموزش برنامه نویسی جاوا - کلیک کنید (+)
۲۹ ساعت	آموزش برنامه نویسی PHP - کلیک کنید (+)
۲۰ ساعت	آموزش برنامه نویسی C++ - کلیک کنید (+)
۸ ساعت	آموزش پیشرفته C++ (شی گرایبی در سی پلاس پلاس) - کلیک کنید (+)
۸ ساعت	آموزش مقدماتی برنامه نویسی سی شارپ (C#) - کلیک کنید (+)
۱۴ ساعت	مجموعه آموزش های کاربردی برنامه نویسی C# (سی شارپ) - کلیک کنید (+)
۱۴ ساعت	آموزش شی گرایبی در سی شارپ (C#) - کلیک کنید (+)
۱۸ ساعت	آموزش برنامه نویسی پایتون - مقدماتی - کلیک کنید (+)
۵ ساعت	آموزش تکمیلی برنامه نویسی پایتون - کلیک کنید (+)
۱۳ ساعت	آموزش برنامه نویسی C - کلیک کنید (+)
۱۱ ساعت	آموزش SQL Server - مقدماتی - کلیک کنید (+)
۶ ساعت	آموزش SQL Server - تکمیلی - کلیک کنید (+)
۳ ساعت	آموزش کار با دستورات پایگاه داده در SQL Server - کلیک کنید (+)
۹ ساعت	مجموعه آموزش های برنامه نویسی متلب (MATLAB) - کلیک کنید (+)
۷ ساعت	مجموعه آموزش های برنامه نویسی متلب پیشرفته - کلیک کنید (+)
۱۶ ساعت	آموزش برنامه نویسی اندروید (Android) - مقدماتی - کلیک کنید (+)
۲۰ ساعت	آموزش برنامه نویسی اندروید (Android) - تکمیلی - کلیک کنید (+)
۲۶ ساعت	آموزش برنامه نویسی اندروید (Android) - پیشرفته - کلیک کنید (+)
۳ ساعت	آموزش پروژه محور برنامه نویسی دلفی (Delphi) - کلیک کنید (+)
۴ ساعت	آموزش زبان برنامه نویسی فرترن (Fortran) - مقدماتی - کلیک کنید (+)
۲ ساعت	آموزش مروری زبان برنامه نویسی FORTRAN با مثال های کاربردی - کلیک کنید (+)
۱۰ ساعت	آموزش پروژه محور اینترنت اشیا (IoT) - کنترل لوازم منزل با شبکه و پیامک و ماژول های SIM و ESP8266 - کلیک کنید (+)
۵ ساعت	آموزش برنامه نویسی Swift (سوئیفت) برای برنامه نویسی iOS - کلیک کنید (+)
۶ ساعت	آموزش برنامه نویسی پاسکال (Pascal) - کلیک کنید (+)
۱۶ ساعت	آموزش برنامه نویسی برای دانش آموزان با اسمال بیسیک (Small Basic) - کلیک کنید (+)
۶ ساعت	آموزش برنامه نویسی تصویری به کودکان با زبان اسکرچ (Scratch) - کلیک کنید (+)
۱ ساعت	آموزش محافظت از کدهای نرم افزاری با SmartAssembly (اسمارت اسمبلی) - کلیک کنید (+)

برنامه نویسی (ادامه از صفحه قبل)	
مدت زمان تقریبی	عنوان آموزش
۱۰ ساعت	آموزش مبانی برنامه نویسی (الگوریتم و فلوچارت) با رویکرد حل مسأله - کلیک کنید (+)
۱۰ ساعت	آموزش پروژه محور Python (پایتون) - ساخت نرم افزار برای Windows و Linux - کلیک کنید (+)
۵ ساعت	آموزش برنامه نویسی آردوینو (Arduino) با پایتون (Python) - کلیک کنید (+)
۲ ساعت	آموزش ساخت اپلیکیشن اندروید (Android) با PHP و MySQL - کلیک کنید (+)
۳ ساعت	آموزش مقدماتی زبان برنامه نویسی کاتلین (Kotlin) برای توسعه اندروید (Android) - کلیک کنید (+)
۱ ساعت	آموزش داکر (Docker) برای توسعه دهندگان - کلیک کنید (+)
۱ ساعت	آموزش توسعه وب با زبان برنامه نویسی Go - کلیک کنید (+)
۱۳ ساعت	آموزش ویژوال بیسیک دات نت (Visual Basic.NET) - مقدماتی - کلیک کنید (+)
۱۶ ساعت	آموزش ویژوال بیسیک دات نت (Visual Basic.NET) - تکمیلی - کلیک کنید (+)
۱ ساعت	آموزش تاریخچه گیت (Git) و مقدمات آن - کلیک کنید (+)
۱ ساعت	آموزش گیت (Git) برای مدیریت نسخه توزیع شده - کلیک کنید (+)
۵ ساعت	آموزش زبان برنامه نویسی AWK در لینوکس - پردازش و آنالیز فایل های متنی - کلیک کنید (+)
۷ ساعت	آموزش برنامه نویسی R و نرم افزار RStudio - کلیک کنید (+)
۵ ساعت	آموزش تکمیلی برنامه نویسی R و نرم افزار RStudio - کلیک کنید (+)
۱ ساعت	آموزش مقدماتی ساخت ربات تلگرام با PHP - کلیک کنید (+)
۹ ساعت	آموزش کتابخانه قالب استاندارد (STL) در C++ برای ساده نویسی برنامه ها - کلیک کنید (+)
۴ ساعت	آموزش برنامه نویسی با روش سه لایه به زبان VB.Net - کلیک کنید (+)
۴ ساعت	آموزش بازی سازی در پایتون با کتابخانه Pygame - کلیک کنید (+)
۵ ساعت	آموزش پایتون گرافیکی (رابط های گرافیکی پایتون) - کلیک کنید (+)
۱ ساعت	آموزش ساخت ربات تلگرام با پایتون (Python) - کلیک کنید (+)
۷ ساعت	مجموعه آموزش های طراحی رابط های گرافیکی (GUI) در متلب
۱۴ ساعت	آموزش تکمیلی طراحی رابط گرافیکی کاربر (GUI) با متلب (MATLAB) - کلیک کنید (+)
۱ ساعت	آموزش ساخت ربات تلگرام با متلب (MATLAB) - کلیک کنید (+)
۱ ساعت	آموزش ساخت ربات تلگرام با جاوا (Java) - کلیک کنید (+)
۲ ساعت	آموزش پایگاه داده ها در جاوا - کلیک کنید (+)
۳ ساعت	آموزش کار با فایل ها در جاوا با پکیج Java/IO (به همراه پروژه دفترچه یادداشت) - کلیک کنید (+)
۱ ساعت	آموزش پروژه محور جاوا - تولید QR code و ایجاد و اسکن بارکد (Barcode) - کلیک کنید (+)

برنامه نویسی (ادامه از صفحه قبل)	
مدت زمان تقریبی	عنوان آموزش
۶ ساعت	آموزش پروژه محور جاوا (Java) - طراحی و ساخت شبکه اجتماعی - کلیک کنید (+)
۴ ساعت	آموزش طراحی رابط کاربری مدرن در .NET با کامپوننت های جانوس (Janus) - کلیک کنید (+)
۲ ساعت	آموزش ساخت بازی ساده در ویژوال بیسیک - کلیک کنید (+)
۱۰ ساعت	آموزش پیاده سازی سیستم انبارداری و فروش با ویژوال بیسیک دات نت (VB.NET) - مقدماتی - کلیک کنید (+)
۱۲ ساعت	آموزش پیاده سازی سیستم انبارداری و فروش با ویژوال بیسیک دات نت (VB.NET) - تکمیلی - کلیک کنید (+)
۴ ساعت	آموزش مقدماتی پیاده سازی مسائل بهینه سازی در پایتون (Python) - کلیک کنید (+)
۱۲ ساعت	آموزش الگوهای طراحی (Design Patterns) در پایتون (Python) - کلیک کنید (+)
۲ ساعت	آموزش رایگان نکات و ترفندهای متلب - کلیک کنید (+)
۱۴ ساعت	مجموعه آموزش های برنامه نویسی متلب برای علوم و مهندسی - کلیک کنید (+)
۶ ساعت	آموزش فریم ورک Spring در جاوا - کلیک کنید (+)
۱۸ ساعت	آموزش ORM هایبرنیت (Hibernate) جاوا (Java) - کلیک کنید (+)
۲ ساعت	آموزش آشنایی با LINQ to SQL در C# - کلیک کنید (+)
۱ ساعت	آموزش برنامه نویسی تحت شبکه با سی شارپ در قالب پروژه - کلیک کنید (+)
۴ ساعت	آموزش برنامه نویسی با روش سه لایه به زبان سی شارپ - کلیک کنید (+)
۳ ساعت	آموزش Cryptography در دات نت - کلیک کنید (+)
۴ ساعت	آموزش قفل نرم افزاری در C# از طریق رجیستری - کلیک کنید (+)
۱۹ ساعت	آموزش پروژه محور سی شارپ (C#) - سیستم مدیریت آموزشگاه - کلیک کنید (+)
۱۴ ساعت	آموزش پروژه محور سی شارپ (C#) - سیستم مدیریت حسابداری و انبارداری - کلیک کنید (+)
۷ ساعت	آموزش برنامه نویسی سورس کدهای الگوریتمی در سی شارپ - کلیک کنید (+)
۱۰ ساعت	آموزش پروژه محور سی شارپ (C#) - پیاده سازی سیستم هتلداری - کلیک کنید (+)
۴ ساعت	آموزش ساخت سیستم آزمون گیر در سی شارپ - کلیک کنید (+)
۳ ساعت	آموزش ارسال و دریافت پیامک با مودم در سی شارپ (C#) - کلیک کنید (+)
۱۱ ساعت	آموزش پروژه محور سی شارپ (C#) - پیاده سازی سیستم مدیریت آتلیه - کلیک کنید (+)
۷ ساعت	آموزش پروژه محور سی شارپ (C#) - پیاده سازی سیستم نوبت دهی - کلیک کنید (+)
۹ ساعت	آموزش پروژه محور سی شارپ (C#) - پیاده سازی سیستم صدور بلیط - کلیک کنید (+)
۳ ساعت	آموزش رسم نمودار در سی شارپ (C#) - کلیک کنید (+)
۷ ساعت	آموزش پروژه محور سی شارپ (C#) - سیستم مدیریت بانک - کلیک کنید (+)
۵ ساعت	آموزش ساخت بازی با سی شارپ (C#) - کلیک کنید (+)
۲ ساعت	آموزش Metaprogramming (برنامه نویسی متا) در پایتون - کلیک کنید (+)

برنامه نویسی (ادامه از صفحه قبل)	
مدت زمان تقریبی	عنوان آموزش
۱۲ ساعت	آموزش کاربردی Entity Framework در سی شارپ (C#) - کلیک کنید (+)
۶ ساعت	آموزش پروژه محور سی شارپ (C#) - پیاده سازی سیستم مدیریت کارمندان - کلیک کنید (+)
۶ ساعت	آموزش پروژه محور سی شارپ (C#) - پیاده سازی سیستم مدیریت چک های بانکی - کلیک کنید (+)
۴ ساعت	آموزش پروژه محور سی شارپ (C#) - شبیه سازی دستگاه عابر بانک - کلیک کنید (+)
۲ ساعت	آموزش گزارش گیری با کریستال ریپورت و استیمول سافت (Stimulsoft) در سی شارپ (C#) - کلیک کنید (+)
۲ ساعت	آموزش پایگاه داده اس کیو لایت (SQLite) در سی شارپ (C#) - کلیک کنید (+)
۶ ساعت	آموزش پروژه محور سی شارپ (C#) - پیاده سازی نرم افزار حسابداری شخصی - کلیک کنید (+)
۱۵ ساعت	آموزش پروژه محور سی شارپ (C#) - پیاده سازی سیستم رستوران آنلاین - کلیک کنید (+)
۳ ساعت	آموزش کار با بانک اطلاعاتی اکسس (Access) در سی شارپ (C#) - کلیک کنید (+)
۲ ساعت	آموزش کار با فایل ها و دایرکتوری در C# (سی شارپ) - کلیک کنید (+)
۳ ساعت	آموزش پروژه محور سی شارپ C# - پیاده سازی نرم افزار FeedReader (فیدریدر) - کلیک کنید (+)
۴ ساعت	آموزش کار با دستورات گرافیکی در سی شارپ (C#) - کلیک کنید (+)
۳ ساعت	آموزش مقدماتی WPF در سی شارپ (C#) برای ساخت و اجرای اینترفیس - کلیک کنید (+)
۴ ساعت	آموزش WPF در سی شارپ برای ساخت و اجرای اینترفیس - پیشرفته - کلیک کنید (+)
۲ ساعت	آموزش WPF در سی شارپ (C#) برای ساخت و اجرای اینترفیس - تکمیلی - کلیک کنید (+)
۱ ساعت	آموزش ساخت ربات تلگرام با سی شارپ (C#) - کلیک کنید (+)
۱۲ ساعت	آموزش پروژه محور پایگاه داده LocalDB در سی شارپ (C#) - سیستم مدیریت کارمندان - کلیک کنید (+)
۵ ساعت	آموزش کتابخانه Spire.Doc در سی شارپ (C#) برای ایجاد و ویرایش فایل Word - کلیک کنید (+)
۱ ساعت	آموزش پروژه محور سی شارپ (C#) - ساخت برنامه دانلودگر - کلیک کنید (+)

علوم کامپیوتر	
مدت زمان تقریبی	عنوان آموزش
۱۰ ساعت	آموزش ساختمان داده ها - کلیک کنید (+)
۸ ساعت	آموزش پیشرفته ساختمان داده (همراه با حل نمونه سوالات کنکور ارشد و دکتری) - کلیک کنید (+)
۲۰ ساعت	آموزش ساختمان داده ها (مرور - تست کنکور ارشد) - کلیک کنید (+)
۱۱ ساعت	آموزش سیستم های عامل - کلیک کنید (+)
۱۲ ساعت	آموزش سیستم عامل (مرور - تست کنکور ارشد) - کلیک کنید (+)
۸ ساعت	آموزش پایگاه داده ها - کلیک کنید (+)
۸ ساعت	آموزش پایگاه داده ها (مرور - تست کنکور ارشد) - کلیک کنید (+)
۱۰ ساعت	آموزش طراحی و پیاده سازی زبان های برنامه سازی - کلیک کنید (+)
۹ ساعت	آموزش طراحی و پیاده سازی زبان های برنامه سازی (مرور - تست کنکور ارشد) - کلیک کنید (+)
۱۵ ساعت	آموزش طراحی الگوریتم - کلیک کنید (+)
۸ ساعت	آموزش طراحی الگوریتم به همراه حل مثال های عملی - کلیک کنید (+)
۴ ساعت	آموزش روش های حل روابط بازگشتی - کلیک کنید (+)
۲ ساعت	آموزش رابطه های بازگشتی در طراحی الگوریتم و ساختمان گسسته (مرور - تست کنکور ارشد) - کلیک کنید (+)
۱۵ ساعت	آموزش طراحی الگوریتم (مرور - تست کنکور ارشد) - کلیک کنید (+)
۲ ساعت	آموزش روش تقسیم و حل در طراحی الگوریتم (مرور - تست کنکور کارشناسی ارشد) - کلیک کنید (+)
۱۸ ساعت	آموزش مبانی منطق و نظریه مجموعه ها - کلیک کنید (+)
۱۵ ساعت	آموزش مدار منطقی (مرور - تست کنکور ارشد) - کلیک کنید (+)
۱۲ ساعت	آموزش مبانی الکترونیک دیجیتال - بخش اول - کلیک کنید (+)
۱۱ ساعت	آموزش مبانی الکترونیک دیجیتال - بخش دوم - کلیک کنید (+)
۲۰ ساعت	آموزش معماری کامپیوتر با رویکرد حل مساله - مقدماتی - کلیک کنید (+)
۱۴ ساعت	آموزش معماری کامپیوتر با رویکرد حل مساله - پیشرفته - کلیک کنید (+)
۹ ساعت	آموزش نظریه زبان ها و ماشین ها - کلیک کنید (+)
۸ ساعت	آموزش نظریه زبان ها و ماشین (مرور - تست کنکور ارشد) - کلیک کنید (+)
۳ ساعت	آموزش شبیه سازی و تست نظریه زبان ها و ماشین در JFLAP - کلیک کنید (+)
۴ ساعت	آموزش یادگیری عمیق در پزشکی - کلیک کنید (+)
۱۶ ساعت	آموزش ساختمان گسسته با رویکرد حل مساله - کلیک کنید (+)
۱۲ ساعت	آموزش ساختمان گسسته (مرور و حل تست های کنکور کارشناسی ارشد) - کلیک کنید (+)
۱۳ ساعت	آموزش آمار و احتمال مهندسی - کلیک کنید (+)
۱۴ ساعت	آموزش آمار و احتمال مهندسی (حل تمرین و تست کنکور ارشد) - کلیک کنید (+)

علوم کامپیوتر (ادامه از صفحه قبل)	
مدت زمان تقریبی	عنوان آموزش
۱۹ ساعت	آموزش شبکه های کامپیوتری ۱ - کلیک کنید (+)
۲۳ ساعت	آموزش شبکه های کامپیوتری ۲ - همراه با حل مسأله - کلیک کنید (+)
۱ ساعت	آموزش شبیه سازی شبکه های مبتنی بر سیستم (Cisco) در نرم افزار Boson NetSim - کلیک کنید (+)
۱۵ ساعت	آموزش طراحی کامپایلر - کلیک کنید (+)
۱۵ ساعت	آموزش طراحی کامپایلر (مرور و حل تست های کنکور کارشناسی ارشد) - کلیک کنید (+)
۸ ساعت	آموزش ذخیره و بازیابی اطلاعات - کلیک کنید (+)
۲۳ ساعت	آموزش ساختمان داده ها همراه با پیاده سازی در C++ - کلیک کنید (+)
۹ ساعت	آموزش انتقال داده ها در شبکه های کامپیوتری و اینترنت به همراه حل مسائل - کلیک کنید (+)
۵ ساعت	آموزش مقدماتی Hadoop (هدوپ) برای تجزیه و تحلیل کلان داده - کلیک کنید (+)
۲۲ ساعت	آموزش نظریه صف (Queueing theory) - کلیک کنید (+)
۴ ساعت	آموزش گرافیک کامپیوتری - کلیک کنید (+)
۱۶ ساعت	آموزش گرافیک کامپیوتری با OpenGL - کلیک کنید (+)
۱۲ ساعت	آموزش مهندسی نرم افزار ۱ - کلیک کنید (+)
۱۰ ساعت	آموزش زبان تخصصی مهندسی کامپیوتر - کلیک کنید (+)
۱ ساعت	آموزش مروری بر پیچیدگی محاسبات (Computational Complexity) - کلیک کنید (+)
۱ ساعت	آموزش تبدیل فلوچارت به کد با Flowgorithm - کلیک کنید (+)
۱ ساعت	آموزش نرم افزار RAPTOR برای ترسیم فلوچارت - کلیک کنید (+)
۲ ساعت	آموزش حل سوالات کنکور کارشناسی ارشد کامپیوتر - ۹۴ - کلیک کنید (+)
۱۷ ساعت	آموزش مرور و حل تست های کنکور کارشناسی ارشد معماری کامپیوتر - کلیک کنید (+)
۱۰ ساعت	آموزش برنامه نویسی اسمبلی - کلیک کنید (+)
۱۵ ساعت	آموزش نظریه گراف و کاربردها - کلیک کنید (+)
۱۵ ساعت	آموزش ترکیبیات و کاربردها - کلیک کنید (+)
۸ ساعت	آموزش نظریه بازی ها و کاربرد آن در اقتصاد - کلیک کنید (+)
۴ ساعت	آموزش مدل سازی UML با نرم افزار Rational Rose - کلیک کنید (+)
۱۳ ساعت	آموزش الگوریتم موازی و پردازش موازی - کلیک کنید (+)
۲۰ ساعت	آموزش ترکیب اطلاعات (Information Fusion) - کلیک کنید (+)
۳ ساعت	آموزش نهان نگاری دیجیتال (Digital watermarking) - کلیک کنید (+)
۳ ساعت	آموزش نهان نگاری و پیاده سازی آن در سی شارپ (C#) - کلیک کنید (+)
۱ ساعت	آموزش واترمارکینگ در نهان نگاری با سی شارپ (C#) - کلیک کنید (+)
۱ ساعت	آموزش شبکه های پیچیده پویا (Complex Dynamical Networks) - کلیک کنید (+)

علوم کامپیوتر (ادامه از صفحه قبل)	
مدت زمان تقریبی	عنوان آموزش
۳ ساعت	آموزش تخمین تلاش لازم برای توسعه نرم افزارها با متلب - کلیک کنید (+)
۴ ساعت	آموزش توسعه نرم افزار با متد ICONIX و زبان مدل سازی UML- کلیک کنید (+)
۱ ساعت	آموزش نگارش پروپوزال، پایان نامه و ارائه - ویژه پردازش تصویر و بینایی کامپیوتر - کلیک کنید (+)

هوش مصنوعی	
مدت زمان تقریبی	عنوان آموزش
۴ ساعت	آموزش هوش مصنوعی - کلیک کنید (+)
۱۹ ساعت	آموزش هوش مصنوعی - تکمیلی - کلیک کنید (+)
۲۴ ساعت	آموزش هوش مصنوعی توزیع شده - کلیک کنید (+)
۱۵ ساعت	آموزش هوش مصنوعی (مرور و حل تست کنکور ارشد) - کلیک کنید (+)
۲۸ ساعت	مجموعه آموزش های شبکه های عصبی مصنوعی در متلب - کلیک کنید (+)
۱۰ ساعت	مجموعه آموزش های کاربردی شبکه های عصبی مصنوعی - کلیک کنید (+)
۲ ساعت	آموزش طراحی سیستم های فازی عصبی یا ANFIS با استفاده از الگوریتم های فراابتکاری و تکاملی - کلیک کنید (+)
۵ ساعت	آموزش کاربردی مکترونیک - محاسبات پایه طراحی درب های خودکار - کلیک کنید (+)
۴ ساعت	آموزش شبکه عصبی GMDH به همراه پیاده سازی عملی در متلب - کلیک کنید (+)
۳ ساعت	آموزش شبکه های عصبی گازی به همراه پیاده سازی عملی در متلب - کلیک کنید (+)
۳ ساعت	آموزش طبقه بندی و بازشناسی الگو با شبکه های عصبی LVQ در متلب - کلیک کنید (+)
۳ ساعت	آموزش یادگیری عمیق (Deep learning) - کلیک کنید (+)
۳ ساعت	آموزش برنامه نویسی یادگیری عمیق با پایتون (TensorFlow و Keras) - کلیک کنید (+)
۲۱ ساعت	مجموعه آموزش های سیستم های فازی در متلب - کلیک کنید (+)
۱۴ ساعت	مجموعه آموزش های تئوری و عملی الگوریتم ژنتیک - کلیک کنید (+)
۱ ساعت	آموزش پیاده سازی ترکیب الگوریتم ژنتیک و PSO در متلب - کلیک کنید (+)
۱۹ ساعت	مجموعه آموزش های بهینه سازی چند هدفه در متلب - کلیک کنید (+)
۲ ساعت	آموزش حل مساله فروشنده دوره گرد با استفاده از الگوریتم ژنتیک - کلیک کنید (+)
۲ ساعت	آموزش انتخاب ویژگی یا Feature Selection - کلیک کنید (+)
۴ ساعت	آموزش انتخاب ویژگی با استفاده از الگوریتم های فراابتکاری و تکاملی - کلیک کنید (+)
۲ ساعت	آموزش تخمین خطای طبقه بندی یا Classifier Error Estimation - کلیک کنید (+)
۱ ساعت	آموزش کاهش تعداد رنگ تصاویر با استفاده از روش های خوشه بندی هوشمند - کلیک کنید (+)

مهندسی برق	
مدت زمان تقریبی	عنوان آموزش
۱۱ ساعت	آموزش الکترونیک ۱ - کلیک کنید (+)
۲۵ ساعت	آموزش حل تمرین الکترونیک ۱ - کلیک کنید (+)
۱۱ ساعت	آموزش الکترونیک ۱ (مرور و حل تست) - کلیک کنید (+)
۸ ساعت	آموزش الکترونیک ۲ - کلیک کنید (+)
۲۵ ساعت	آموزش حل تمرین الکترونیک ۲ - کلیک کنید (+)
۱۲ ساعت	آموزش الکترونیک ۲ (مرور و حل تست) - کلیک کنید (+)
۱۵ ساعت	آموزش الکترونیک ۳ - کلیک کنید (+)
۱۲ ساعت	آموزش الکترونیک ۳ (مرور و حل مساله) - کلیک کنید (+)
۱۴ ساعت	آموزش مبانی مهندسی برق ۱ - کلیک کنید (+)
۱۴ ساعت	آموزش مبانی مهندسی برق ۱ (مرور و حل مساله) - کلیک کنید (+)
۱۶ ساعت	آموزش مبانی مهندسی برق ۲ - کلیک کنید (+)
۲۴ ساعت	آموزش مدارهای الکتریکی ۱ - کلیک کنید (+)
۱۷ ساعت	آموزش حل تمرین مدارهای الکتریکی ۱ - کلیک کنید (+)
۱۴ ساعت	آموزش مدارهای الکتریکی ۱ (مرور و حل تست) - کلیک کنید (+)
۱۱ ساعت	آموزش مدارهای الکتریکی ۲ - کلیک کنید (+)
۱۸ ساعت	آموزش مدارهای الکتریکی ۲ (حل تمرین) - کلیک کنید (+)
۱۳ ساعت	آموزش مدارهای الکتریکی ۲ (مرور و حل تست) - کلیک کنید (+)
۹ ساعت	آموزش هندسه فینسلر (Finsler Geometry) - کلیک کنید (+)
۱۶ ساعت	آموزش زبان تخصصی مهندسی برق (همراه با حل نمونه سوالات کنکور ارشد) - کلیک کنید (+)
۲ ساعت	آموزش مروری ماشین های الکتریکی ۱ - کلیک کنید (+)
۱۴ ساعت	آموزش ماشین های الکتریکی ۱ - کلیک کنید (+)
۱۶ ساعت	آموزش ماشین های الکتریکی ۱ (مرور و حل تست) - کلیک کنید (+)
۱۷ ساعت	آموزش ماشین های الکتریکی ۲ - کلیک کنید (+)
۱۳ ساعت	آموزش ماشین های الکتریکی ۲ (مرور و حل تست کنکور ارشد) - کلیک کنید (+)
۱۱ ساعت	آموزش ماشین های الکتریکی ۳ - کلیک کنید (+)
۹ ساعت	آموزش مدارهای مخبراتی - کلیک کنید (+)
۹ ساعت	آموزش ماشین های الکتریکی مخصوص - کلیک کنید (+)
۱۴ ساعت	آموزش کنترل موتورهای الکتریکی صنعتی ۱ - کلیک کنید (+)
۱۷ ساعت	آموزش کنترل موتورهای الکتریکی صنعتی ۲ - کلیک کنید (+)
۱۵ ساعت	آموزش کنترل صنعتی - حل مساله - کلیک کنید (+)

مهندسی برق (ادامه از صفحه قبل)	
مدت زمان تقریبی	عنوان آموزش
۹ ساعت	مجموعه آموزش های مدارهای منطقی (طراحی دیجیتال) - کلیک کنید (+)
۱۲ ساعت	آموزش مبانی الکترونیک دیجیتال - بخش اول - کلیک کنید (+)
۱۱ ساعت	آموزش مبانی الکترونیک دیجیتال - بخش دوم - کلیک کنید (+)
۲۰ ساعت	آموزش الکترومغناطیس مهندسی - کلیک کنید (+)
۱۸ ساعت	آموزش الکترومغناطیس مهندسی (مرور و حل تست کنکور ارشد) - کلیک کنید (+)
۹ ساعت	آموزش الکترومغناطیس ۱ - کلیک کنید (+)
۳ ساعت	آموزش مقدماتی شبکه های هوشمند (Smart grid) انرژی الکتریکی - کلیک کنید (+)
۵ ساعت	آموزش کاربردی میکاترونیک - محاسبات پایه طراحی درب های خودکار - کلیک کنید (+)
۲ ساعت	آموزش کامسول مولتی فیزیکس - COMSOL Multiphysics (مباحث منتخب) - کلیک کنید (+)
۴ ساعت	آموزش نرم افزار COMSOL Multiphysics برای پدیده های الکترومغناطیس - کلیک کنید (+)
۸ ساعت	آموزش شبیه سازی مدولاسیون دیجیتال در متلب (MATLAB) - کلیک کنید (+)
۴ ساعت	آموزش نرم افزار سیلواکو (SILVACO) برای شبیه سازی افزاره های نیمه رسانا - کلیک کنید (+)

ابزار مدیریت وبسایت	
مدت زمان تقریبی	عنوان آموزش
۱ ساعت	آموزش کار با cPanel - کلیک کنید (+)
۱ ساعت	آموزش کار با کنترل پنل plesk برای ارائه خدمات وب - کلیک کنید (+)
۱ ساعت	آموزش مدیریت هاست با دایرکت ادمین (DirectAdmin) - کلیک کنید (+)
۳ ساعت	آموزش جوملا (Joomla) برای راه اندازی و مدیریت وب سایت - کلیک کنید (+)
۱ ساعت	آموزش نرم افزار Mobirise برای ساخت وب سایت بدون کدنویسی - کلیک کنید (+)
۸ ساعت	آموزش مقدماتی Oracle APEX برای تولید و توسعه نرم افزارهای تحت وب - کلیک کنید (+)
۱۱ ساعت	آموزش وردپرس (WordPress) - مقدماتی - کلیک کنید (+)
۶ ساعت	مجموعه آموزش های راه اندازی سایت و کار با وردپرس - تکمیلی - کلیک کنید (+)
۳ ساعت	آموزش گوگل آنالیتیکس (Google Analytics) برای تحلیل آمار وب سایت - کلیک کنید (+)
۲ ساعت	آموزش آنالیز و بهینه سازی سایت با GTmetrix - کلیک کنید (+)
۱۰ ساعت	آموزش تست نفوذ در وب و راه های مقابله با آن - کلیک کنید (+)

طراحی و گرافیک	
مدت زمان تقریبی	عنوان آموزش
۱۲ ساعت	آموزش فتوشاپ (Photoshop) - مقدماتی - کلیک کنید (+)
۱۴ ساعت	مجموعه آموزش های کاربردی طراحی و گرافیک با نرم افزار کورل (CorelDRAW) - کلیک کنید (+)
۱۰ ساعت	آموزش کورل پیشرفته - کلیک کنید (+)
۱۰ ساعت	آموزش نرم افزار طراحی گرافیکی Adobe Illustrator - کلیک کنید (+)
۵ ساعت	آموزش نرم افزار طراحی گرافیکی Adobe Illustrator - تکمیلی - کلیک کنید (+)
۱ ساعت	آموزش مقایسه نرم افزارهای گرافیکی Photoshop , CorelDRAW , Illustrator - کلیک کنید (+)
۲ ساعت	آموزش مقدماتی طراحی کاراکتر در Adobe Illustrator (کاراکتر پلیس) - کلیک کنید (+)
۸ ساعت	آموزش نرم افزار Blender برای طراحی و مدل سازی سه بعدی - کلیک کنید (+)
۱۴ ساعت	آموزش مدل سازی کاراکتر در بلندر - ایده پردازی تا ریگ - کلیک کنید (+)
۱ ساعت	آموزش طراحی فلت (Flat design) با کورل (CorelDRAW) - کلیک کنید (+)
۳ ساعت	آموزش فتوشاپ برای استفاده در صنعت چاپ و تبلیغات - کلیک کنید (+)
۳ ساعت	آموزش تایپوگرافی (Typography) - سایه زدن حروف در فتوشاپ (Photoshop) - کلیک کنید (+)
۱ ساعت	آموزش پروژه محور فتوشاپ (Photoshop) - ساخت ماسک (Mask) فانتزی - کلیک کنید (+)
۲ ساعت	آموزش آشنایی با پنل Adjustments در فتوشاپ - کلیک کنید (+)
۳ ساعت	آموزش ساخت جلوه های ویژه در فتوشاپ - کلیک کنید (+)
۶ ساعت	آموزش Adobe Lightroom (ادوبی لایت روم) برای ویرایش و روتوش تصاویر - کلیک کنید (+)
۲ ساعت	آموزش Adobe Dimension برای ایجاد تصاویر سه بعدی محصولات و تبلیغات - کلیک کنید (+)
۲ ساعت	آموزش Adobe Bridge (ادوبی بریج) برای مدیریت فایل های چندرسانه ای - کلیک کنید (+)
۴ ساعت	آموزش ایندیزاین (InDesign) برای چاپ و صفحه آرایی کتاب و مجلات - کلیک کنید (+)
۲۲ ساعت	آموزش نرم افزار SketchBook Pro (اسکچ بوک) برای طراحی آناتومی حیوانات - کلیک کنید (+)
۳ ساعت	آموزش طراحی مفهومی (Conceptual) خودرو با SketchBook - کلیک کنید (+)
۳ ساعت	آموزش نقاشی سه بعدی با نرم افزار Paint 3D ویندوز - کلیک کنید (+)
۳ ساعت	آموزش مبانی هنرهای تجسمی ۱ - کلیک کنید (+)
۱ ساعت	آموزش اصول طراحی نشانه (Sign Design) - کلیک کنید (+)
۹ ساعت	آموزش نقاشی دیجیتال با Photoshop و Mischieff - کلیک کنید (+)
۲ ساعت	آموزش طراحی لوگو با Adobe Illustrator - کلیک کنید (+)
۳ ساعت	آموزش طراحی محیط بازی های دو بعدی با نرم افزار Adobe Illustrator و Photoshop - کلیک کنید (+)

طراحی و گرافیک (ادامه از صفحه قبل)	
مدت زمان تقریبی	عنوان آموزش
۱ ساعت	آموزش طراحی لوگو - آشنایی با مفاهیم و ایده پردازی (همراه با مثال عملی) - کلیک کنید (+)
۲ ساعت	آموزش طراحی برچسب و کاور CD و DVD - کلیک کنید (+)
۵ ساعت	آموزش ابزار سه بعدی در فتوشاپ (Photoshop) - کلیک کنید (+)
۲ ساعت	آموزش فتوشاپ سه بعدی - طراحی و متحرک سازی - کلیک کنید (+)
۲ ساعت	آموزش موتور رندر آرنولد (Arnold Renderer) برای ایجاد جلوه های ویژه در DS Max ^۳ - کلیک کنید (+)
۵ ساعت	آموزش کاربرد فتوشاپ در معماری - کلیک کنید (+)

روباتیک	
مدت زمان تقریبی	عنوان آموزش
۲ ساعت	آموزش رباتیک و ربات های سری - کلیک کنید (+)
۹ ساعت	آموزش برنامه نویسی آردوینو (Arduino) با محوریت پروژه های رباتیک - مقدماتی - کلیک کنید (+)
۷ ساعت	آموزش برنامه نویسی آردوینو (Arduino) - تکمیلی - کلیک کنید (+)
۳ ساعت	آموزش سینماتیک مستقیم و معکوس روبات ها - کلیک کنید (+)
۵ ساعت	آموزش مبانی ربات های برنامه پذیر - کلیک کنید (+)
۱۶ ساعت	آموزش شبیه سازی و کنترل ربات با اندروید (Android) - کلیک کنید (+)
۱۳ ساعت	آموزش مکترونیک کاربردی ۱ - کلیک کنید (+)
۷ ساعت	آموزش مکترونیک کاربردی ۲ - کلیک کنید (+)
۵ ساعت	آموزش کاربردی مکترونیک - محاسبات پایه طراحی درب های خودکار - کلیک کنید (+)
۱۹ ساعت	آموزش برد آردوینو (Arduino) با انجام پروژه های عملی - کلیک کنید (+)
۵ ساعت	آموزش پروژه محور آردوینو با سی شارپ و اندروید - کنترل موتور پله ای - کلیک کنید (+)
۵ ساعت	آموزش پیاده سازی سیستم های کنترلی با آردوینو - کلیک کنید (+)

فصل ۴:

هم‌روندی (انحصار متقابل و همگام‌سازی)

عملکرد چند برنامه‌ای به خاطر این ابداع شد که زمان پردازش کامپیوتر به صورت پویا بین تعدادی کار بتواند تقسیم گردد. برای زمینه‌های چند برنامه‌ای، چند پردازشی و پردازش توزیعی و همچنین طراحی سیستم عامل، موضوع همزمانی اساسی است. فرایندهای هم‌رند (Concurrent)، به دفعات نیاز به برقراری ارتباط با یکدیگر دارند. این فرایندها نیاز به هماهنگی، تبادل داده و استفاده از منابع مشترک دارند.

مباحث مطرح در ارتباط بین فرایندها

در طراحی سیستم عامل، سه موضوع زیر در رابطه با ارتباط بین فرایندها مطرح است:

۱- همگام‌سازی (Synchronization)

اگر بین فرایندها وابستگی وجود داشته باشد، ترتیب درست انجام کارها باید رعایت شود.

۲- تبادل اطلاعات (Communication)

فرایندها می‌توانند با مکانیسم‌هایی چون "حافظه مشترک، تبادل پیام، فایل مشترک و لوله" با یکدیگر تبادل اطلاعات کنند.

۳- رقابت فرایندها

فرایندها در فعالیت‌های بحرانی یکدیگر مداخله نکنند و شرایط رقابتی (Race Condition) برای آنها رخ ندهد.

حالت‌های ممکن ارتباط بین فرایندها (IPC)

۱- فرایندهایی که به طور مستقیم با یکدیگر تبادل داده و همکاری دارند.

۲- فرایندهایی که غیر مستقیم با یکدیگر تبادل داده و همکاری دارند.

۳- فرایندهایی که هیچ اطلاعی از یکدیگر ندارند.

سه مسئله کنترلی

در مورد فرایندهای رقیب، با سه مسئله کنترلی زیر باید برخورد شود:

۱- انحصار متقابل

۲- بن بست

۳- گرسنگی (قحطی)

انحصار متقابل

فرض کنید چند فرایند برای دسترسی به یک منبع غیر اشتراکی مانند چاپگر رقابت می کنند. در طی اجراء، هر یک از فرایندها، فرمان هایی را به دستگاه ورودی/ خروجی ارسال می کنند. چنین منبعی را منبع بحرانی و بخشی از برنامه که از آن استفاده می کند را بخش بحرانی آن برنامه می گوئیم. مهم این است که در یک زمان، تنها یک برنامه مجاز است تا در بخش بحرانی خود باشد.

بخش هایی از برنامه که رفتار آنها با عوامل مشترک، ایجاد رقابت می کنند، را ناحیه بحرانی (Critical Region) می گویند.

بن بست

اعمال انحصار متقابل دو مسئله کنترلی، بن بست و گرسنگی را به وجود می آورد. دو فرایند P1 و P2 و دو منبع بحرانی R1 و R2 مفروض است که هر یک از فرایندها برای انجام عمل خود به هر دو منبع نیاز دارند. اگر منبع R1 به P2 و منبع R2 به P1 داده شود، هر یک منتظر منبع دیگر می باشند و هیچ یک از فرایندها، منبعی را که در اختیار دارد را رها نمی کند تا فرایند دیگر آن را دریافت کرده و بخش بحرانی خود را انجام دهد. بنابراین هر دو فرایند در بن بست قرار می گیرند.

گرسنگی

فرض کنید هر یک از سه فرایند P1 و P2 و P3 متناوباً نیازمند دسترسی به منبع R هستند. وقتی P1 این منبع را در اختیار گیرد، P2 و P3 در انتظار آن منبع، به تاخیر انداخته می شوند. با خروج P1 از ناحیه بحرانی، یکی از P2 یا P3 باید R را در اختیار گیرند. اگر P3 منبع R را بگیرد و قبل از پایان بخش بحرانی مجدداً P1 درخواست R را بکند و بعد از پایان P3، اجازه به P1 داده شود و مکرراً این عمل بین P1 و P3 ادامه یابد، P2 به صورت نامحدود از دسترسی به منبع R محروم می ماند و گرسنگی می کشد.

مثال

در صورت اجرای هم روند و موازی دو پروسس زیر، چه خروجی هایی ممکن می باشد؟

p1	p2
cout<< 1; cout<< 2;	cout<< 3; cout<< 4;

پاسخ:

امکان تولید خروجی های زیر می باشد:

۱- 1234 : ابتدا P1 به طور کامل اجرا شده و سپس P2 اجرا می شود.

۲- 3412 : ابتدا P2 به طور کامل اجرا شده و سپس P1 اجرا می شود.

- ۳- 1342 : ابتدا دستور اول در P1 اجرا شده و سپس P2 به طور کامل اجرا شده و در نهایت دستور دوم P1 اجرا می شود.
- ۴- 3124 : ابتدا دستور اول از P2 ، بعد اجرای کامل P1 ، مجددا دستور دوم از P2 اجرا شود.
- ۵- 1324 : ابتدا دستور اول از P1، بعد دستور اول از P2، مجددا دستور دوم از P1 و در نهایت دستور دوم از P2 اجرا شود.
- ۶- 3142 : ابتدا دستور اول از P2 ، بعد دستور اول از P1 ، مجددا دستور دوم از P2 و در نهایت دستور دوم از P1 اجرا شود.

مثال

با فرض اینکه دو پردازنده P1 و P2 به صورت هم روند وجود دارند. نحوه ایجاد رشته $A(CD)^*B$ چگونه می باشد؟

P1	P2
<pre>while(TRUE){ cout<<"A"; cout<<"B"; }</pre>	<pre>while(TRUE){ cout<<"C"; cout<<"D"; }</pre>

پاسخ:

ابتدا اجرای P1 شروع شده و دستور `cout<<"A"` اجرا می شود. در این لحظه وقفه رخ داده و به P2 سوئیچ می شود. فرایند P2 چند مرتبه اجرا می شود. در نهایت با رخ دادن وقفه و سوئیچ به P1 ، دستور چاپ B اجرا می شود.

مثال

در صورتی که دو پروسس P1 و P2 به صورت هم روند اجرا شوند، نحوه چاپ BADC چگونه است؟ (مقدار اولیه متغیرهای x و y برابر صفر است).

P1	P2
<pre>while (x=0); cout<<"A"; cout<<"D"; y=1;</pre>	<pre>cout<<"B"; x=1; while (y=0); cout<<"C";</pre>

پاسخ:

نحوه چاپ رشته BADC به این صورت است که ابتدا فرایند P2 اجرا شده و کاراکتر B را چاپ کرده و بعد از یک شدن x ، به فرایند P1 سوئیچ شده و از حلقه عبور کرده و کاراکتر A و سپس D چاپ شده و در نهایت بعد از یک شدن y به P2 سوئیچ شده و از حلقه عبور کرده و کاراکتر C چاپ می شود.

رویکردهای انحصار متقابل

شرایطی که باید رعایت شود تا یک همکاری درست و کارا بین فرایندهای هم روند برقرار باشد، عبارتند از:

۱- انحصار متقابل (Mutual Exclusion)

از بین فرایندهایی که برای یک منبع یکسان دارای ناحیه بحرانی هستند، در هر لحظه فقط یک فرایند مجاز است که در ناحیه بحرانی خود باشد.

۲- پیشرفت (Progress)

فرایندی که فعلا تصمیم به ورود به ناحیه بحرانی را ندارد و در ناحیه غیر بحرانی می باشد، و دستورالعمل‌های عادی برنامه خود را اجرا می کند، نباید در تصمیم‌گیری برای ورود فرایندهای دیگر به ناحیه بحرانی شرکت کند. (امکان ممانعت نداشته باشد)

۳- انتظار محدود (Bounded Waiting)

باید مدت انتظار فرایندهایی که نیاز به ورود به ناحیه بحرانی دارند، محدود باشد. یعنی نباید دچار گرسنگی و بن بست شوند.
گرسنگی: به مدت نامعلوم و بدون حد بالای مشخص، منتظر فرایندهای دیگر بودن.
بن بست: تا ابد منتظر ورود به ناحیه بحرانی خود بودن.

البته علاوه بر رعایت ۳ شرط بالا، مسئله را باید در حالت کلی حل کرد و فرضی برای ساده سازی راه حل به کار نبرد. همچنین الگوریتم حالت قطعی و غیر تصادفی داشته باشد.

✍ اگر فرایند P2 بخواهد وارد ناحیه بحرانی شود در حالی که P1 در ناحیه بحرانی قرار دارد، P2 باید منتظر بماند تا P1 خارج شود. P2 برای انتظار کشیدن دو راه "انتظار مشغول و مسدود شدن" پیش رو دارد. روش انتظار مشغول دارای مشکل اتلاف پردازنده است و از این روش وقتی استفاده می کنیم که زمان انتظار کوتاه باشد.

برای تحقق انحصار متقابل، پیشنهادهای مختلفی وجود دارد. این راه حل‌ها را به صورت چهار رویکرد زیر، دسته بندی می‌کنیم:

۱- نرم افزاری

۲- با حمایت سخت افزار (با کمک دستورالعمل‌های خاص CPU)

۳- با حمایت سیستم عامل (با کمک فراخوان‌های سیستمی خاص)

۴- با حمایت زبان برنامه سازی (با کمک کامپایلر)

رویکردهای نرم افزاری انحصار متقابل

راه حل‌های نرم افزاری مستقیماً توسط برنامه‌ها استفاده می‌شوند و وجود حافظه اشتراکی ضروری می‌باشد. در این راه حل‌ها از دستورالعمل‌های خاص توسط سخت افزار استفاده نمی‌شود و حمایتی از سیستم عامل و زبان‌های برنامه سازی نداریم.

الگوریتم Decker

آقای Decker اولین شخصی بود که یک راه حل نرم افزاری دو فرایندی برای مسئله انحصار متقابل ارائه داد. Decker با پنج مرحله تلاش به راه حل درست رسید. این تلاش‌ها در زیر آورده شده است.

تلاش اول (تناوب قطعی)

در این روش از یک متغیر سراسری مشترک به نام turn استفاده شده که دو مقدار 0 یا 1 را می‌تواند بگیرد. مقدار اولیه turn برابر 0 است. بنابراین ابتدا P0 می‌تواند وارد ناحیه بحرانی شود. در این حالت P1 در حلقه while منتظر می‌ماند تا P0 از ناحیه بحرانی خارج شده و turn را 1 کند. در این صورت P1 می‌تواند وارد ناحیه بحرانی شود. برنامه فرایندها به صورت زیر است:

<pre> P0(void) { while(TRUE) { while(turn != 0) ; /*wait*/ critical-section(); turn = 1; non-critical- section(); } } </pre>	<pre> P1(void) { while(TRUE) { while(turn != 1) ; /*wait*/ critical-section(); turn = 0; non-critical- section(); } } </pre>
---	---

بررسی شرط‌ها در تلاش اول:

۱- انحصار متقابل

راه حل گفته شده انحصار متقابل را تضمین می‌کند و امکان ندارد که P0 و P1 با هم وارد ناحیه بحرانی شوند. چون P0 در صورتی وارد ناحیه بحرانی می‌شود که مقدار turn برابر 0 باشد و P1 با مقدار 1 وارد می‌شود. بنابراین امکان ندارد که هر دو به turn نگاه کنند و هر دو با هم مقدارهای 0 و 1 را ببینند.

۲- پیشرفت

این روش شرط پیشرفت را رعایت نمی‌کند. چون اگر P1 وارد ناحیه بحرانی شود و کارش تمام شده و به بخش غیر بحرانی برود، در این حالت turn برابر 0 می‌باشد. حال نوبت P0 است که وارد ناحیه بحرانی شود، ولی می‌خواهد به مدت طولانی در ناحیه غیر بحرانی بماند. P1 به سرعت کارش در ناحیه غیر بحرانی تمام شده و قصد ورود مجدد به ناحیه بحرانی را دارد. اما چون turn برابر 0 است در حلقه انتظار می‌ماند تا بالاخره P0 وارد ناحیه بحرانی شده و بعد از خروج، turn را 1 کرده تا P1 بتواند وارد ناحیه بحرانی شود. در این سناریو، P1 توسط فرایندی منتظر مانده بود که در ناحیه بحرانی نبود و جلوی پیشرفت اش را گرفته بود.

۳- انتظار محدود

در این روش قحطی نداریم، چون فرایندها به صورت یک در میان و نوبتی وارد ناحیه بحرانی می‌شوند. همچنین بن بست نیز نداریم. بنابراین شرط انتظار محدود رعایت می‌شود.

یکی از معایب این روش این است که سرعت عملیات توسط فرایند کندتر تعیین می‌شود، چون فرایندها برای دسترسی به ناحیه بحرانی باید به صورت یک در میان عمل کنند.

یکی از معایب این روش این است که اگر فرایندی در ناحیه بحرانی از کار بیفتد، فرایند دیگر تا ابد منتظر خواهد ماند.

تلاش دوم

در این روش از دو متغیر پرچم مشترک به نام های `flag[0]` و `flag[1]` با مقدار اولیه FALSE استفاده می‌شود که هر کدام متعلق به یک فرایند است. هر فرایندی که قصد ورود به ناحیه بحرانی خود را دارد، پرچم خود را TRUE می‌کند.

در این روش هر فرایند دارای کلید مجزا برای ورود به ناحیه بحرانی است تا اگر فرایندی قصد استفاده از ناحیه بحرانی را نداشت، فرایند دیگر بتواند به ناحیه بحرانی خود دسترسی داشته باشد. برنامه فرایندها به صورت زیر است:

```
boolean flag[2] = {FALSE,FALSE};
```

<pre>P0(void){ while(TRUE) { while(flag[1]); flag[0] = TRUE; critical-section(); flag[0] = FALSE; non-critical- section (); } }</pre>	<pre>P1(void){ while(TRUE) { while(flag[0]); flag[1] = TRUE; critical-section(); flag[1] = FALSE; non-critical- section (); } }</pre>
---	---

بررسی شرط‌ها در تلاش دوم

۱- انحصار متقابل

این روش انحصار متقابل رعایت نمی‌شود. یعنی حتی از روش اول هم بدتر است. سناریو: فرض کنید که P0، flag[1] را خوانده و آن را FALSE می‌بیند، اما قبل از TRUE کردن flag[0]، P1 اجرا شود و flag[0] را خوانده و آن را FALSE می‌بیند و برای ورود به ناحیه بحرانی flag[1] را TRUE کرده و وارد می‌شود. حال در این زمان به P0 سوئیچ شده و flag[0] را TRUE کرده و این فرایند هم وارد ناحیه بحرانی می‌شود! بنابراین چون هر دو فرایند در یک زمان وارد ناحیه بحرانی خود شده‌اند، شرط انحصار متقابل برقرار نمی‌باشد.

۲- پیشرفت

این روش شرط پیشرفت را رعایت می‌کند. اگر P0 در ناحیه غیر بحرانی خود باشد، flag[0] را FALSE نگه می‌دارد تا P1 بتواند وارد ناحیه بحرانی خودش شود. اگر P0 برای مدت طولانی تصمیم به ورود به ناحیه بحرانی را نداشته باشد، P1 به دفعات می‌تواند وارد ناحیه بحرانی شود و سپس خارج شود. یعنی P0 جلوی پیشرفت P1 را نمی‌گیرد.

۳- انتظار محدود

این روش شرط انتظار محدود را رعایت نمی‌کند، چون امکان قحطی دارد. در این تلاش امکان ورود پی در پی یک فرایند به ناحیه بحرانی و عدم دستیابی فرایند دیگر به ناحیه بحرانی وجود دارد. سناریو: فرض کنید P0 در ناحیه بحرانی است و به P1 سوئیچ می‌شود. چون flag[0] برابر TRUE است، P1 نمی‌تواند وارد ناحیه بحرانی شود. بعد از پایان کوانتوم، پردازنده به P0 داده شده و این فرایند سریعاً

ناحیه بحرانی اش را اجرا کرده و سعی به ورود مجدد به ناحیه بحرانی را دارد و این اجازه به او داده می شود. P0 مجدداً flag[0] را TRUE کرده و اگر به P1 سوئیچ شود، باز هم نمی تواند اجازه ورود بگیرد. بنابراین تلاش برای دستیابی به ناحیه بحرانی تصادفی است و امکان قحطی وجود دارد.

یکی از معایب این روش این است که اگر فرایندی در ناحیه بحرانی از کار بیفتد، فرایند دیگر تا ابد منتظر خواهد ماند.

تلاش سوم

در تلاش دوم هر فرایند ابتدا وضعیت فرایند مقابل را چک کرده و سپس flag خود را TRUE می کند. بنابراین اگر هر دو به طور همزمان قصد ورود به ناحیه بحرانی را داشته باشند، flag یکدیگر را FALSE می بینند و با هم وارد می شوند. برای حل این مشکل دو سطر مسئله را عوض می کنیم.

```
boolean flag[2] = {FALSE,FALSE};
```

<pre>P0(void){ while(TRUE) { flag[0] = TRUE ; while(flag[1]); critical-section(); flag[0]= FALSE; non-critical- section(); } }</pre>	<pre>P1(void){ while(TRUE) { flag[1] = TRUE; while(flag[0]); critical-section(); flag[1]= FALSE; non-critical- section(); } }</pre>
--	---

بررسی شرط ها:

۱- انحصار متقابل

اگر یکی از فرایندها بتواند وارد ناحیه بحرانی شود، چون قبل از بررسی flag مقابل، flag خود را TRUE کرده، جلوی ورود فرایند مقابل را می گیرد. بنابراین شرط انحصار متقابل برقرار است.

۲- پیشرفت

این روش شرط پیشرفت را رعایت می کند. (با همان استدلال تلاش دوم)

۳- انتظار محدود

در این روش شرط انتظار محدود رعایت نمی شود، چون امکان بن بست وجود دارد. سناریو: فرض کنید که P0، flag[0] را TRUE کند ولی قبل از بررسی flag[1] در برنامه P0، به P1 سوئیچ شود و P1، flag[1] را TRUE کند. در این صورت هر دو فرایند تا ابد در حلقه انتظار گرفتار می شوند و بن بست رخ می دهد.

تلاش چهارم (ادب و تعارف)

در تلاش قبلی هر فرایند می تواند روی حق خود برای ورود به بخش بحرانی اش پافشاری کند. در این روش هر فرایند متغیر flag خود را TRUE کرده تا خواست خود برای ورود به بخش بحرانی را نشان دهد، اما آماده است flag را تغییر دهد تا به فرایند دیگر احترام گذارد. یعنی فرایندی که قصد ورود به ناحیه بحرانی را دارد، اگر ببیند که فرایند مقابل هم می خواهد به ناحیه بحرانی وارد شود، فالg خود را برای مدت کوتاهی FALSE کرده تا فرایند مقابل بتواند وارد شود.

برنامه فرایندها به صورت زیر است:

```
boolean flag[2] = {FALSE,FALSE};
```

<pre>P0(void) { while(TRUE) { flag[0] = TRUE; while(flag[1]) { flag[0] = FALSE; delay_for_a_short_time(); flag[0] = TRUE; } critical-section(); flag[0] = FALSE; non-critical- section(); } }</pre>	<pre>P1(void) { while(TRUE) { flag[1] = TRUE; while(flag [0]) { flag[1] = FALSE; delay_for_a_short_time(); flag[1] = TRUE; } critical-section(); flag[1] = FALSE; non-critical- section(); } }</pre>
---	--

بررسی شرط ها:

۱- انحصار متقابل:

راه حل گفته شده انحصار متقابل را تضمین می کند. (با استدلال تلاش سوم)

۲- پیشرفت:

این روش شرط پیشرفت را رعایت می کند. (با همان استدلال تلاش دوم و سوم)

۳- انتظار محدود:

در این روش شرط انتظار محدود رعایت نمی شود، چون ممکن است یک فرایند به مدت نامعلوم و بدون حد بالای مشخص، گرفتار قسمت تاخیر (delay) شود و فرایند مقابل به دفعات وارد ناحیه بحرانی شود. بنابراین به دلیل امکان گرسنگی، شرط انتظار محدود رعایت نمی شود. همچنین به علت امکان وجود Livelock، نیز شرط انتظار محدود رعایت نمی شود.

مشکل Livelock

در تلاش چهارم مشکل بن بست وجود ندارد، اما مشکل جدیدی به نام Livelock وجود دارد. با دنبال کردن اجرای زیر، این مشکل را توضیح می‌دهیم:

(۱) P0 ، flag[0] را TRUE کند.

(۲) P1 ، flag[1] را TRUE کند.

(۳) P0 ، flag[1] را بررسی کند.

(۴) P1 ، flag[0] را بررسی کند.

(۵) P0 ، flag[0] را FALSE کند.

(۶) P1 ، flag[1] را FALSE کند.

هر دو فرایند در یک زمان به مدت کوتاه یکسان عقب نشینی می‌کنند. سپس با هم بر می‌گردند و مراحل بالا را تکرار می‌کنند. اگر این دنباله به طور نامحدود تکرار شود، ممکن است هیچ کدام از فرایندها نتوانند وارد ناحیه بحرانی شوند. البته این تکرار بن بست نمی‌باشد، چون با تغییر در سرعت نسبی فرایندها، این چرخه شکسته می‌شود.

تلاش پنجم (راه حل صحیح)

Decker بعد از چهار تلاش ناموفق، یک راه حل درست که شرایط انحصار متقابل، انتظار محدود و پیشرفت را با هم رعایت می‌کند، ارائه داد. در این روش متغیر نوبت را با متغیرهای پرچم ترکیب کرد. این الگوریتم در زیر آورده شده است.

```
boolean flag[2] = {FALSE,FALSE};
turn=0;
```

<pre> P0(void){ while(TRUE){ flag[0] = TRUE; while(flag[1]) if (turn == 1){ flag[0] = FALSE; while(turn==1) do; flag[0] = TRUE; } critical-section(); turn = 1; flag[0] = FALSE; non-critical-section (); } } </pre>	<pre> P1(void){ while(TRUE){ flag[1] = TRUE; while(flag[0]) if (turn == 0){ flag[1] = FALSE; while(turn==0) do; flag[1] = TRUE; } critical-section(); turn = 0; flag[1] = FALSE; non-critical-section (); } } </pre>
--	---

هنگامی که P0 بخواهد وارد بخش بحرانی خود شود، در flag مربوط به خود مقدار TRUE می‌گذارد. سپس flag مربوط به P1 را بررسی می‌کند که دو حالت رخ می‌دهد:

الف - flag[1] برابر TRUE باشد:

اگر turn برابر یک باشد، P0 به P1 احترام گذاشته و با FALSE کردن پرچم اش منتظر می‌ماند. در این هنگام P0 کاری انجام نمی‌دهد تا turn برابر صفر شود و سپس flag خودش را TRUE می‌کند.

ب - flag[1] برابر FALSE باشد:

P0 وارد بخش بحرانی شده و بعد از خروج از بخش بحرانی، در flag خود مقدار FALSE می‌گذارد تا بخش بحرانی را آزاد کند و در turn مقدار 1 را قرار می‌دهد تا حق پافشاری را به P1 واگذارد.

خلاصه ای از وضعیت تلاش های Decker

در جدول زیر، هر جا که شرط رعایت می‌شود با علامت ✓ مشخص شده است:

انتظار محدود	پیشرفت	انحصار متقابل	تلاش های Decker
✓	-	✓	تلاش اول
-	✓	-	تلاش دوم
-	✓	✓	تلاش سوم
-	✓	✓	تلاش چهارم
✓	✓	✓	تلاش پنجم

الگوریتم Peterson

چندین سال بعد، peterson راه حل ساده و زیبایی را برای حل مسئله انحصار متقابل ارائه کرد. این الگوریتم به سادگی برای n فرایند نیز قابل تعمیم است.

```
boolean flag[2] = {FALSE,FALSE};
turn=0;
```

<pre>P0(void){ { while (TRUE){ flag[0] = TRUE; turn = 0; while (turn==0 && flag[1]); critical-section(); flag[0] = FALSE; non-critical-section(); } }</pre>	<pre>P1(void){ { while (TRUE){ flag[1] = TRUE; turn = 1; while(turn==1 && flag[0]); critical-section(); flag[1] = FALSE; non-critical-section(); } }</pre>
--	---

در کتاب استالینگز، مقدار turn برعکس مقدار دهی و تست می شود که تفاوتی با روش بالا ندارد. به طور مثال برای P0، داریم:

```
turn = 1;
while(turn==1 && flag[1]);
```

نحوه کار کردن این الگوریتم:

فرض کنید P0 و P1 به طور تقریباً همزمان (P1 کمی دیرتر)، قصد ورود به ناحیه بحرانی را دارند. هر دو فرایند، شماره خود را در turn ذخیره کرده ولی P1 که دیرتر شماره اش را ذخیره کرده، شماره اش در turn می ماند و turn برابر 1 می شود. حال زمانی که P0 و P1 به دستور while می رسند، P0 از حلقه عبور کرده و وارد ناحیه بحرانی می شود ولی P1 در حلقه می چرخد (انتظار مشغول) و وارد ناحیه بحرانی نمی شود.

الگوریتم Peterson، شرایط انحصار متقابل، انتظار محدود و پیشرفت را با هم رعایت می کند. و مشکل بن بست و Livelock و قحطی ندارد. این الگوریتم ساده ترین و کوتاهترین راه حل نرم افزاری است.

معایبی که در هر یک از تلاش های Decker و همچنین روش Peterson وجود دارند، عبارتند از:

- ۱- اگر یکی از فرایندها در داخل ناحیه بحرانی از کار بیفتد، فرایند دیگر تا ابد منتظر می ماند.
- ۲- مبتنی بر انتظار مشغول می باشند.

رویکردهای انحصار متقابل با حمایت سخت افزار

می توان به کمک راه حل هایی که از دستورالعمل های ویژه ماشین استفاده می کنند، و نیاز به حمایت از طرف پردازنده دارند، مسئله انحصار متقابل را حل کرد. این راه حل ها عبارتند از:

۱- دستورالعمل از کار انداختن وقفه ها

۲- دستورالعمل TSL


۳- دستورالعمل SWAP

دستورالعمل از کار انداختن وقفه ها

در این راه حل، هر فرایند باید به محض ورود به ناحیه بحرانی، تمام وقفه ها را از کار بیندازد و دقیقا قبل از خروج از ناحیه بحرانی، همه وقفه ها را مجددا فعال سازد. در این صورت پردازنده نمی تواند از فرایندی به فرایند دیگر سوئیچ کند، چون وقفه ساعت غیر فعال است. بنابراین وقتی که فرایندی وقفه ها را غیر فعال می کند، می تواند بدون ترس از دخالت فرایندهای دیگر، به خواندن و نوشتن در حافظه مشترک بپردازد. رویکرد از کار انداختن وقفه ها برای انحصار متقابل به صورت زیر است:

```
P(int i){
  while(TRUE)
  {
    disable_interrupts();
    critical_section();
    enable_interrupts();
    non_critical_section();
  }
}
```

امکان دارد فرایندی وقفه ها را بعد از غیر فعال کردن، مجددا فعال نکند. (از معایب این روش) 

در سیستم های چندپردازنده ای، غیرفعال کردن وقفه ها، فقط در پردازنده ای که این دستورالعمل را اجرا کرده تاثیر گذار است و پردازنده های دیگر می توانند به ناحیه بحرانی دسترسی داشته باشند. 

دستورالعمل TSL

بسیاری از کامپیوترها دارای دستورالعمل (Test and Set Lock) TSL می باشند. این دستورالعمل محتویات یک کلمه از حافظه (lock) را خوانده و در ثبات قرار می دهد. سپس مقدار 1 را در همان آدرس از حافظه ذخیره می کند. این عملیات غیر قابل تقسیم انجام می شوند و تا اینکه اجرای دستورالعمل تمام نشود، هیچ فرایند و یا پردازنده دیگری نمی تواند به این کلمه از حافظه دسترسی پیدا کند.

در زیر تابع ورود به ناحیه بحرانی (enter_region) با استفاده از دستوالعمل TSL، به زبان اسمبلی آورده شده است:

```
enter_region:
    tsl  reg , lock
    cmp  reg , #0
    jne  enter_region
    ret
```

توسط اولین دستور، مقدار قبلی lock در رجیستر ذخیره شده و سپس در lock مقدار 1 ذخیره می شود. توسط دستور دوم مقدار قبلی lock با 0 مقایسه می شود. اگر 0 نبود، این عملیات تکرار شده (حلقه انتظار مشغول) و اگر 0 بود، زیربرنامه بازگشت کرده، در حالی که lock را 1 کرده است. تذکر: برای پاک کردن lock در هنگام خروج از ناحیه بحرانی، کافی است در آن 0 را ذخیره کرد:

```
move  lock , #0
ret
```

راه حل داده شده، انحصار متقابل و پیشرفت را رعایت می کند ولی انتظار محدود به دلیل گرسنگی را رعایت نمی کند.

دستور swap

می توان از دستورات عملی به نام swap، برای نوشتن یک روال ورود به ناحیه بحرانی استفاده کرد. این دستور می تواند در یک عمل واحد غیر قابل تقسیم، محتویات یک رجیستر پردازنده را با محتویات یک کلمه حافظه، جابجا کند.

```
enter_region:
    move  reg , #1
    swap  reg, lock
    cmp  reg, #0
    jne  enter_region
    ret
```

تذکر: برای پاک کردن lock در هنگام خروج از ناحیه بحرانی، کافی است در آن 0 را ذخیره کرد.

تذکر: xchg نام دیگر swap می باشد.

راه حل داده شده، انحصار متقابل و پیشرفت را رعایت می کند ولی انتظار محدود به دلیل گرسنگی را رعایت نمی کند.

راهکارهای سیستم عامل و زبان برنامه سازی برای تدارک همزمانی

راهکارهای سیستم عامل و زبان برنامه سازی برای تدارک همزمانی عبارتند از:

۱- سمافور (راهنما)

۲- ناظر (مانیتور)

۳- تبادل پیام.

سمافور

راه حل های نرم افزاری و سخت افزاری که بررسی کردیم ، دارای نقاط ضعف زیادی بودند. حال ابزاری به نام سمافور را معرفی می کنیم که دارای قدرت زیادی در برقراری انحصار متقابل می باشد و همچنین از عهده انواع مختلف مسایل همگام سازی بر می آید.

سمافور یک ساختار شامل فیلدهای زیر است:

۱- شمارنده صحیح (count)

از شمارنده برای شمارش تعداد wakeup هایی که می خواهند هدر بروند، استفاده می شود. با این کار این سیگنال ها برای استفاده های بعدی ذخیره می شوند.

۲- صف (queue)

از صف برای نگهداری فرایندهای بلوکه شده بر روی سمافور استفاده می شود.

توسط Dijkstra ، دو تابع wait و signal مطرح شدند که به ترتیب تعمیم یافته sleep و wakeup هستند.

توابع wait و signal به صورت زیر است:

<pre>void wait(semaphore s) { s.count = s.count -1; if (s.count < 0) { place this process in s.queue; block this process; } }</pre>	<pre>void signal(semaphore s) { s.count = s.count +1; if (s.count <= 0) { remove a process from s.queue; place this process in ready queue; } }</pre>
--	---

تابع wait ، یک واحد از شمارنده کم کرده و اگر منفی شود، فرایند مربوطه در صف، مسدود می شود.

تابع signal، یک واحد به شمارنده اضافه می‌کند، اگر مقدار شمارنده بیشتر از صفر نشود، یک فرایند از قبل مسدود شده در صف، آزاد می‌شود.

در بعضی از متون از حرف P به جای wait و از حرف V به جای signal استفاده می‌شود.

در بعضی از متون از down به جای wait و از up به جای signal استفاده می‌شود.

سمافور بر دو نوع عمومی و دودویی می‌باشد. در سمافور عمومی که آن را بررسی کردیم، شمارنده می‌تواند مثبت، صفر و یا منفی باشد. اما شمارنده در سمافور دودویی، فقط مقادیر 0 و 1 را دریافت می‌کند. توابع wait و signal برای سمافور باینری به صورت زیر است:

<pre>void wait(semaphore s) { if (s.count = 1) s.count = 0; else{ place this process in s.queue; block this process; } }</pre>	<pre>void signal(semaphore s) { if (s.queue is empty) s.count = 1; else{ remove a process from s.queue; place this process in ready queue; } }</pre>
--	---

قدرت سمافور دودویی معادل با سمافور عمومی است.

در سمافورهای صفی برای نگهداری فرایندهای بلوکه شده استفاده می‌شود. اگر خروج از این صف به ترتیب ورود باشد (FIFO)، به آن سمافور قوی می‌گویند و اگر این چنین نباشد، به آن سمافور ضعیف می‌گویند. در سمافور ضعیف، امکان گرسنگی وجود دارد. در این کتاب سمافورها از نوع قوی فرض می‌شوند.

انحصار متقابل با استفاده از سمافورها

می‌خواهیم انحصار متقابل را به کمک سمافورها برقرار کنیم. برای حل یک مسئله با سمافور، باید بدانیم که:

۱- به چند سمافور نیاز است.

۲- مقدار اولیه شمارنده هر سمافور چند باید باشد.

۳- عمل `wait` و `signal` در کجای برنامه باید روی سمافور انجام شود.

مسئله انحصار متقابل برای دو فرایند `p1` و `p2` توسط برنامه زیر پیاده‌سازی شده است. در این برنامه عمل `wait` قبل از ورود به ناحیه بحرانی و عمل `signal` بعد از خروج از ناحیه بحرانی روی سمافور دودویی `mutex` با مقدار اولیه 1 انجام شده است:

```
semaphore mutex =1;
void p(int i)
{
    while(TRUE)
    {
        wait(mutex);
        critical-section( );
        signal(mutex);
        non-critical-section( );
    }
}
```

فرض کنیم `P1` اول اجرا شود. با اجرای تابع `wait`، چون مقدار سمافور یک است، آن را صفر می‌کند و وارد ناحیه بحرانی می‌شود. در زمانی که `P1` در ناحیه بحرانی است، اگر `P2` سعی به ورود به ناحیه بحرانی داشته باشد، چون سمافور برابر 0 است، این فرایند بلوکه شده و در صف قرار می‌گیرد. بعد از خروج `P1` از ناحیه بحرانی، تابع `signal` را اجرا کرده و چون صف خالی نیست، `P2` که در صف قرار دارد را آزاد می‌کند و `P2` می‌تواند وارد ناحیه بحرانی شود.

می‌توان با استفاده از سمافور عمومی، انحصار متقابل را برای بیش از دو فرایند نیز پیاده‌سازی کرد.

سمافوری که ترتیب خروج در آن مشخص نباشد، سمافور ضعیف نامیده می‌شود. در این نوع سمافور، امکان گرسنگی وجود دارد. (در سمافور قوی، منطق صف FIFO است.)

پیاده‌سازی انحصار متقابل به کمک سمافور دارای مزایایی است از جمله: ارضای شرط‌های انحصار متقابل، پیشرفت و انتظار محدود. این روش عادلانه است، CPU را تلف نمی‌کند و مشکل اولویت معکوس ندارد.

همگام سازی با استفاده از سمافورها

سمافور دارای توانایی زیادی در حل مسائل همگام سازی دارد. دو فرایند را در نظر بگیرید که باید ابتدا دستور S1 در P1 و سپس دستور S2 در P2 اجرا شود. برای پیاده سازی این همگام سازی از سمافوری به نام S با مقدار اولیه صفر به صورت زیر استفاده می کنیم:

P1	P2
.	.
S1;	wait(S);
signal(S);	S2;
.	.

مشخص است که تا زمانی که S1 از P1 اجرا نشود، اجرای S2 ممکن نیست. تذکر: معمولا مقدار اولیه سمافور در همگام سازی برابر صفر و در انحصار متقابل برابر 1 است.

مثال

آیا ترتیب اجرای P1, P2, P3 ممکن است؟ (مقدار اولیه سمافورها S و Q برابر صفر است)

P1	P2	P3
.	wait(Q);	wait(S);
.	.	.
signal(S);	.	signal(Q);

پاسخ: خیر.

بعد از اجرای P1، فرایند P2 نمی تواند اجرا شود، چون در ابتدای P2 دستور wait(Q) قرار دارد که به علت یک بودن سمافور Q، باعث بلوکه شدن P2 می شود. ولی اگر بعد از P1، فرایند P3 اجرا شود، به علت وجود دستور signal(Q) در انتهای آن، می توان بعد از P3 فرایند P2 را اجرا کرد. پس یک ترتیب اجرای ممکن عبارت است از: P1, P3, P2.

مثال

با فرض اینکه مقدار اولیه دو سمافور S و Q برابر صفر باشد، نحوه چاپ ABCDE را مشخص کنید.

P0	P1
signal(Q)	wait(Q);
wait(S);	cout<<"A";
cout<<"C";	signal(S);
cout<<"D";	cout<<"B";
	cout<<"E";

پاسخ:

در زیر ترتیب اجرا با شماره مشخص شده است:

P0	P1
(1) signal(Q);	(2) wait(Q);
(6) wait(S);	(3)
(7) cout<<"C";	cout<<"A";
(8) cout<<"D";	(4)
	cout<<"B";
	(5) signal(S);
	(9) cout<<"E";

مثال

نحوه چاپ 1324 را مشخص کنید؟ (مقدار اولیه دو سمافور S و Q برابر صفر است)

P1	P2
cout<<"1";	wait(S);
signal(S);	cout<<"3";
wait(Q);	signal(Q);
cout<<"2";	wait(S);
signal(S);	cout<<"4";

پاسخ: به علت استفاده از دستور wait(S) در ابتدای P2، و صفر بودن S، اجرا نمی‌تواند با P2 شروع شود. بنابراین ابتدا P1 اجرا شده و عدد 1 چاپ می‌شود. سپس توسط signal(S) مقدار سمافور S برابر یک می‌شود. به علت رسیدن به wait(Q)، و صفر بودن Q، نمی‌توان ادامه داد. با تعویض متن به P2، چون S برابر یک شده از wait(S) رد شده و S صفر می‌شود. سپس 3 چاپ می‌شود. حال دستور signal(Q) مقدار Q را برابر یک کرده و با رسیدن به wait(S)، چون S، صفر است، نمی‌توان ادامه داد و به ادامه P1 رفته و چون Q برابر یک است از wait(Q) رد شده و Q برابر صفر شده و سپس عدد 2 چاپ می‌شود. در نهایت توسط signal(S) مقدار S برابر یک شده و به P2 پرش کرده و از wait(S) عبور کرده و مقدار 4 چاپ می‌شود. در زیر ترتیب اجرا با شماره مشخص شده است:

P1	P2
(1) cout<<"1";	(3) wait(S);
(2) signal(S);	(4) cout<<"3";
(6) wait(Q);	(5) signal(Q);
(7) cout<<"2";	(9) wait(S);
(8) signal(S);	(10) cout<<"4";

مثال

نحوه اجرای فرایندهای P1 و P2 چگونه باشد، تا مقدار نهایی برابر $a=10$, $b=6$ شود؟
(مقدار اولیه سمافور S برابر یک و مقدار اولیه متغیرهای a,b نیز برابر یک می باشند.)

P1	P2
$a=a+2;$	$a=3;$
$b=b+1;$	wait(s);
signal(s);	$b=a+b;$
$b=b+1;$	wait(s);
	$a=a+b;$

پاسخ:

ابتدا سه دستور اول P2 اجرا می شود. بعد از اجرای این سه دستور داریم: $a=3, b=4, s=0$
سپس سه دستور اول P1 اجرا شده و بعد از اجرا خواهیم داشت: $a=5, b=5, s=1$
مجدداً به P2 بر گشته و دو دستور بعدی آن اجرا شده و داریم: $a=10, b=5, s=0$
در نهایت به P1 رفته و بعد از اجرای دستور باقی مانده خواهیم داشت: $a=10, b=6$

مثال

با فرض اینکه مقدار اولیه سمافور S برابر 8 ، سمافور P برابر 3 و سمافور Q برابر 1 باشد، حداکثر چند فرایند پشت هر سمافور قرار می گیرد؟

.
wait(S);
wait(P);
wait(Q);
.
signal(Q);
signal(P);
signal(S);
.

پاسخ:

از n فرایند، حداکثر 8 فرایند می تواند از wait(S) عبور کند و بقیه $(n-8)$ فرایند در صف سمافور S می خوابند. از 8 فرایندی که از سمافور S عبور کرده، حداکثر 3 فرایند می تواند از سمافور P عبور کند و 5 فرایند در صف P می خوابند. در نهایت از 3 فرایندی که از سمافور P عبور کرده، حداکثر 1 فرایند از wait(Q) عبور کرده و 2 فرایند در صف Q می خوابند.



مسئله تولیدکننده و مصرف کننده

یک تولیدکننده یا بیشتر، نوعی داده را تولید و آنها را در بافری به اندازه n قرار می دهند. یک مصرف کننده، این اقلام را یکی یکی از بافر برمی دارد. سیستم باید از همپوشانی اعمال بافر جلوگیری کند، یعنی در هر زمان مصرف کننده یا تولید کننده می تواند به بافر دسترسی داشته باشد.

این راه حل از سه سمافور استفاده می کند:

۱- سمافور **mutex** :

سمافوری برای رعایت شرط انحصار متقابل است، تا تولید کننده و مصرف کننده به طور همزمان به بافر دسترسی نداشته باشند. (با مقدار اولیه 1)

۲- سمافور **full** :

سمافوری برای شمارش تعداد خانه های پر بافر (با مقدار اولیه 0)

۳- سمافور **empty** :

سمافوری برای شمارش تعداد خانه های خالی بافر (با مقدار اولیه n)

```
void producer(void){
    int item;
    while(TRUE) {
        item= produce( );
        wait(empty);
        wait(mutex);
        insert(item);
        signal(mutex);
        signal(full);    }
}
```

```
void consumer(void){
    int item;
    while(TRUE) {
        wait(full);
        wait(mutex);
        item=remove( );
        signal(mutex);
        signal(empty);
        consume( );    }
}
```

اگر در مسئله تولید کننده- مصرف کننده ، بافر نامحدود باشد، دیگر نیازی به سمافور empty نیست. در نتیجه دستورات `wait(empty)` از تولید کننده و دستورات `signal(empty)` از مصرف کننده حذف می شود.

مسئله غذا خوردن فیلسوف ها

پنج فیلسوف دور یک میز دایره ای نشسته اند. هر فیلسوف یک بشقاب ماکارونی دارد. بین هر جفت از بشقاب ها، یک چنگال قرار دارد.



هر فیلسوف برای خوردن از دو چنگال طرفین بشقاب استفاده می‌کند. زندگی هر فیلسوف از دو دوره متناوب خوردن و فکر کردن تشکیل شده است. زمانی که هر فیلسوف گرسنه می‌شود، سعی می‌کند دو چنگال سمت چپ و راست خود را بردارد. اگر موفق شد، برای مدتی غذا می‌خورد و سپس چنگال‌ها را زمین می‌گذارد و به فکر کردن ادامه می‌دهد. مسأله تغذیه فیلسوفان علاوه بر **انحصار متقابل** (که در یک زمان دو فیلسوف نمی‌توانند از یک چنگال استفاده کنند)، باید جوابگوی **بن بست** و **گرسنگی** نیز باشد. راه حل: هر فیلسوف ابتدا چنگال چپ و سپس چنگال راست را بر می‌دارد. بعد از تغذیه یک فیلسوف، دو چنگالی که استفاده می‌کرد را روی میز گذاشته و دیگران می‌توانند استفاده کنند.

```
semaphore room=4;
semaphore fork[5]={1};
void philosopher (int i){
    while(TRUE){
        think( );
        wait( room );
        wait( fork[i] );
        wait( fork[(i+1) % 5] );
        eat( );    ناحیه بحرانی
        signal ( fork[(i+1) % 5] );
        signal ( fork[i] );
        signal ( room );
    }
}
void main( ){
    parbegin (p(0),p(1),p(2),p(3),p(4));
}
```

سمافور room با مقدار اولیه 4، برای این است که اجازه ورود به بیش از چهار نفر داده نشود. اگر حداکثر چهار فیلسوف نشسته باشند، حداقل یک نفر به دو چنگال دسترسی خواهد داشت. اگر از room استفاده نمی‌شد، و اجازه ورود همزمان هر پنج نفر، داده می‌شد، همه آنها چنگالهای چپ خود را برداشته و دیگر چنگال اضافی نمی‌ماند که کسی بتواند چنگال راست خود را بردارد. بنابراین بن بست رخ می‌داد.

در کتاب تنبؤوم، راه حل زیر داده شده است. در این راه حل، از یک آرایه به نام state استفاده می‌کند که وضعیت جاری فیلسوف (فکر کردن (0)، گرسنگی (1) و خوردن (2)) را نگهداری می‌کند. یک فیلسوف در صورتی که هیچ یک از فیلسوفان چپ و راستش، در حال خوردن نباشند، می‌تواند غذا بخورد. این راه حل بن بست ندارد.

```
#define LEFT (i-1) % 5
#define RIGHT (i+1) % 5
typedef int semaphore;
semaphore mutex=1;
semaphore s[5];
int state[5];
void philosopher (int i){
    while(TRUE){
        think();
        take_forks(i);
        eat();
        put_fork(i);
    }
}
void take_forks(int i){
    wait(mutex);
    state[i]= 1;
    test(i);
    signal(mutex);
    wait(s[i]);
}
void put_forks(int i){
    wait(mutex);
    state[i]=0;
    test( LEFT );
    test( RIGHT );
    signal(mutex);
}
void test(int i){
    if( state[i]==1 && state[LEFT]!=2 && state[RIGHT]!= 2 ) {
        state[i]=2;
        signal(s[i]);
    }
}
```

مسئله خوانندگان و نویسندگان

در این مسئله، ناحیه داده ای (مثل فایل) وجود دارد که بین تعدادی از فرایندها مشترک است. فرایند های خواننده می خواهند از این ناحیه بخوانند و فرایندهای نویسنده می خواهند در آن بنویسند.

شرایط این مسئله:

- ۱- هر تعداد از خوانندگان می توانند به صورت همزمان از فایل بخوانند.
- ۲- در هر زمان تنها یک فرایند ممکن است در این فایل بنویسد.
- ۳- هنگامی که نویسنده ای در حال نوشتن است، هیچ خواننده ای نمی تواند فایل را بخواند.

برای مثال یک سیستم رزرواسیون هواپیمایی را در نظر بگیرید که تعداد زیادی فرایند در آن برای نوشتن و خواندن با یکدیگر رقابت می کنند. چند فرایند می توانند به طور همزمان پایگاه داده را بخوانند ولی اگر یک فرایند در حال به روز رسانی پایگاه داده باشد، فرایندهای دیگر حتی خوانندگان، نباید به پایگاه داده دسترسی داشته باشند.

این مسئله دارای سه حالت است:

- ۱) خوانندگان اولویت دارند. (تا زمانی که خواننده ای وجود دارد، به خواننده اجازه ورود می دهیم)
- ۲) نویسندگان اولویت دارند.
- ۳) خوانندگان و نویسندگان بدون اولویت هستند.

حالتی را بررسی می‌کنیم که خوانندگان اولویت دارند:

```
typedef int semaphore;
semaphore mutex=1;
semaphore w=1;
int rc=0;
```

```
void writer()
{
    while(TRUE)
    {
        wait(w);
        writing();
        signal(w);
    }
}
```

```
void reader()
{
    while(TRUE)
    {
        wait(mutex);
        rc = rc+1;
        if (rc == 1) wait(w);
        signal(mutex);

        reading();

        wait(mutex);
        rc = rc-1;
        if (rc == 0) signal(w);
        signal(mutex);
    }
}
```

روال نویسنده:

اولین نویسنده، با عمل $wait(w)$ ، اجازه دسترسی پیدا کرده (چون مقدار اولیه w برابر 1 است)، ولی با صفر شدن این سمافور، اجازه دسترسی نویسندگان و خوانندگان دیگر گرفته می‌شود. در این روال، تا هنگامی که نویسنده ای به فایل دسترسی دارد، هیچ نویسنده ای و خواننده دیگری نمی‌تواند به فایل دسترسی داشته باشد.

روال خواننده:

فرایند خواننده از سمافور W برای اعمال انحصار متقابل، از متغیر سراسری rc ، برای شمارش تعداد خوانندگان و از سمافور $mutex$ برای اطمینان از تغییر مناسب rc استفاده می‌کند.

دستور افزایش rc و کنترل آن در بین $wait(mutex)$ و $signal(mutex)$ قرار دارد، تا تغییر rc ، انحصاری شود. همچنین بعد از پایان عمل خواندن، دستور کاهش rc و کنترل آن در بین $wait(mutex)$ و $signal(mutex)$ قرار دارد تا تغییر rc ، انحصاری شود.

توسط if اول، به اولین خواننده اجازه ورود داده می‌شود، البته در صورتی که نویسنده ای فعال نباشد. توسط if پایانی، بررسی می‌کنیم که اگر خواننده فعال دیگری وجود نداشته باشد، در صورت اینکه نویسنده بلوکه شده ای داشته باشیم، به آن اجازه داده شود.

در این روش شرط انحصار متقابل و شرط پیشرفت رعایت می‌شود ولی شرط انتظار محدود رعایت نمی‌شود. (چون اولویت با خوانندگان است و امکان گرسنگی نویسندگان وجود دارد).

مانیتور

مشاهده کردید که همگام سازی فرایندها با سمافور پیچیده است. در واقع تنها مشکل سمافور در سیستمی که حافظه مشترک دارد، پیچیدگی استفاده از آن در حل مسائل همگام سازی می‌باشد. اگر برنامه نویس در استفاده از سمافور دقت لازم را نکند، ممکن است دچار بن بست شود.

مانیتور (ناظر) ساختاری از زبان برنامه سازی است که همان کار سمافور را انجام می‌دهد و کنترل آن هم ساده تر است. سمافور اغلب توسط سیستم عامل پشتیبانی می‌شود و می‌تواند توسط زبان برنامه سازی نیز پشتیبانی شود، اما مانیتور باید فقط توسط زبان برنامه سازی پشتیبانی شود.

ساختار مانیتور در زبانهای برنامه سازی به صورت برنامه کتابخانه ای پیاده سازی شده است. این به افراد اجازه می‌دهد تا قفل های مانیتور را روی هر شیئی بگذارند. مانیتور مولفه ای نرم افزاری، مشتمل بر یک یا چند رویه، دنباله ای از مقدارگذاری های اولیه و داده های محلی است.

ویژگی های اصلی مانیتور:

۱ - متغیرهای داده ای محلی مانیتور، تنها برای رویه های خود مانیتور قابل دسترس بوده و هیچ رویه دیگری به آنها دسترسی ندارد.


۲ - یک فرایند با احضار یکی از رویه های مانیتور، وارد آن می‌شود.


۳ - در هر زمان تنها یک فرایند می‌تواند در مانیتور در حال اجرا باشد، فرایندهای دیگری که مانیتور را احضار کرده اند تا فراهم شدن مانیتور، معلق خواهند بود. (برقراری انحصار متقابل)


متغیرهای شرطی (condition variable)

مانیتور با استفاده از متغیرهای شرطی که تنها از داخل مانیتور، قابل دسترس هستند، از همگام سازی حمایت می کند. برای این کار از دو تابع کتابخانه ای `cwait(condition)` و `csignal(condition)` که ساختاری از زبان برنامه سازی هستند، استفاده می کنند. در بعضی از منابع، از C اول این دستورها استفاده نمی شود.

اگر فرایندی `cwait(c)` را صدا بزند، پشت شرط C به خواب می رود. اگر فرایندی `csignal(c)` را صدا بزند، یک پیغام بیدار باش برای فرایندهایی که پشت شرط C خوابیده اند می فرستد که فقط یکی از آنها توسط زمانبند سیستم بیدار می شود.

متغیرهای شرطی، شمارنده نیستند و مانند سمافور نمی توانند سیگنال ها را برای استفاده آینده، ذخیره کنند. 

امتیازی که مانیتورها نسبت به سمافورها در همگام سازی فرایندها دارند، این است که تمام اعمال همگام سازی در محدوده مانیتور است. بنابراین کشف خطاها و تعیین اینکه همگام سازی صحیح انجام شده است، ساده تر می باشد. 

کامپایلر(نه برنامه نویس) انحصار متقابل را به صورت خودکار در مانیتور برقرار می سازد. 

حل مسئله تولیدکننده- مصرف کننده توسط مانیتور

برای حل مسائل به کمک مانیتور، نواحی بحرانی را در داخل مانیتور تعریف می کنیم. برای بخش هایی در مسئله که نیاز به همگام سازی وجود دارد، از متغیرهای شرطی کمک می گیریم.

```
monitor ProducerConsumer;
    condition full,empty;
    integer count;
```

```
procedure enter;
begin
    if count = N then cwait (full);
    enter_item;
    count := count + 1;
    if count=1 then csignal(empty);
end;
```

```
procedure remove;
begin
    if count = 0 then cwait (empty);
    remove_item;
    count := count - 1;
    if count=N-1 then csignal(full);
end;
```

```
procedure producer;
begin
    while true do
        begin
            produce_item;
            ProducerConsumer.enter;
        end
end;
```

```
procedure consumer;
begin
    while true do
        begin
            ProducerConsumer.remove;
            consume_item;
        end
end;
```

زبانهای برنامه سازی مانند C و پاسکال استاندارد، مانیتور را پشتیبانی نمی کنند.

در عمل همگام سازی مانیتورها هم ممکن است اشتباه رخ دهد. مثلاً اگر هر یک از اعمال csignal در ناظر bounded buffer حذف شوند، در این صورت فرایندهایی که وارد صف شرط مربوط به آن می شوند، تا ابد معطل خواهند بود.

در زبان جاوا، اگر نخی اجرای متدی که در معرفی آن از واژه کلیدی synchronized استفاده شده را شروع کند، هیچ نخ دیگری اجازه ندارد شروع به اجرای هیچ یک از متدهای synchronized درون آن کلاس کند. جاوا متغیرهای شرطی ندارد و به جای آنها دو رویه wait و notify دارد که وقتی این دو رویه در متدهای synchronized به کار روند، مشکل رقابتی پیش نمی آید.

بر اثر اجرای csignal ممکن است فرایند دیگری که از قبل wait شده بود، فعال گردد و در نتیجه بیش از یک فرایند در یک زمان در مانیتور فعال شود. برای پرهیز از فعالیت همزمان دو فرایند درون یک مانیتور، Hansen پیشنهاد کرد که فرایندی که csignal را انجام داده باید فوراً از مانیتور خارج شود. بنابراین یک دستور csignal می تواند فقط به عنوان آخرین دستور در رویه مانیتور به کار رود.

تبادل پیام (Message Passing)

وقتی فرایندها با یکدیگر محاوره می کنند، نیازهای همگام سازی و ارتباط باید تامین شود. فرایندها نیاز به همگام سازی دارند تا انحصار متقابل اعمال گردد. ممکن است فرایندهایی که با یکدیگر همکاری می کنند نیاز به تبادل اطلاعات داشته باشند. یک رویکرد در فراهم کردن این دو عمل، تبادل پیام است. عمل واقعی تبادل پیام به شکل دو اولیه زیر ارائه می شود. این اولیه ها، فراخوان سیستمی هستند که می توان آنها را در رویه های کتابخانه ای قرار داد:

1 - send (destination , message)

2- receive (source , message)

فراخوان send، پیامی را از آدرسی که در فضای آدرس فرایند مبدا قرار دارد را برداشته (message) و به فرایند مقصد (destination) ارسال می کند. فراخوان receive، یک پیام را از فرایند مبدا (source) دریافت کرده و آن را در آدرس مشخص شده (message) قرار می دهد.

دو نوع آدرس دهی وجود دارد:

۱- مستقیم: پیامها مستقیماً از فرستنده به گیرنده فرستاده می شود.

۲- غیر مستقیم: فرستنده پیام را به یک صف مشترک (صندوق پستی) ارسال می کند و گیرنده پیام را از صندوق پستی بر می دارد. (سیستم عامل برای هر فرایند، بافری به نام mailbox ایجاد می کند).

اگر بین فرستنده ها و گیرنده ها رابطه چند به یک وجود داشته باشد، به صندوق پستی، "درگاه" می گویند.

پیام در یک قالب متداول از دو بخش، سرآمد (حاوی اطلاعاتی درباره پیام) و بدنه (حاوی خود پیام) تشکیل شده است.

همگام سازی به کمک تبادل پیام

می توان به کمک send و receive امکان هماهنگی بین فرایندها را ایجاد کرد. تبادل پیام بین دو فرایند متضمن سطحی از همگام سازی بین آنها می باشد.

زمانی که فرایندی، send را اجرا می کند، دو امکان وجود دارد:

الف- فرایند فرستنده تا دریافت پیام مسدود می شود.

ب- فرایند فرستنده، بدون توجه به عمل انجام شده، اجرایش ادامه می یابد.

زمانی که فرایندی receive را اجرا می کند، دو امکان وجود دارد:

الف- اگر پیامی قبلاً فرستاده شده، این پیام دریافت شده و اجرا ادامه می یابد.

ب- اگر پیام منتظری وجود نداشته باشد، فرایند تا رسیدن پیام مسدود می ماند و یا به اجرا ادامه می دهد. بنابراین

فرستنده و گیرنده هر دو می توانند مسدود شوند یا مسدود نشوند باشند.

پیاده سازی انحصار متقابل توسط تبادل پیام

از تبادل پیام می توان برای اعمال انحصار متقابل استفاده کرد. فرض می کنیم که send مسدود نشونده و receive مسدود شونده است. یعنی وقتی فرایندی می خواهد اطلاعاتی را دریافت کند، با اجرای receive بلوکه می شود تا اطلاعات برایش فرستاده شود.

مجموعه ای از فرایندهای همزمان در صندوق پستی mutex شریکند و می توانند از این صندوق پستی برای ارسال و دریافت پیام استفاده کنند.

const n= تعداد فرایندها ;


```
void p(int i)
```

```
{
  while(true)
  {
    receive(mutex , msg);
    /*critical section */
    send(mutex , msg);
    /*non critical section */
  }
}
```

```
void main()
```

```
{
  create_mailbox(mutex);
  send(mutex , null);
  Par begin( P(1) , P(2), ...,P(n) );
}
```

صندوق پستی با پیامی با محتوای تهی مقدار گذاری اولیه شده است. فرایندی که می خواهد وارد بخش بحرانی خود شود، ابتدا سعی می کند پیامی دریافت نماید. اگر صندوق پستی خالی باشد، مسدود می گردد. هنگامی که فرایندی پیام را به دست می آورد، بخش بحرانی خود را انجام داده و سپس پیام را به صندوق پستی بر می گرداند. در نتیجه این پیام نقش نشانه ای را بازی می کند که از فرایندی به فرایند دیگر منتقل می شود.

 اگر بیش از یک فرایند عمل دریافت را همزمان انجام دهند، در این صورت:

- ۱- اگر پیامی باشد، تنها به یک فرایند داده شده و بقیه مسدود می شوند.
 - ۲- اگر صندوق پستی خالی باشد، تمام فرایندها مسدود می گردند. موقعی که پیامی فراهم می شود، تنها یکی از فرایندهای مسدود فعال شده و پیام به همان فرایند داده شود.
- این فرض ها در تمام امکانات تبادل پیام، برقرار است.

حل مسئله تولیدکننده- مصرف کننده توسط تبادل پیام

می توان توسط تبادل پیام، راه حلی برای مسئله تولید کننده و مصرف کننده با بافر محدود ارائه داد. این برنامه علاوه بر علائم، داده ها را نیز عبور می دهد. تولید کننده با تولید داده ها، آنها را به عنوان پیام به صندوق پستی mc می فرستد. مصرف کننده تا هنگامی که حداقل یک پیام در صندوق وجود دارد، می تواند مصرف کند. بنابراین صندوق پستی mc مانند یک بافر عمل می کند.

راه حل:

یک صندوق پستی برای تولید کننده(mp) و یک صندوق پستی برای مصرف کننده(mc) ایجاد می شود:

```
create_mailbox(mc);
```

```
create_mailbox(mp);
```

سپس صندوق پستی تولید کننده، به اندازه ظرفیت بافر، با پیام تهی پر می شود:

```
for(i=0 ; i<capacity; i++)
```

```
send(mp,NULL);
```

سپس دو تابع تولید کننده و مصرف کننده به صورت همروند صدا زده می شود:

<pre>void producer() { message m1; while(TRUE) { receive(mp,m1); m1=produce(); send(mc,m1); } }</pre>	<pre>void consumer() { message m2; while(TRUE) { receive(mc,m2); consume(m2); send(mp,null); } }</pre>
--	---

زمانی که فرایندهای فرستنده و گیرنده پیام بر روی یک ماشین اجرا می شوند، مسئله کارایی پیش می آید. به عبارتی کپی کردن پیام ها از یک فرایند به فرایند دیگر همیشه آهسته تر از انجام عملیات سمافور و یا ورود به یک مانیتور است.

گیرنده می تواند به محض دریافت پیام از فرستنده، یک پیام تصدیق (acknowledgment) به فرستنده بفرستد و او را از دریافت پیام با خبر سازد.

در یک سیستم توزیعی با حافظه اختصاصی برای رعایت انحصار متقابل، از راه حل تبادل پیام استفاده می شود.

کنکور ارشد

(مهندسی کامپیوتر - دولتی ۷۹)

۱- دو فرایند P1 و P2 زیر به صورت هم روند اجرا می شوند. در صورتی که مقدار اولیه متغیر سراسری a صفر باشد، بعد از اجرای کامل دو فرآیند، کدام یک از گزینه های ذیل نادرست می باشد؟
(امکان اجرای آنها به صورت Interleaved نیز وجود دارد. یعنی در هر لحظه از اجرای فرایند، امکان وقوع وقفه و سوئیچ به فرایند دیگر وجود دارد.)

p1	p2
a=1;	b=a; c=a;

(۱) هر یک از مقادیر a و b و c یک می باشد.

(۲) مقادیر b و c صفر می باشد و مقدار a یک است.

(۳) مقادیر a و c هر کدام یک می باشد و مقدار b صفر است.

(۴) مقادیر a و b هر کدام یک می باشد و مقدار c صفر است.

پاسخ: جواب گزینه ۴ است.

بعد از اجرای کامل دو فرایند، مقادیر این متغیرها نامشخص است و حالت های زیر ممکن می باشد:

(۱) ابتدا p1 به طور کامل اجرا شده و سپس p2 اجرا شود. در این حالت $a=1, b=1, c=1$.

(۲) ابتدا p2 به طور کامل اجرا شده و سپس p1 اجرا شود. در این حالت $a=1, b=0, c=0$.

(۳) دستور اول از p2 اجرا شده و به p1 سوئیچ شود. بعد از اجرای p1 به p2 سوئیچ شده و دستور بعدی آن اجرا شود. در این حالت داریم $a=1, b=0, c=1$.

گزینه ۴ نادرست است، چون وقتی مقدار b یک می شود که دستور $b=a$ بعد از دستور $a=1$ اجرا شود. بعد از اجرای این دو دستور، نوبت به اجرای دستور $c=a$ می باشد که مقدار c یک می شود.

(مهندسی IT - دولتی ۸۴)

۲- در یک سیستم هم روند، هر پروسس برای ورود به بخش بحرانی، تابع `enter-region` و پس از خروج از بخش بحرانی تابع `leave-region` آمده در زیر را صدا می‌کند. کدام گزینه صحیح است؟

```
#define FALSE 0
#define TRUE 1
#define N 2
int turn;
int interested[N]; /* all values initially 0(FALSE) */
void enter-region(int process){
    int other;
    other = 1 - process;
    interested[process] = TRUE;
    turn = process;
    while (turn == process && interested[other] );
}
void leave-region(int process){
    interested[process] = FALSE;
}
```

(۱) امکان گرسنگی وجود دارد (starvation) (۲) امکان بن بست وجود دارد. (deadlock)

(۳) انحصار متقابل تأمین می‌شود. (۴) انحصار متقابل تأمین نمی‌شود.

پاسخ: جواب گزینه ۳ است.

این تست همان راه حل پیترسون در کتاب Tanenbaum، است. در نتیجه هر سه شرط "انحصار متقابل، انتظار محدود و پیشرفت" را تأمین می‌کند. و مشکل بن بست و گرسنگی ندارد.

در این راه حل، هر فرایند قبل از ورود به ناحیه بحرانی، تابع `enter_region` و پس از خروج، تابع `leave_region` را فراخوانی می‌کند.

به طور مثال برای `process=0` داریم:

```
P0(void)
{
    while(TRUE)
    {
        enter_region(0);
        critical-section( );
        leave_region(0);
    }
}
```



(مهندسی IT - دولتی ۸۳)

۳- الگوریتم زیر یک راه حل نرم افزاری برای حل مسئله ناحیه بحرانی است. در این الگوریتم:

(IT - دولتی ۸۳)

```
var c1, c2 : boolean;
turn: integer;
c1 :=TRUE;
c2 :=TRUE;
```

cobegin

P1:

loop

```
c1:=FALSE;
turn:=1
while (not c2) and (turn=1) do;
CS1;
c1:=TRUE;
```

end loop

P2:

loop

```
c2:=FALSE;
turn=2;
while (not c1) and (turn=2) do;
CS2;
c2:=TRUE;
```

end loop

coend

۱) این الگوریتم صحیح می باشد.

۲) Dead lock وجود دارد.

۳) Starvation وجود دارد.

۴) شرایط ناحیه بحرانی برآورده نمی شود (استفاده انحصاری)

پاسخ: گزینه ۱ جواب است.

این الگوریتم، مشابه راه حل Peterson است. در پیترسون مقدار اولیه flag ها false بود و هر فرایند قبل از ورود به ناحیه بحرانی، آن را true و بعد از خروج آن را false می کرد. در این تست، این منطق معکوس شده است که تاثیری بر عملکرد صحیح آن نمی گذارد و مانند پیترسون درست کار می کند و همه شرایط را رعایت می کند.



(مهندسی کامپیوتر - دولتی ۷۴)

۴- آیا کد صوری زیر برای مسأله Critical Section بین دو فرآیند هم روند قابل قبول است؟ چرا؟
(اپراتور AND به معنای اجرای هم روند است.)

```
{
  int turn;
  boolean falg[2];
  proc (int i ) {
    while(TRUE)
    {
      compute;
      falg[i]:= TRUE;
      turn:=( i+1) mod 2;
      while( falg[(i+1) mod 2] and turn = i );
      critical-section;
      flag[i] := FALSE;
    }
  }
  turn:=0;
  falg [0] :=FALSE;
  falg [1] :=FALSE;
  proc(0) AND proc(1);
}
```

(۱) خیر- زیرا Deadlock وجود دارد.

(۲) بلی- زیرا شرط Mutual Exclusion برقرار است.

(۳) خیر- زیرا شرط Mutual Exclusion برقرار نیست.

(۴) بلی- زیرا Deadlock وجود ندارد.

پاسخ: جواب گزینه ۳ است.

روش به کار رفته، روش پیترسون است با این تفاوت که قبل از حلقه در turn ، مقدار معکوسی قرار داده می شود. (یعنی برای p_0 مقدار 1 و برای p_1 مقدار 0). که این کار باعث نادرست شدن روش می شود.

فرض کنید P_0 اجرا شود و flag خود را true کرده و توسط دستور $turn:=(i+1)mod 2$ مقدار turn را یک می کند و چون turn برابر صفر نیست از while عبور کرده و وارد ناحیه بحرانی می شود. اگر در این زمان وقفه ای رخ دهد و به P_1 سوئیچ شود، این فرایند نیز بعد از true کردن flag خودش ، با اجرای دستور $turn:=(i+1)mod 2$ مقدار turn را صفر کرده و چون turn برابر یک نیست ، این فرایند نیز وارد ناحیه بحرانی می شود. بنابراین شرط انحصار متقابل برقرار نیست.



(مهندسی IT - دولتی ۸۸)

۵- راه حل ناحیه بحرانی زیر را برای فرآیندهای $P_i : (i = 1, 2)$ در نظر بگیرید. کدام مورد صحیح است؟

shared var

turn: integer; turn:=0;

Pi :

while (1) {

flag[i]:=TRUE;

turn:= (turn+ i) %2 + 1;

while(not(flag[i]) or turn == i %2 +1);

Critical – Section

flag[i]:= FALSE;

turn:=(turn+ i) %2 + 1;

Non Critical – Section

}

(۱) راه حل ناحیه بحرانی کاملاً صحیح است.

(۲) شرط پیشرفت (Progress) تنها شرطی است که نقض می‌گردد.

(۳) شرط انحصار متقابل (mutual exclusion) تنها شرطی است که نقض می‌گردد.

(۴) هر دو شرط انحصار متقابل و پیشرفت نقض می‌شوند.

پاسخ: گزینه ۴ جواب است.

مقدار flag ها در این راه حل بی‌اثرند. چون در بدنه هر فرایند، به جای بررسی flag فرایند مقابل، flag خود فرایند چک می‌شود. بعد از حذف flag ها، کد هر دو فرآیند به صورت زیر می‌باشد:

P1	P2
turn:=0	turn:=0;
while(1){	while(1){
turn:= (turn+1) %2 + 1;	turn:= (turn+2) %2 + 1;
while(turn == 2);	while(turn == 1);
Critical – Section	Critical – Section
turn:=(turn+1) %2 + 1;	turn:=(turn+2) %2 + 1;
Non Critical – Section}	Non Critical – Section}

اگر p1 اجرا شود، مقدار turn را به 2 تغییر می‌دهد و در حلقه while منتظر می‌ماند. اگر در این لحظه به p2، سوئیچ شود، p2، نیز مقدار turn را به 1 تغییر می‌دهد و در حلقه while منتظر می‌ماند. بنابراین بن بست رخ داده و شرط انتظار محدود نقض می‌شود. شرط پیشرفت نیز نقض می‌شود، چون زمانی که p1 در حلقه while منتظر است، اگر p2 که در خارج از ناحیه بحرانی است، قصد ورود به ناحیه بحرانی را نداشته باشد، مانع پیشرفت p1 می‌شود. با این شرایط بدیهی است که شرط انحصار متقابل نیز نقض می‌شود. بنابراین هر ۳ شرط "انحصار متقابل، پیشرفت و انتظار محدود" نقض می‌شود. تذکر: بهتر بود طراح محترم، در گزینه ۴، شرط انتظار محدود را نیز اضافه می‌کرد، تا داوطلب فکر نکند که منظور این است که فقط آن دو شرط که در گزینه ۴ آمده، نقض می‌شود.



(مهندسی IT – دولتی ۹۱)

۶- آیا کد زیر می تواند راه حلی برای دو پردازش هم روند باشد؟

```

proc ( int i ){
  while(TRUE){
    computation;
    key[i] = TRUE;
    while ( key[i] )
      swap (key[i] , lock );
    CS ناحیه بحرانی
    lock = FALSE;
  }
}
lock = FALSE;
key[1] = FALSE;
key[2] = FALSE;

```

۱) راه حل صحیح نیست، زیرا انحصار متقابل (mutual exclusion) رعایت نمی شود.

۲) راه حل صحیح نیست، زیرا شرط پیشرفت برقرار نیست.

۳) راه حل صحیح نیست، زیرا تضمینی برای محدودیت زمان انتظار ندارد.

۴) راه حل صحیح است.

پاسخ: جواب گزینه ۳ است.

راه حل داده شده، انحصار متقابل و پیشرفت را رعایت می کند. ولی چون نوبت دهی برای ورود به ناحیه بحرانی کاملا تصادفی است، شرط انتظار محدود رعایت نمی شود.

(مهندسی IT – دولتی ۹۰)

۷- پیاده سازی زیر از عملیات تجزیه ناپذیر (atomic) روی سمافورها را در نظر بگیرید:

```

typedef struct{
  int value;
  struct process *list;
}semaphore;

```

<pre> wait(semaphore *S) { S -> value--; if (S -> value < 0) { add this process to S -> List; block(-(S -> value)); } } </pre>	<pre> signal(semaphore *S) { S -> value++; if (S -> value <= 0) { remove a process P from S -> List; wakeup(P); } } </pre>
---	--

مفروضات و تعاریف زیر را نیز داریم:

$block(n)$: فرآیندهای بلوک شونده را به ترتیب n از کوچک به بزرگ مرتب کرده که به این ترتیب فرآیند اول با $block(1)$ در سر صف قرار می‌گیرد و به ترتیب n (از کوچک به بزرگ) توسط $wakeup(P)$ از حالت بلوک خارج می‌شوند.

$S \rightarrow value = 1$

برنامه‌های سیستم به شکل زیر مفروضند:

```
mutex: semaphore;
do{
    wait(mutex);
    critical-section();
    signal(mutex);
    remainder-section
}while(TRUE)
```

کدام گزینه صحیح است؟

- (۱) خواص انحصار متقابل و انتظار محدود برقرارند ولی پیشرفت برقرار نیست.
- (۲) خواص انحصار متقابل و پیشرفت برقرارند ولی انتظار محدود برقرار نیست.
- (۳) خواص پیشرفت و انتظار محدود برقرارند ولی انحصار متقابل برقرار نیست.
- (۴) خواص انحصار متقابل برقرار است ولی پیشرفت و انتظار محدود برقرار نیست.

پاسخ: جواب گزینه ۲ است.

چون ترتیب خروج فرایندها بلوکه شده در صف، بر اساس ورود آنها نیست (سمافور ضعیف)، بنابراین امکان گرسنگی (قحطی) وجود دارد. بنابراین شرط انتظار محدود برقرار نیست.

(مهندسی کامپیوتر – آزاد ۸۶)

۸- در صورتی که دو کد زیر برای حل مسأله کلاسیک همزمانی بافر محدود با ظرفیت n باشد، آن گاه مقدار سمافورهای x و y و z باید چه مقداری باشند؟

مصرف کننده	تولید کننده
<pre>while (1) { wait (z) wait (y) حذف قطعه از بافر signal(y) signal (x) استفاده از قطعه }</pre>	<pre>while (1) { تولید قطعه wait (x) wait (y) اضافه کردن قطعه به بافر signal (y) signal (z) }</pre>

(۲) $x=n, y=n, z=1$

(۱) $x=0, y=1, z=n$

(۴) $x=0, y=n, z=1$

(۳) $x=n, y=1, z=0$

پاسخ: جواب گزینه ۳ است. از x برای شمارش تعداد خانه های خالی با مقدار اولیه n ، از z برای شمارش تعداد خانه های پر با مقدار اولیه 0 و از y برای انحصار متقابل با مقدار اولیه 1 استفاده می شود.

(مهندسی کامپیوتر – دولتی ۸۴)

۹- اگر مقادیر اولیه سمافورهای n, s به ترتیب $0, 1$ باشد، چنانچه دو زیر روال به طور هم روند اجرا شوند، کدام یک از گزینه های زیر صحیح است؟

procedure producer	procedure consumer
begin	begin
repeat	repeat
produce;	wait(s);
wait(s);	wait(n);
append;	take;
signal(n);	signal(s);
signal(s);	signal(n);
forevere	forever
end;	end;

(۱) راه حل کاملا درست است.

(۲) امکان بن بست وجود دارد.

(۳) امکان عدم تأمین انحصار متقابل وجود دارد.

(۴) امکان دارد که consumer در حالت گرسنگی بماند و producer فعال باشد.

پاسخ: همان مسئله تولید کننده- مصرف کننده با بافر نامحدود است، ولی جای دو wait در مصرف کننده عوض شده است. اگر ابتدا مصرف کننده اجرا شود، wait(s) ، s را صفر و wait(n) ، n را 1 کرده و در نتیجه مصرف کننده مسدود می شود. حال اگر تولید کننده اجرا شود، wait(s) ، s را 1 می کند و در نتیجه تولید کننده نیز مسدود می شود. در این لحظه چون هر دو فرایند مسدود هستند، بنابراین بن بست رخ می دهد.

(مهندسی IT – دولتی ۸۵)

۱۰- در راه حل زیر برای شام خواران فیلسوف (۵ فیلسوف)، فرض کنید اجرای روال های برداشتن دو چنگال و گذاشتن هر چنگال از ابتدا تا انتهای روال با رعایت کامل Mutual Exclusion انجام می شود.

روال take-forks(i) دو چنگال سمت راست و چپ را بررسی می کند و اگر هر دو موجود بودند برمی دارد و گرنه عمل بررسی را تکرار می کند.

روال put-fork(i) چنگال شماره i را می‌گذارد و از روال خارج می‌شود.

```
void philosopher (int i){
    while(1) {
        think;
        take-forks(i);
        eat;
        put-fork(i);
        put-fork((i+1)%n);
    }
}
```

کدام گزینه درست است؟

(۱) دارای بن بست

(۲) فاقد بن بست و گرسنگی

(۳) دارای بن بست و گرسنگی

(۴) فاقد بن بست ولی دارای گرسنگی

پاسخ: گزینه ۴ دست است.

در این راه حل چون توسط روال take-forks(i) یا هر دو چنگال همزمان برداشته شده و یا هیچ کدام برداشته نمی‌شود، بنابراین بن بست رخ نمی‌دهد.

در این روش امکان گرسنگی وجود دارد، چون همه چیز تصادفی است و نوبت و عدالت رعایت نمی‌شود و ممکن است یک فیلسوف به مدت نامعلوم گرسنه بماند.

(مهندسی کامپیوتر – آزاد ۸۷)

۱۱- کدام گزینه در مورد مانیتور درست نیست؟

(۱) فقط یک روال مانیتور در آن می‌تواند فعال باشد.

(۲) یک مانیتور هیچ‌گاه در بخش بحرانی خود مسدود نمی‌شود.

(۳) پیاده‌سازی مانیتور در سطح کامپایلر انجام می‌شود.

(۴) مانیتور به صورت ضمنی عملیات درخواست ورود/خروج از بخش بحرانی را انجام می‌دهد.

پاسخ: جواب گزینه ۲ است.

در مانیتور امکان مسدود شدن با wait بر روی متغیر شرطی، وجود دارد.

فصل ۵:

بن بست

بن بست (Deadlock) یعنی مسدود بودن دائمی مجموعه‌ای از فرایندها که برای منابع سیستم رقابت می‌کنند یا با یکدیگر در ارتباط هستند. این مسدود بودن ادامه می‌یابد تا سیستم عامل عمل فوق‌العاده‌ای انجام دهد، مثلاً فرایندی را حذف کند یا مجبور به برگشت به عقب کند.

یک مثال معمول از بن بست، ترافیک است. چهار خودرو که در یک زمان به چهار راهی رسیده‌اند مفروض است. چهار ربع این چهار راه منابع محسوب می‌شوند. اگر هر چهار خودرو وارد تقاطع بشوند، هر خودرو یک ربع از چهار راه (یک منبع) را در اختیار گرفته ولی نمی‌تواند پیش برود. چون منبع مورد نیاز دوم در اختیار خودروی دیگر است و این امر باعث ایجاد بن بست خواهد شد. (هر خودرو برای عبور از چهار راه به دو ربع از چهارراه نیاز دارد).

شرایط بن بست

اگر چهار شرط زیر همزمان در سیستمی وجود داشته باشد، بن بست رخ می‌دهد:

۱- نگهداری و انتظار (Hold and Wait)

فرایندی وجود دارد که حداقل یک منبع را در اختیار داشته باشد (نگهداری) و منتظر به دست آوردن منبع دیگری باشد که فعلاً در اختیار فرایند دیگری است.

۲- انحصاری بودن (قبضه نشدنی) (Non Preemption)

هنگامی که فرایندی منبعی را در اختیار دارد و نتوان آن منبع را به زور باز پس گرفت.

۳- انحصار متقابل (Mutual Exclusion)

در هر زمان تنها یک فرایند می‌تواند از یک منبع استفاده کند و اگر فرایند دیگری درخواست همان منبع را کند، باید منتظر بماند تا منبع آزاد شود.

۴- انتظار چرخشی (Circular Wait)

مجموعه‌ای از فرایندهای منتظر $\{P_0, P_1, \dots, P_n\}$ وجود دارد که P_0 منتظر منبعی باشد که در اختیار P_1 است و P_1 منتظر منبعی باشد که در اختیار P_2 است و به همین ترتیب، P_{n-1} منتظر منبعی باشد که در اختیار P_n است و در نهایت P_n منتظر منبعی است که در اختیار P_0 است. در واقع چرخه‌ای از انتظار وجود دارد.

برقرار بودن هر چهار شرط، شروط لازم و کافی برای وقوع بن بست می باشند.

شرط انتظار چرخشی منجر به شرط نگهداری و انتظار می گردد.

مثال

سیستمی با دو منبع R1 و R2 مفروض است. اگر فرایند P1 منبع R1 و فرایند P2 منبع R2 را در اختیار داشته باشد و فرایند P1 برای تکمیل اجرا به منبع R2 و فرایند P2 به منبع R1 نیاز داشته باشد، بن بست رخ می دهد.

اگر در سیستمی n فرایند و m منبع (از یک نوع) موجود باشد، در صورت برقرار بودن شرط

$$\sum_{i=1}^n Request(i) < m + n$$

بن بست رخ نخواهد داد.

مثال

کامپیوتری دارای m مورد از یک منبع می باشد و n فرایند برای در اختیار گرفتن آنها با هم رقابت می کنند. هر فرایند حداکثر به دو مورد نیاز دارد. حداکثر مقدار n که به ازای آن می توان مطمئن بود، سیستم دچار بن بست نشود، چقدر است؟

حل: چون در این مثال هر فرایند حداکثر به دو منبع نیاز دارد، مجموع کل درخواستها برای n فرایند برابر 2n می باشد و داریم:

$$\sum_{i=1}^n Request(i) < m + n \Rightarrow 2n < m + n \Rightarrow n < m$$

بنابراین حداکثر مقدار n که به ازای آن بن بست رخ نمی دهد، برابر m - 1 می باشد.

مثال

یک سیستم کامپیوتری دارای 6 عدد TapeDrive است که n پردازنده برای دستیابی به آنها رقابت می کنند. هر پردازنده به دو درایو نیاز دارد. این سیستم به ازای حداکثر چه ارزشهایی از n فاقد بن بست است؟

$$\sum_{i=1}^n Request(i) < m + n \Rightarrow 2n < 6 + n \Rightarrow n < 6$$

بنابراین حداکثر پنج فرایند می تواند وجود داشته باشد. در این حالت اگر هر کدام از فرایندها یک منبع را در اختیار گیرند، یک منبع آزاد باقی می ماند که یکی از فرایندها با گرفتن آن کامل می شود و به همین ترتیب بقیه فرایندها کامل خواهند شد و بن بست رخ نمی دهد. ■

گراف تخصیص منابع

توسط گراف تخصیص منابع می‌توان مسئله بن بست را دقیقتر تشریح کرد. در این گراف، فرایندها با دایره و منابع با مربع نشان داده می‌شوند.

در گراف تخصیص منابع دو نوع یال وجود دارد:

۱- یال درخواست $P_i \rightarrow R_j$

فرایند P_i نمونه‌ای از منبع R_j را درخواست کرده است و منتظر آن منبع است.

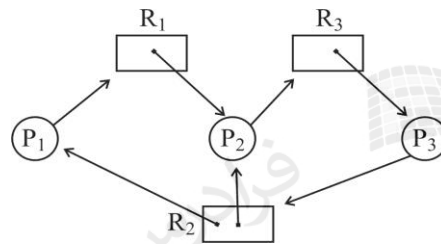
۲- یال تخصیص $R_j \rightarrow P_i$

نمونه‌ای از منبع R_j به فرایند P_i تخصیص داده شده است.

اگر گراف تخصیص منابع فاقد چرخه باشد، حالت بن بست وجود ندارد.

مثال

در گراف تخصیص منابع زیر، وضعیت بن بست را بررسی کنید؟



حل:

حالتهای فرایند در این گراف عبارتند از:

۱- فرایند P_1 نمونه‌ای از منبع R_2 را در اختیار دارد و منتظر نمونه‌ای از منبع R_1 است.

۲- فرایند P_2 نمونه‌ای از منبع R_1 و منبع R_2 را در اختیار دارد و منتظر نمونه‌ای از منبع R_3 است.

۳- فرایند P_3 نمونه‌ای از منبع R_3 را در اختیار دارد و منتظر نمونه‌ای از منبع R_2 است.

با توجه به این گراف، دو چرخه کمینه وجود دارد و به همین علت همه فرایندها در بن بست قرار دارند. این چرخه‌ها عبارتند از:

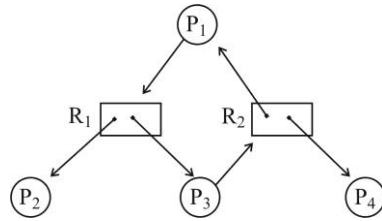
1) $P_1 \rightarrow R_1 \rightarrow P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_1$

2) $P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_2$



مثال

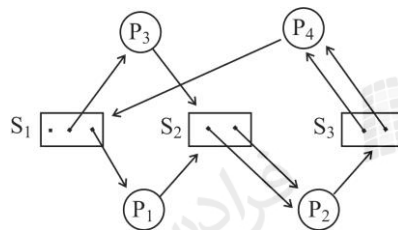
در گراف زیر وضعیت بن بست را بررسی نمایید.



حل: با وجود چرخه $P_1 \rightarrow R_1 \rightarrow P_3 \rightarrow R_2 \rightarrow P_1$ ، بن بست رخ نمی دهد. چون فرایند P_4 می تواند منبع نمونه R_2 را آزاد کند و آنگاه منبع R_2 به فرایند P_3 داده می شود و چرخه از بین برود. ■ اگر چرخه ای در گراف وجود داشته باشد، ممکن است بن بست وجود داشته باشد.

مثال

سیستمی متشکل از چهار پردازنده P_1 و P_2 و P_3 و P_4 و سه نوع منبع قابل استفاده مجدد S_1 و S_2 و S_3 را در



نظر بگیرید. در وضعیت موجود، کدام یک از پردازنده ها در بن بست قرار ندارند؟

حل: تعداد واحدهای هر منبع به ترتیب 2,2,3 است. P_1 یک واحد از منبع S_1 را در اختیار دارد و تقاضای یک واحد از S_2 را کرده است. P_2 دو واحد از S_2 را در اختیار دارد و تقاضای یک واحد از S_3 را کرده است. P_3 یک واحد از S_1 را در اختیار دارد و تقاضای یک واحد از S_2 را کرده است. P_4 نیز دو واحد از S_3 را در اختیار دارد و تقاضای یک واحد از S_1 کرده است. فرایند P_4 یک واحد از S_1 که موجود است را در اختیار گرفته و کامل می شود. بعد از کامل شدن P_4 ، منابعی را که در اختیار داشت (دو تا S_3 و یک S_1) را آزاد می کند. سپس فرایند P_2 کامل می شود، چون منابع درخواستی موجود است. بعد از P_2 فرایند P_1 و P_3 کامل می شوند. یعنی هیچ فرایندی در حالت بن بست نمی باشد. ■

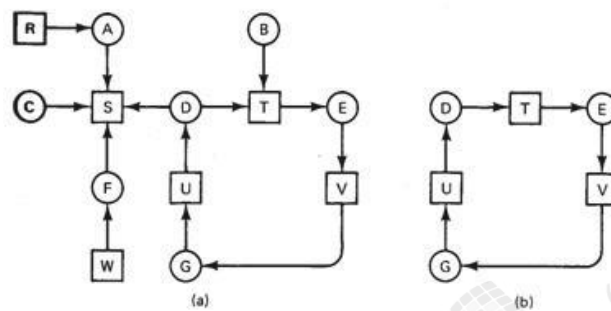
اگر هر نوع منبع دارای چند نمونه باشد، وجود چرخه الزاماً به معنای بن بست نمی باشد. یعنی در این حالت وجود چرخه شرط لازم برای وجود بن بست است ولی شرط کافی نیست.

مثال

سیستمی با 7 فرایند و 6 منبع مفروض است. با توجه به ۷ درخواست زیر، وضعیت را مشخص کنید؟

1. Process A holds R and wants S.
2. Process B holds nothing and wants T.
3. Process C holds nothing and wants S.
4. Process D holds U and wants S and T.
5. Process E holds T and wants V.
6. Process F holds W and wants S.
7. Process G holds V and wants U.

حل: گراف منابع زیر است:



مشخص می شود که حلقه DTEVGUD وجود دارد. بنابراین فرایندهای B, D, E, G در بن بست قرار دارند. اما فرایندهای A, F, C در بن بست نمی باشند، چون منبع S می تواند به هر یک از آنها داده شود و بعد از پایان کار با این منبع، به فرایند دیگر داده شود. ■

روش های رفع بن بست

روشهای رفع بن بست را می توان به سه گروه عمده تقسیم کرد:

۱- پیشگیری (جلوگیری) از بن بست (Prevention)

سیستم طوری طراحی شود که از قبل امکان بن بست از بین برود، یعنی تضمین کنیم که حداقل یکی از چهار شرط بن بست رخ ندهد.

۲- اجتناب از بن بست (Avoidance)

در مورد درخواست‌های منبع طوری رفتار شود که حداقل یکی از چهار شرط بن بست رخ ندهد.

۳- کشف بن بست (Detection)

هرجا که ممکن باشد، منابع درخواستی به فرایندها داده می شود. سیستم عامل به طور متناوب الگوریتمی را دنبال می کند تا وجود شرط انتظار چرخشی را کشف کند.

مثال

مثالی از کاربرد الگوریتم کشف بن بست (Deadlock Detection):

سیستمی با ۳ فرایند و ۴ نوع منبع مفروض است. تعداد موجود از هر منبع در زیر آمده است:

Tape drives=4 , Plotters=2 , Printers=3 , CD ROM=1

که این موضوع به صورت $A=(4 \ 2 \ 3 \ 1)$ نشان داده می شود.

در این سیستم، فرایندها منابع زیر را در اختیار دارند:

فرایند اول = یک پرینتر، فرایند دوم = دو Tape و یک CD ، فرایند سوم = یک پلاتر و دو پرینتر

این موضوع با ماتریس زیر نشان داده می شود:

$$C = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 1 & 2 & 0 \end{bmatrix}$$

بعد از این تخصیص از هر یک از منابع به تعداد زیر باقی می ماند:

Tape drives =2 , Plotters=1 , Printers=0 , CD ROM =0

اما فرایندها برای اجرا نیاز به منابع دیگری دارند که در زیر مشخص شده است:

فرایند اول = دو Tape و یک CD, فرایند دوم = یک Tape و یک پرینتر،

فرایند سوم = دو Tape و یک پلاتر

که این موضوع با ماتریس زیر نشان داده می شود:

$$R = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 2 & 1 & 0 & 0 \end{bmatrix}$$

با توجه به اطلاعات بالا، آیا در این سیستم بن بست رخ می دهد؟

حل: فرایند اول نمی تواند اجرا شود، چون نیاز به یک CD دارد که موجود نیست. فرایند دوم نیز نمی تواند

اجرا شود، چون نیاز به یک پرینتر دارد که موجود نیست. ولی فرایند سوم می تواند اجرا شود، چون به دو

tape و یک پلاتر نیاز دارد که موجود است. بعد از اجرای فرایند سوم ، منابعی که در اختیار داشته (یک پلاتر

و دو پرینتر)، آزاد شده و تعداد منابع آزاد به صورت زیر در می آید:

Tape drives =2 , Plotters=2 , Printers=2 , CD ROM =0

در این وضعیت، دو فرایند دیگر می توانند اجرا شوند و بنابراین بن بست رخ نمی دهد.



ترمیم

بعد از کشف بن بست، روشهای زیر برای ترمیم آن وجود دارد:

- الف- قطع تمام فرایندهای بن بست.
- ب- برگشت فرایندهای بن بست به نقاطی که از قبل برای بررسی تعریف شده و شروع مجدد تمام فرایندها.
- ج- قطع پی در پی فرایندهای بن بست تا جایی که دیگر بن بست وجود نداشته باشد.
- د- قبضه کردن پی در پی منابع تا جایی که دیگر بن بست وجود نداشته باشد.

روش های پیشگیری از بن بست

روشهای پیشگیری از بن بست عبارتند از:

- الف- غیر مستقیم (پیشگیری از بروز یکی از سه شرط لازم اول)
- ب- مستقیم (پیشگیری از بروز شرط چهارم)

روشهای پیشگیری برای هر یک از چهار شرط بن بست

۱- نگهداری و انتظار

برای پیشگیری از رخ دادن این شرط، باید فرایند را مجبور کرد تا همه منابع مورد نیاز را یکباره درخواست کند و تا زمانی که همه منابع به او داده نشود، فرایند را مسدود کرد.

۲- انحصاری بودن (قبضه نکردن)

برای پیشگیری از بروز این شرط دو راه پیشنهاد شده است:

- الف- اگر فرایندی منابعی را در اختیار دارد، درخواست جدیدش موافقت نشود. در این حالت باید منابع قبلی خود را آزاد کند و در صورت نیاز، مجدداً با منابع اضافی درخواست نماید.
- ب- اگر فرایندی منبعی را درخواست کند که در اختیار فرایند دیگر است، سیستم عامل آن فرایند را قبضه کرده و منابعش را آزاد کند. (البته فرایندها نباید اولویت یکسان داشته باشند).

۳- انحصار متقابل

منابع غیرقابل اشتراک باید انحصاری باشند و نمی‌توان این شرط را رد کرد. به طور نمونه یک چاپگر نمی‌تواند همزمان بین چند فرایند مشترک باشد. و یا به یک فایل نمی‌توان توسط چند فرایند جهت نوشتن دسترسی پیدا کرد.

۴- انتظار چرخشی

برای پیشگیری از بروز این شرط، یک ترتیب خطی از انواع منابع تعریف می‌کنیم. اگر منبعی از نوع R به یک فرایند تخصیص یابد، در ادامه تنها منابعی را می‌تواند درخواست کند که نوع آنها بعد از R قرار گرفته باشد. مثلاً اگر $i < j$ آنگاه R_i قبل از R_j است. اگر فرایند P_1 منبع R_i را در اختیار داشته باشد و R_j را تقاضا کند و فرایند P_2 منبع R_j را در اختیار داشته باشد و R_i را تقاضا کند، این حالت غیر ممکن است چون مستلزم $i < j$ و $j < i$ می‌باشد.

پیشگیری از شرط انتظار چرخشی، ممکن است موجب کند کردن فرایندها و رد کردن غیر ضروری دسترسی به منابع گردد.

روش های اجتناب از بن بست

برای اجتناب از بن بست دو رویکرد وجود دارد:

- ۱- عدم شروع فرایندی که ممکن است درخواست هایش منجر به بن بست شود.
- ۲- عدم پاسخ به درخواست های منبع اضافی از طرف فرایندی که با این تخصیص ممکن است منجر به بن بست شود. (الگوریتم بانکداران)

در اجتناب از بن بست، تصمیم‌گیری زیر بصورت پویا انجام می‌شود:
 " اگر منبع درخواست شده، تخصیص داده شود، آیا می‌تواند منجر به بن بست شود یا نه ".
 یعنی اجتناب از بن بست نیازمند اطلاع از درخواستهای آینده است.

محدودیت های اجتناب از بن بست

- ۱- تعداد منابع تخصیص باید ثابت باشد.
- ۲- فرایندی که منبعی در اختیار دارد نمی‌تواند خارج شود.
- ۳- حداکثر منابع مورد نیاز هر فرایند باید از قبل معلوم شود.
- ۴- فرایندهای مورد نظر باید مستقل باشند.

امتیاز اجتناب از بن بست این است که قبضه کردن فرایند، لازم نمی‌باشد. (بر خلاف کشف)

حالت امن، حالتی است که در آن حداقل یک ترتیب از فرایندها وجود دارد که می‌توانند اجرا و کامل شوند، بدون اینکه بن بست رخ دهد.

وضعیت بن بست، یک وضعیت ناامن است، اما تمام وضعیت های ناامن، الزما، بن بست نیستند، چون ممکن است یک فرایند به حداکثر منابع اش نیاز پیدا کند و یا حتی بخشی از منابع تخصیص یافته اش را آزاد کند.

الگوریتم بانکداران

در این الگوریتم که برای اجتناب از بن بست استفاده می شود از بردارها (آرایه یک بعدی) و ماتریسهای (آرایه دو بعدی) زیر استفاده می شود:

- ۱- بردار Resource : شامل مقدار کل هر یک از منابع.
- ۲- بردار Available : شامل مقدار کل هر یک از منابعی که موجود است (تخصیص داده نشده)
- ۳- ماتریس Claim (حداکثر نیاز): شامل درخواستهای فرایندها.
- ۴- ماتریس Allocation : شامل منابع تخصیص داده شده به فرایندها.

مثال ۵-۱۴

در سیستم زیر با چهار فرایند و سه منبع آیا حالت امن وجود دارد؟ (Resource:R1=9,R2=3,R3=6)

	R_1	R_2	R_3
P1	3	2	2
P2	6	1	3
P3	3	1	4
P4	4	2	2

Claim

	R_1	R_2	R_3
P1	1	0	0
P2	6	1	2
P3	2	1	1
P4	0	0	2

Allocation

حل:

با توجه به ماتریس Allocation ، مشخص می شود که تعداد نه منبع R_1 ، دو منبع R_2 ، پنج منبع R_3 تخصیص داده شده اند. بنابراین با توجه به منابع اولیه، منابع بعد از تخصیص به صورت ($R_1=0, R_2=1, R_3=1$) می باشد. سپس ماتریس NEED را از تفاضل دو ماتریس Claim و Allocation بدست می آوریم:

فرایند	R_1	R_2	R_3
P1	2	2	2
P2	0	0	1
P3	1	0	3
P4	4	2	0

حال با توجه به ماتریس NEED و تعداد منابع آزاد بعد از تخصیص، مشاهده می شود که اجرای فرایند P1 در حال حاضر ممکن نیست، چون هیچ منبع R_1 آزادی نداریم. ولی منابع مورد نیاز P2 موجود است. بعد از اجرای P2، منابعی که در اختیار داشته آزاد شده و به Available اضافه می کند:

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

Claim

1	0	0
0	0	0
2	1	1
0	0	2

Allocation

6	2	3
---	---	---

Available

توجه شود که بعد از اجرای P2، سطر مربوط به فرایند P2 در هر دو ماتریس، صفر شدند. در این حالت P1 می‌تواند اجرا شود. بعد از اجرای P1، منابع در اختیار او به بردار Available اضافه می‌شود:

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

Claim

0	0	0
0	0	0
2	1	1
0	0	2

Allocation

7	2	3
---	---	---

Available

و بعد از اجرای P3 داریم:

0	0	0
0	0	0
0	0	0
4	2	2

Claim

0	0	0
0	0	0
0	0	0
0	0	2

Allocation

9	3	4
---	---	---

Available

در نهایت P4 اجرا می‌شود و بردار Available مانند بردار Resource خواهد شد.
بنابراین می‌توان ترتیب زیر را یک حالت امن سیستم نام برد:

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



هریک از روشهای برخورد با بن بست (پیشگیری، اجتناب و کشف) دارای مزایا و معایبی هستند. بنابراین بهتر است از روشهای متفاوتی در شرایط متفاوت استفاده کرد.

خلاصه رویکردها

در رابطه با سه رویکرد پیشگیری، کشف و اجتناب طرحهای مختلفی ارائه شده که در جدولهای زیر بررسی شده اند:

اجتناب		
معایب	مزایا	طرح
ضرورت اطلاع از منابع مورد نیاز آینده		
امکان مسدود شدن طولانی فرایندها		

کشف		
معایب	مزایا	طرح
	عدم تاخیر در آغاز فرایند	
	تسهیل پردازش در حین کار	

پیشگیری		
معایب	مزایا	طرح ها
نا کارآمدی	در مورد فرایندهایی که فعالیت شایعی را انجام می دهند خوب کار می کند.	
تاخیر شروع فرایند	عدم نیاز به قبضه کردن	
قبضه کردن بیش از تعداد لازم		
در معرض شروع شدنهای مجدد مدور		
قبضه کردن نه چندان مفید	امکان اعمال کنترل‌های زمان ترجمه	
اجازه ندادن به درخواست منابع به صورت افزایشی	عدم نیاز به محاسبه در زمان اجرا ، به دلیل اینکه مساله در طرح سیستم حل شده است.	

کنکور ارشد

(مهندسی IT – دولتی ۸۸)

۱- سیستمی با 3 فرایند و 2 فایل read-only را در نظر بگیرید. با فرض اینکه هر فرایند حداکثر به خواندن 2 فایل نیاز داشته باشد، تعداد وضعیت های بن بست (Deadlock) حداکثر برابر کدام است؟

- (۱) 0 (۲) 3 (۳) 4 (۴) 5

حل: گزینه ۱ درست است.

چون فایل فقط خواندنی است، شرط انحصار متقابل برای آن مطرح نبوده و هرگز بن بست رخ نمی دهد.

(مهندسی کامپیوتر – آزاد ۸۹)

۲- سیستمی شامل 4 فرایند همروند و 2 منبع یکسان قابل استفاده مجدد را در نظر بگیرید به شرط آن که هر فرایند حداکثر به 2 منبع نیاز داشته باشد، تعداد وضعیت های بن بست (deadlock states) در این سیستم به خاطر منبع مذکور حداکثر چند حالت می باشد؟

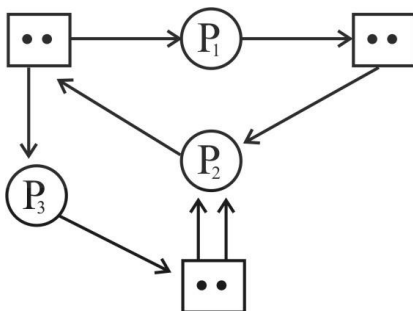
- (۱) 10 (۲) 5 (۳) 0 (۴) 6

حل: جواب گزینه ۴ است.

$$\binom{4}{2} = \frac{4!}{2! \times 2!} = 6$$

(مهندسی کامپیوتر – آزاد ۸۴)

۳- اگر گراف بن بست زیر نمایش لحظه‌ای درخواست‌های منابع و نیز تملک منابع توسط سه فرایند P_1 و P_2 و P_3 باشد و در صورتی که فقط با اتمام یک فرایند، منابع تملک شده توسط آن فرایند آزاد شوند، آن گاه ترتیب اتمام فرایندها از راست به چپ کدام است؟



(۲) P_3 و P_2 و P_1

(۱) P_2 و P_3 و P_1

(۴) هیچ کدام، زیرا سیستم در وضعیت بن بست فرار دارد.

(۳) P_3 و P_1 و P_2

حل: جواب گزینه ۲ است.

منبع سمت چپ بالا را R_1 و منبع سمت راست بالا را R_2 و منبع پایین را R_3 می‌نامیم. دو نمونه منبع R_1 در اختیار فرایندهای P_1 و P_3 می‌باشد. فرایند P_2 و P_3 نمی‌تواند ابتدا اجرا شوند، چون منبع درخواستی آنها آزاد نیست. اما P_1 یک نمونه از R_2 را که آزاد است گرفته و اجرا می‌شود. بعد از پایان اجرایش، منبع R_1 که در اختیار داشت را رها کرده و P_2 با گرفتن آن اجرا می‌شود. بعد از پایان اجرایش، منبع R_3 که در اختیارش بود را آزاد کرده و P_3 با گرفتن یک نمونه از R_3 اجرا می‌شود. بنابراین P_1 ، P_2 و در نهایت P_3 اجرا می‌شود.

(مهندسی IT – دولتی ۸۷)

۴- اگر در سیستم عاملی به هر منبع یک شماره اولویت منحصر به فردی اختصاص داده شود و از پردازش درخواست معینی با اولویت کمتر یا مساوی اولویت منبع hold شده توسط همان فرایند ممانعت به عمل آید، کدام یک از گزینه‌های زیر صحیح است؟

- (۱) این روش از بن بست جلوگیری می‌کند ولی احتمال گرسنگی وجود دارد.
 - (۲) این روش موسوم به درخواست افزایش است و جهت پیشگیری از بن بست به کار می‌رود.
 - (۳) این روش مبتنی بر کشف بن بست و بدین ترتیب عامل‌های بن بست تشخیص داده می‌شوند.
 - (۴) این روش موسوم به درخواست افزایش است و به صورت دینامیکی از بن بست اجتناب می‌کند.
- حل: جواب گزینه ۲ است.

(مهندسی IT – دولتی ۸۴)

۵- برای پیشگیری از بن بست از طریق شرط انتظار چرخشی، روش مبتنی بر شماره گذاری همه منابع و الزام فرایندها به درخواست منابع به ترتیب عددی (مثلا صعودی)، پیشنهاد شده است. در پیاده سازی آن با چه مشکلی روبرو خواهیم شد؟

- (۱) منابع سیستم را هدر می‌دهد.
 - (۲) نیاز به پیش بینی آینده دارد.
 - (۳) همه منابع قابل Spool نیست و خود فضای Spool بر روی دیسک نیز موجب بن بست می‌شود.
 - (۴) هر دو مورد ۱ و ۲ صحیح است.
- حل: جواب گزینه ۴ است.

(مهندسی IT - دولتی ۸۴)

۶- تحت سیستم عاملی 4 فرایند فعال و 2 منبع مدیریت می شود. وضعیت سیستم تحت جدول ذیل بیان شده است. حالت سیستم چیست؟ منابع در دسترس عبارتند از: $(R_1 = 2, R_2 = 1)$

	R_1	R_2
P_1	7	2
P_2	1	3
P_3	1	1
P_4	3	0

مقادیر اختصاص داده

شده

	R_1	R_2
P_1	9	5
P_2	2	6
P_3	2	2
P_4	5	0

حداکثر نیاز

(۲) نا امن

(۱) امن

(۳) به طور قاطع نمی توان پیش بینی کرد. (۴) بستگی دارد چه فرایندی چه منبعی را تقاضا کند.

حل: جواب گزینه ۲ است.

با تفاضل دو ماتریس داده شده، ماتریس Need به صورت زیر مشخص می شود:

	R_1	R_2
P_1	2	3
P_2	1	3
P_3	1	1
P_4	2	0

با توجه به ماتریس need و منابع در دسترس یعنی $(R_1 = 2, R_2 = 1)$ ، مشخص است که P_1 یا P_2 نمی توانند اجرا شوند، چون به سه R_2 نیاز دارند در حالی که فقط یکی موجود است. اما P_3 می تواند اجرا شود. بعد از اجرای P_3 ، منابعی که در اختیار داشته را آزاد می کند و منابع در دسترس $(R_1 = 3, R_2 = 2)$ می شوند. بعد از اجرای P_3 ، باز هم P_1 یا P_2 نمی توانند اجرا شوند، چون باز هم سه R_2 نداریم. بنابراین P_4 را اجرا می کنیم. بعد از اجرای P_4 ، منابع در دسترس $(R_1 = 6, R_2 = 2)$ خواهد شد، باز هم نمی توان P_1 یا P_2 را اجرا کرد. در نتیجه سیستم در وضعیت نا امن قرار دارد. ■

(مهندسی IT – آزاد ۸۹)

۷- وضعیت زیر یک وضعیت.....بردار منابع: $(A=5, B=5, C=2, D=4)$

	A	B	C	D
P1	4	4	2	5
P2	2	0	0	2
P3	1	1	1	2
P4	3	1	0	3
P5	1	1	0	1

ماتریس حداکثر نیاز

	A	B	C	D
P1	2	1	1	4
P2	0	0	0	2
P3	1	1	1	1
P4	2	0	0	2
P5	0	1	0	0

ماتریس تخصیص

(۱) امن است. (۲) بن بست است. (۳) نا امن است. (۴) گرسنگی است.

حل: جواب گزینه ۱ است.

با تفاضل دو ماتریس داده شده، ماتریس Need به صورت زیر مشخص می شود:

	A	B	C	D
P1	2	3	1	1
P2	2	0	0	0
P3	0	0	0	1
P4	1	1	0	1
P5	0	0	0	1

با توجه به ماتریس need و منابع در دسترس یعنی $(A = 5, B = 5, C = 2, D = 4)$ ، مشخص است که تعداد منابع به اندازه ای است که می توان به هر یک از فرایندها به هر ترتیبی اختصاص داد. در نتیجه سیستم امن است. یکی از این ترتیب ها:

۱- P1 اجرا شده و بعد از پایان اجرایش منابع در دسترس برابر خواهند بود:

$$(A = 7, B = 6, C = 3, D = 8)$$

۲- P2 اجرا شده و بعد از پایان اجرایش منابع در دسترس برابر خواهند بود:

$$(A = 7, B = 6, C = 3, D = 10)$$

۳- P3 اجرا شده و بعد از پایان اجرایش منابع در دسترس برابر خواهند بود:

$$(A = 8, B = 7, C = 4, D = 11)$$

۴- P4 اجرا شده و بعد از پایان اجرایش منابع در دسترس برابر خواهند بود:

$$(A = 10, B = 7, C = 4, D = 13)$$

۵- P5 اجرا شده و بعد از پایان اجرایش منابع در دسترس برابر خواهند بود:

$$(A = 10, B = 8, C = 4, D = 13)$$



فصل ۶:

مدیریت حافظه

حافظه، آرایه بزرگی از کلمات یا بایت‌ها است. یکی از وظایف سیستم عامل، مدیریت حافظه است. مدیریت حافظه، در سیستم چند برنامه‌ای باید بسیار کارآمد باشد و حافظه به نحوی باید تخصیص یابد که فرایندهای بیشتری در آن قرار بگیرند.

انواع حافظه عبارتند از: ثابت، حافظه نهان، حافظه اصلی، حافظه دیسک که در فصل اول بررسی شدند. مدیریت ثابت بر عهده کامپایلر، مدیریت حافظه نهان، سخت‌افزاری، مدیریت حافظه اصلی و مدیریت حافظه دیسک بر عهده سیستم عامل است. حافظه‌هایی که مدیریت آنها بر عهده سیستم عامل است را در این فصل و فصل‌های بعد بررسی خواهیم کرد.

مدیریت حافظه ابتدایی

از آنجا که در بعضی سیستم‌های امروزی، از طرح‌های ساده مدیریت حافظه، استفاده می‌شود، این طرح‌ها را مطالعه می‌کنیم.

تک برنامه‌نگی ساده

ساده‌ترین طرح مدیریت حافظه این است که در هر لحظه، فقط یک برنامه در حال اجرا باشد. امروزه از طرح تک برنامه‌نگی بیشتر در سیستم‌های توکار استفاده می‌شود.

سه مدل سازماندهی حافظه شامل یک سیستم عامل و یک فرایند کاربر عبارتند از:

- ۱- سیستم عامل در پایین حافظه و در RAM باشد.
- ۲- سیستم عامل در بالای حافظه و در ROM باشد.
- ۳- سیستم عامل در پایین حافظه و در RAM باشد. گرداننده‌های دستگاه در بالای حافظه و در ROM باشند. (در MS-DOS از این مدل استفاده می‌شود).

از معایب مدیریت حافظه به روش تک برنامه‌نگی، اتلاف شدید حافظه و سایر منابع است. همچنین امکان اجرای فرایندهای بزرگتر از حافظه اصلی وجود ندارد.

چند برنامه‌گی با پارتیشن ثابت و بدون مبادله

برای اجرای چندبرنامه‌گی می‌توان حافظه را به n پارتیشن تقسیم کرد. اندازه این پارتیشن‌ها می‌تواند یکسان نباشد. وقتی کاری وارد می‌شود، در یک صف ورودی قرار می‌گیرد تا در کوچکترین پارتیشنی که در آن جا می‌شود، قرار داده شود. مقداری از پارتیشن که توسط کار درون آن استفاده نمی‌شود، هدر می‌رود که به آن حفره (Hole) می‌گویند.

در یک سیستم با پارتیشن ثابت، هر پارتیشن می‌تواند، صف مربوط به خود را داشته باشد و یا یک صف مشترک برای همه پارتیشن‌ها در نظر گرفت. در حالت اول، ممکن است صف مربوط به یک پارتیشن بزرگ خالی باشد، در صورتی که صف مربوط به یک پارتیشن کوچک پر شده باشد.

مدیریت حافظه به روش پارتیشن بندی ایستا، امکان ایجاد چندبرنامه‌گی با یک تکنیک ساده را فراهم می‌کند و در مقایسه با تک برنامه‌گی از حافظه و پردازنده بهتر استفاده می‌شود. اما دارای معایب زیر است:

- ۱- تعیین تعداد و اندازه پارتیشن‌ها، مشکل است.
- ۲- درجه چند برنامه‌گی به تعداد پارتیشن‌ها محدود است.
- ۳- نمی‌توان فرایند بزرگتر از بزرگترین پارتیشن را اجرا کرد.
- ۴- نمی‌توان یک فرایند را دو تکه کرد و در دو پارتیشن مجزا قرار داد.
- ۵- اتلاف حافظه به علت بارگذاری قسمت‌هایی از یک فرایند بزرگ که فعلاً به آنها نیاز نیست.
- ۶- تکه تکه شدن داخلی

تذکر: به این علت به این تکه تکه شدن، داخلی می‌گویند که از حفره داخل فضای تخصیص یافته به فرایند ناشی می‌شود.

جا به جایی و حفاظت

در چندبرنامه‌گی دو مشکل جا به جایی و حفاظت رخ می‌دهد که باید حل شوند.

جا به جایی (Relocation)

وقتی که برنامه اصلی با زیر برنامه‌های نوشته شده توسط کاربر و رویه‌های کتابخانه‌ای در یک فضای آدرس ترکیب می‌شوند، مشکلی که به وجود می‌آید این است که پیوند دهنده از کجا باید بداند که آن برنامه در زمان اجرا از چه آدرسی در حافظه شروع می‌شود. برای حل این مشکل، سیستم عامل و سخت افزار باید بتوانند مراجعات به حافظه را که در کد برنامه مطرح می‌شوند را به آدرس‌های فیزیکی که منعکس کننده مکان فعلی برنامه در حافظه اصلی هستند، تبدیل کند. یک راه حل نرم افزاری این است که در همان زمان لود برنامه در حافظه، دستورات برنامه دستکاری شده و آدرس‌ها اصلاح شوند.

حفاظت (Protection)

در سیستم‌های چند کاربره، از حافظه متعلق به برنامه یک کاربر، در مقابل خواندن یا نوشتن برنامه کاربران دیگر باید حفاظت کرد.

یک راه حل:

حافظه به بلوک‌های دو کیلو بایتی که هر بلوک شامل یک کد حفاظتی چهار بیتی است، تقسیم شود. PSW نیز شامل یک کلید 4 بیتی است. همچنین فقط سیستم عامل قادر به تغییر کلید و کد حفاظتی باشد. در این صورت هر دستور فرایند در حال اجرا، که منجر به دسترسی به بلوکی از حافظه شود که کد حفاظتی آن بلوک با کلید درون PSW یکسان نباشد، متوقف شود.

راه حل سخت افزاری دو مشکل حفاظت و جا به جایی

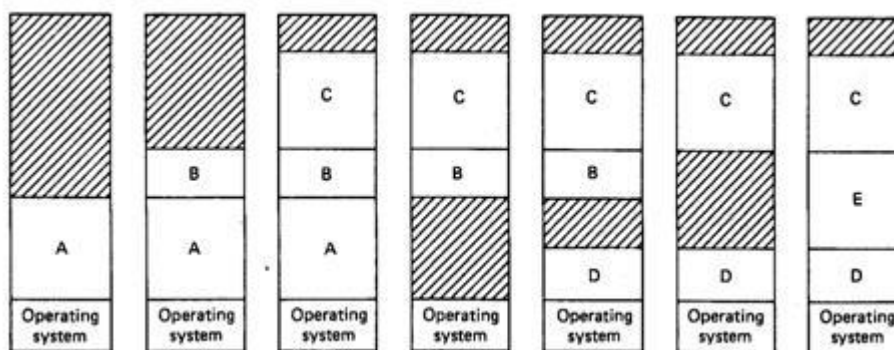
می‌توان از دو رجیستر پایه (Base) و حد (Limit) واقع در MMU برای حل این دو مشکل استفاده کرد. هنگامی که پردازنده به فرایندی داده می‌شود، آدرس شروع پارتیشن در Base و طول پارتیشن در Limit قرار می‌گیرد. برای جلوگیری از مشکل جا به جایی، هر آدرس حافظه‌ای که تولید می‌شود، قبل از ارسال به حافظه، مقدارش به صورت خودکار با محتوای Base جمع می‌شود. برای مشکل حفاظت، آدرس‌ها با محتوای Limit مقایسه می‌شوند تا به فضای خارج از پارتیشن دسترسی نشود. اگر ثبات پایه حاوی 100 و ثبات حد شامل 20 باشد، آن گاه برنامه می‌تواند به آدرس‌هایی از 100 تا خود 120، دستیابی داشته باشد.

مبادله (swapping)

در سیستم‌های اشتراک زمانی در گاهی اوقات نمی‌توان همه فرایندهای فعال را در حافظه اصلی جای داد و باید فرایندهای اضافی به دیسک منتقل شده و بعداً به صورت پویا به داخل حافظه آورده شوند. مبادله (پارتیشن بندی پویا)، ساده‌ترین روشی است که اجازه مبادله بین حافظه و دیسک را می‌دهد. در این روش هر فرایند به طور کامل به حافظه اصلی آورده می‌شود (Swap in) و بعد از مدتی اجرا، به دیسک برگردانده می‌شود (Swap out).

مثلاً در یک محیط چند برنامه‌ای که از الگوریتم زمانبندی RR استفاده می‌کند، وقتی هر فرایندی کوانتوم زمانی خودش را تمام می‌کند، با فرایند دیگری مبادله می‌شود.

شکل زیر نحوه عملکرد سیستمی را نشان می‌دهد که بر اساس مبادله کار می‌کند. در ابتدا فقط فرایند A در حافظه است و سپس فرایندهای B و C، وارد حافظه شده‌اند. سپس فرایند A، خارج شده و بعد D وارد شده و سپس B خارج شده است. در انتها، فرایند E وارد شده است.



اگر انقیاد در زمان اسمبل یا بار کردن باشد، وقتی فرایندی از حافظه بیرون رفت، هنگام برگشت به حافظه، در همان فضای قبلی بر می‌گردد. اگر انقیاد در زمان اجرا صورت گیرد، فرایند در محلی غیر از محل اول قرار می‌گیرد، زیر آدرسهای فیزیکی در زمان اجرا محاسبه می‌شوند.

در پارتیشن بندی پویا، تعداد، موقعیت و اندازه پارتیشن‌ها متفاوت است. این انعطاف پذیری باعث بهبود بهره‌وری حافظه می‌شود.

تخصیص و آزاد سازی حافظه در پارتیشن بندی پویا، پیچیده تر از پارتیشن بندی ایستا است.

در پارتیشن بندی پویا، به خاطر مبادله، حفره‌های متعددی در حافظه به وجود می‌آید که باعث در رفتن حافظه می‌شود. به این مشکل، تکه شدن خارجی می‌گویند. علت اینکه به آن خارجی می‌گویند این است که تکه شدن از حفره خارج فضای تخصیص یافته به فرایندها ناشی می‌شود.

برای مقابله با تکه شدن خارجی از فشردن سازی (Compaction) می‌توان استفاده کرد. در این روش، حفره‌های کوچک با یکدیگر ادغام می‌شوند تا یک بلوک بزرگ از حافظه را ایجاد کنند. ساده‌ترین الگوریتم فشردن سازی این است که تمام حفره‌ها را به یک طرف برده و حفره بزرگی از حافظه آزاد تشکیل می‌شود. ولی این روش هزینه زیادی دارد.

وقتی که حافظه به صورت پویا تخصیص داده می‌شود، سیستم عامل باید بداند که کدام بخش حافظه در هر لحظه، تخصیص داده شده و کدام بخش آزاد است. برای این منظور، دو روش وجود دارد:

۱- مدیریت حافظه با نگاشت های بیتی

نگاشت بیتی (Bitmap) راه حلی را ارائه می‌دهد تا بتوان به سادگی و با استفاده از مقدار ثابتی از حافظه، وضعیت پر و خالی بودن کلمات حافظه را تحت نظر داشت. در این روش، حافظه به چندین واحد تخصیص تقسیم می‌شود. متناظر با هر واحد تخصیص، یک بیت وجود دارد. اگر واحد متناظر آزاد باشد، این بیت 0 است و در صورت پر بودن، 1 است.

مثال

سیستمی از الگوریتم مدیریت حافظه نگاشت بیتی (Bitmap) به صورت زیر استفاده می‌کند:

11001011,00000111

هر بیت نشان دهنده فضای پر (1) یا خالی (0) به ازاء هر 4KB حافظه اصلی است. لیست حفره‌ها به چه صورت است؟

پاسخ:

با نگاه به bitmap داده شده از چپ به راست، سه دسته صفر داریم :

1 100101 1, 000001 1 1

۱- دسته اول: دو صفر (8KB)

۲- دسته دوم: یک صفر (4KB)

۳- دسته سوم: پنج صفر (20KB)

بنابراین لیست حفره‌ها برابر است با: 8,4,20.

هر چه واحد تخصیص کوچکتر باشد، نگاشت بیتی بزرگتر خواهد بود. در مقابل، اگر واحد تخصیص

بزرگ انتخاب شود، نگاشت بیتی کوچکتر خواهد بود.

اگر اندازه فرایند، ضریبی از اندازه واحد تخصیص نباشد، در آخرین واحد مقداری از حافظه هدر می

رود.

برای انتقال یک فرایند به حافظه که به k واحد فضا نیاز دارد، مدیر حافظه باید در نگاشت بیتی

جستجو کرده و k بیت متوالی 0 را پیدا کند. این عمل بسیار کند است و باعث مخالفت از طرح Bitmap

می‌باشد.

۲- مدیریت حافظه با لیست‌های پیوندی

در این روش، یک لیست پیوندی از قطعه‌های آزاد یا تخصیص یافته حافظه تشکیل می‌شود. مزیت این

روش این است که زمانی که فرایندی خاتمه می‌یابد، یا مبادله می‌شود، این لیست به سادگی update می

شود. فرایندی که پایان یافته با همسایه‌های خود ترکیب می‌شود. فیلدهای هر گره در لیست پیوندی مثال

قبل شامل فیلدهای زیر است:

۱- حفره (H) یا فرایند (P) بودن

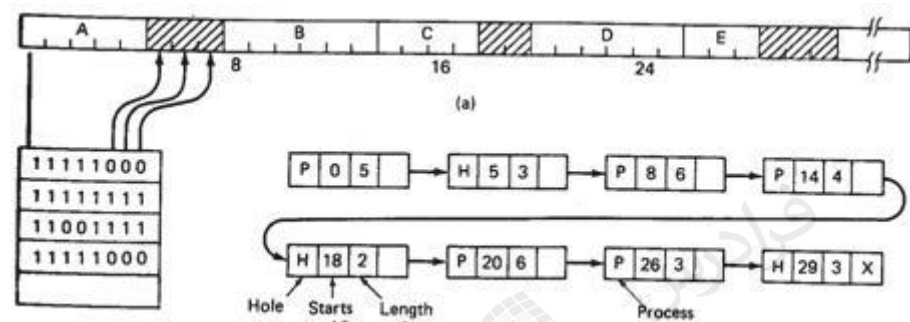
۲- آدرس شروع

۳- طول

۴- آدرس گره بعدی

مثال

شکل زیر قسمتی از حافظه شامل 5 فرایند و 3 حفره را نشان می‌دهد. مناطق هاشور خورده، فضاهای خالی هستند. این حافظه به دو روش Bitmap و لیست پیوندی نشان داده شده است. در طرح Bitmap فضاهای خالی، با صفر و فضاهای پر با 1 مشخص شده‌اند. به طور مثال، 8 بیت اول، شامل پنج تا 1 و سه تا 0 است، چون پنج واحد تخصیص توسط فرایند A پر شده و بعد از آن ۳ واحد خالی می‌باشد. در طرح لیست پیوندی، فضاهای خالی با H و فضاهای پر با P مشخص شده‌اند.



الگوریتم‌های مکان‌یابی و تخصیص حافظه

وقتی فرایندها و حفره‌ها در یک لیست مرتب شده بر اساس آدرس قرار می‌گیرند، الگوریتم‌های مختلفی جهت تخصیص حافظه به یک فرایند وجود دارد. چند مورد از این الگوریتم‌ها عبارتند از:

۱- اولین برازش (First fit)

جستجو از ابتدای حافظه شروع شده و فرایند در اولین حفره‌ای قرار داده می‌شود که در آن جا می‌شود.

۲- برازش بعدی (Next fit)

مانند First fit است، با این تفاوت که جستجو از آخرین محل تخصیص شروع می‌شود.


۳- بهترین برازش (Best fit)

تمام لیست جستجو می‌شود و فرایند در کوچکترین حفره‌ای قرار داده می‌شود که در آن جا می‌شود.

۴- بدترین برازش (Worst fit)

تمام لیست جستجو می‌شود و فرایند در بزرگترین حفره‌ای قرار داده می‌شود که در آن جا می‌شود.

الگوریتم First fit، ساده‌ترین روش و معمولاً سریع‌ترین الگوریتم است. 

الگوریتم First fit، بیشترین کارایی را دارد. یعنی بهره‌وری حافظه در آن قابل قبول است. 

در Best fit ، فضاهای حافظه پر از تکه های کوچک خالی بی مصرف می شود.

سرعت الگوریتم های Best fit و Worst fit پایین است، چون کل لیست باید جستجو شود.

در Next fit ، حفره های بزرگ انتهای حافظه سریع تر شکسته می شود و در ورود فرایندهای بزرگ بعدی، مشکل ایجاد می شود.

اگر برای فضاهای خالی و فرایندها، لیست های جداگانه ای داشته باشیم، سرعت هر چهار الگوریتم ، بیشتر می شود. البته این کار باعث پیچیده شدن مکانیسم آزاد کردن حافظه می شود.

اگر لیست فضاهای خالی بر اساس اندازه فضاها مرتب باشد، سرعت Best fit ، افزایش می یابد.

مثال

در یک سیستم که مدیریت حافظه با استفاده از مبادله انجام می شود، حافظه اصلی شامل فضاهای خالی با اندازه های به ترتیب 12 , 9 , 7 , 18 , 20 , 4 , 10 (چپ به راست) است. برای درخواست تکه هایی از حافظه به طور متوالی و به مقدارهای 12 و 10 و 6 کیلو بایت با استفاده از روش های First fit ، Next fit ، Best fit و Worst fit کدام یک از فضاهای خالی فوق الذکر اشغال می شوند؟

پاسخ: روش First fit

تکه 12 از حفره 20 گرفته می شود و 8 کیلو باقی می ماند و تکه 10 از حفره 10 و تکه 6 از حفره 8 (باقیمانده از 20) گرفته می شود:

	1 0	4	2 0	18	7	9	12
12	1 0	4	<u>8</u>	18	7	9	12
10	<u>0</u>	4	8	18	7	9	12
6	0	4	<u>2</u>	18	7	9	12

روش Next fit :

	10	4	2 0	18	7	9	12
12	10	4	<u>8</u>	18	7	9	12
10	10	4	8	<u>8</u>	7	9	12
6	10	4	8	<u>2</u>	7	9	12

توجه کنید که فرایند 6KB از همان فضای باقی مانده از حفره مرحله قبل استفاده کرد.

روش Best fit :

	1 0	4	2 0	18	7	9	12
12	1 0	4	2 0	18	7	9	0
10	0	4	2 0	18	7	9	0
6	0	4	2 0	18	1	9	0

روش Worst fit :

	10	4	2 0	18	7	9	12
12	10	4	8	18	7	9	12
10	10	4	8	8	7	9	12
6	10	4	8	8	7	9	6

برازش سریع (Quick fit)

در این روش برای هر دسته از فرایندها با اندازه های متداول، یک لیست جداگانه تهیه می شود. جدولی با n خانه را فرض کنید که خانه اول این جدول، اشاره گری است که به ابتدای لیست حفره های 4 کیلو بایتی، خانه دوم، به ابتدای حفره های 8 کیلو بایتی و ... اشاره می کند.

در این روش یافتن حفره ای با اندازه مناسب، بسیار سریع است.

در این روش حافظه، تکه تکه شده و پر از حفره های کوچک و غیر قابل استفاده می شود.

مدیریت حافظه با سیستم رفاقتی

سیستم رفاقتی، یک تعادل قابل قبول برای فائق آمدن بر معایب طرحهای بخش بندی ایستا و پویا است. در بخش بندی ایستا تعداد فرایندهای فعال محدود است و امکان تکه تکه شدن داخلی وجود دارد و در بخش بندی پویا سربار فشرده سازی وجود دارد. در یک سیستم رفاقتی، اندازه بلاکهای حافظه توانی از 2 می باشند.

نحوه کار: اگر اندازه یک حفره برابر 2^k باشد و فرایندی به اندازه S باید به داخل آن مبادله شود، اگر S از نصف اندازه حفره بزرگتر بود، کل فضا به آن داده می شود، در غیر اینصورت، کل بلوک نصف شده و دو بلوک رفیق ایجاد می شود. این روند به صورت بازگشتی تکرار خواهد شد. در صورت آزاد شدن یک حفره، امکان ترکیب رفقای مجاور می باشد.

مثال

شکل زیر نحوه عملکرد سیستم رفاقتی را نشان می دهد:

	1024					
	512			512		
	256		256		512	
Request 70	A	128		256		512
Request 35	A	B	64	256		512
Request 80	A	B	64	C	128	512
Return A	128	B	64	C	128	512
Request 60	128	B	D	C	128	512
Return B	128	64	D	C	128	512
Return D	256			C	128	512
Return C	1024					

ابتدا بلوک 1024 K به دو بلوک 512K تقسیم شده و سپس بلوک اول به دو بلوک 256K و بلوک اول آن به دو بلوک 128K تقسیم می شود. که اولین درخواست (A) یعنی 70K در آن قرار می گیرد. برای پاسخ به درخواست بعدی (B) یعنی 35 K ، حفره 128 K به دو حفره 64 K تقسیم شده و B در حفره اولی قرار می گیرد. درخواست بعدی (C) یعنی 80 K ، در نیمه اول حفره 256 K قرار می گیرد. در این لحظه A آزاد شده و بعد به درخواست (D) یعنی 60K پاسخ داده می شود. با آزاد سازی B ، و بعد از آن D ، سه فضای خالی کنار هم ادغام شده و یک حفره 256K را ایجاد می کنند. در نهایت بعد از آزاد سازی C، بلوک یکپارچه می شود. ■

مزیت سیستم رفاقتی این است که چیدمان حفره ها ساخت یافته است و می توان یک ساختار درختی برای بلوک های پر و خالی ایجاد کرد و با یک الگوریتم بازگشتی، حفره مناسب را خیلی سریع پیدا کرد.

از معایب سیستم رفاقتی، اتلاف حافظه در این روش است. زمانی که یک فرایند 70 کیلو بایتی در یک حفره 128 کیلو بایتی قرار می گیرد، فضایی به اندازه 58 کیلو بایت هدر می رود.

روی هم گذاری (Overlay)

در گذشته، اگر حافظه تخصیص داده شده به یک فرایند از اندازه فرایند کوچکتر بود، از تکنیک Overlay استفاده می شد. در این تکنیک برنامه نویس، برنامه ها را به قسمت هایی به نام Overlay تقسیم می کند. ابتدا Overlay0 اجرا شده و بعد از پایان اجرایش، Overlay1 را صدا می زند و به همین ترتیب ادامه می یابد. در سیستم هایی که امکان نگهداری چندین Overlay به طور همزمان در حافظه وجود داشت، سیستم عامل Overlay ها را روی دیسک نگه می داشت و در مواقع لزوم، عملیات مبادله را انجام می داد. تقسیم برنامه به overlay ها و فراخوانی آنها، بر عهده برنامه نویس بود و سیستم عامل فقط عمل مبادله را انجام می دهد.

صفحه بندی (Paging)

در این روش مدیریت حافظه، حافظه اصلی به بلوک‌هایی با اندازه‌های ثابت به نام قاب (frame) تقسیم می‌شود. حافظه منطقی نیز به بلوک‌هایی با اندازه‌های یکسان به نام صفحه (page) تقسیم می‌شود. وقتی یک فرایند به داخل حافظه آورده می‌شود، تمام صفحات آن به داخل قابهای موجود بار می‌شوند و یک جدول صفحه ایجاد می‌شود. جدول صفحه، محل قاب هر صفحه از فرایند را مشخص می‌کند.

مثال

شکل زیر تخصیص قاب‌های آزاد به صفحه‌های فرایند A با سه صفحه را نشان می‌دهد. جدول صفحه را مشخص نمایید.

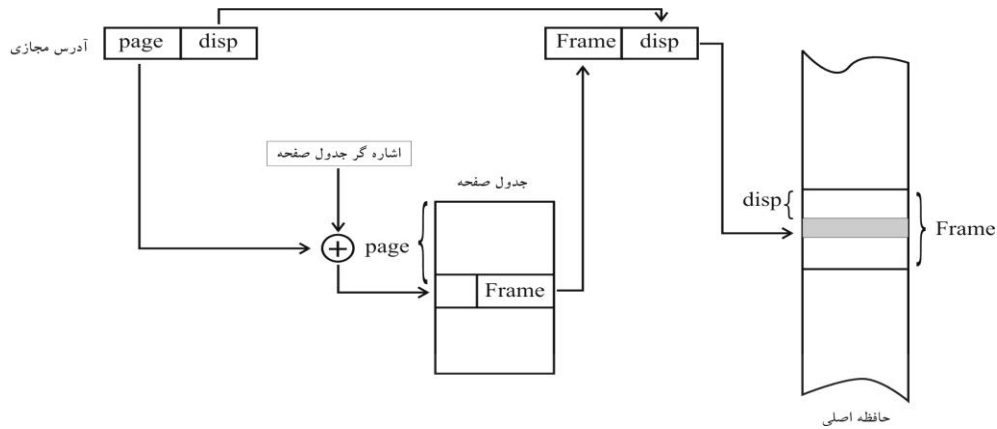
⋮	
5	A.0
6	
7	A.2
8	A.1
9	

پاسخ: جدول صفحه فرایند A با سه صفحه به صورت زیر است:

0	5
1	8
2	7

- آدرس منطقی در صفحه بندی از دو قسمت تشکیل می‌شود: شماره صفحه (P) و آفست (d).
- به آفست، انحراف یا تفاوت مکان نیز می‌گویند.
- اندازه صفحه و اندازه قاب توسط سخت افزار تعریف می‌شود.
- برای ساده شدن طرح صفحه بندی، اندازه صفحه و اندازه قاب باید توانی از ۲ باشند. در این حالت آدرس نسبی و منطقی یکسان هستند.
- پردازنده به کمک جدول صفحه، یک آدرس فیزیکی تولید می‌کند که این آدرس از دو قسمت شماره قاب و آفست تشکیل شده است.

شکل زیر نحوه ترجمه آدرس در سیستم صفحه بندی را نشان می‌دهد:



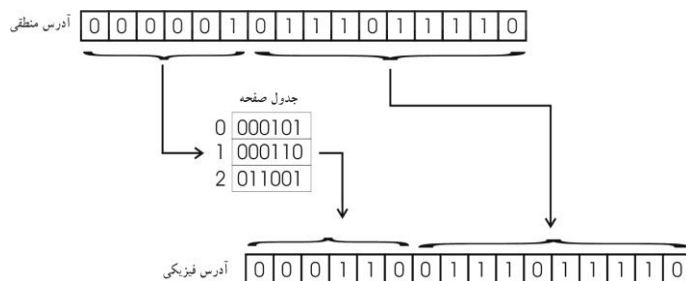
نحوه تولید آدرس فیزیکی با توجه به آدرس منطقی در شکل بالا: شماره صفحه موجود در آدرس مجازی (page) با آدرس شروع جدول صفحه، جمع شده و به آدرس بدست آمده، در جدول صفحه مراجعه می‌شود. در این آدرس شماره قاب (frame) متناظر با صفحه مورد نظر قرار دارد. این شماره قاب و همان آفست (disp) موجود در آدرس مجازی، آدرس فیزیکی را مشخص می‌کنند.

مثال

با توجه به آدرس نسبی 16 بیتی 0000010111011110، چند بیت برای شماره صفحه نیاز است؟ (اندازه صفحه برابر یک کیلو بایت می‌باشد).

پاسخ: چون اندازه صفحه یک کیلو بایت (2^{10} بایت) است، 10 بیت برای آفست مورد نیاز است و چون آدرس 16 بیتی است، 6 بیت برای شماره صفحه باقی می‌ماند. آدرس نسبی داده شده دارای آفست (0111011110) در صفحه (000001) است. شکل زیر نحوه تولید آدرس فیزیکی از آدرس منطقی را

نشان می‌دهد:



تذکره: یک برنامه می‌تواند حداکثر $2^6 = 64$ صفحه یک کیلوبایتی داشته باشد. ■

مثال

یک فضای آدرس دهی منطقی شامل 4 صفحه است و هر صفحه حاوی 2 کلمه است. اگر این صفحات بر روی یک فضای آدرس دهی فیزیکی حاوی 8 قاب صفحه نگاشت شود، آدرس منطقی و فیزیکی چند بیتی خواهد بود؟

$$\text{فضای آدرس دهی منطقی} = 4 \times 2 = 8 = 2^3$$

$$\text{فضای آدرس دهی فیزیکی} = 8 \times 2 = 16 = 2^4$$

بنابراین آدرس منطقی 3 بیتی و آدرس فیزیکی 4 بیتی است. ■

مثال

در یک سیستم حافظه صفحه بندی با یک جدول صفحه حاوی 64 مدخل 10 بیتی و صفحه های 512 بایتی، آدرس منطقی و فیزیکی چند بیتی است؟

اندازه حافظه منطقی برابر است با حاصل ضرب تعداد صفحات (تعداد مدخل ها) در اندازه هر صفحه:

$$\text{بایت} = 2^{15} = 2^6 \times 2^9 = 64 \times 512 = \text{اندازه حافظه منطقی}$$

اندازه حافظه فیزیکی برابر است با حاصل ضرب تعداد آدرسها در اندازه هر صفحه:

$$\text{بایت} = 2^{19} = 2^{10} \times 512 = \text{اندازه حافظه فیزیکی}$$

بنابراین آدرس منطقی 15 بیتی و آدرس فیزیکی 19 بیتی است. ■

مثال

در یک فضای آدرس دهی منطقی هر صفحه حاوی 512 کلمه است. این تعداد صفحات حافظه اصلی فضای آدرس دهی بر روی یک فضای آدرس دهی فیزیکی حاوی 8 قاب صفحه نگاشته می شود. آدرس فیزیکی حاوی چند بیت است؟

$$8 \times 512 = 2^3 \times 2^9 = 2^{12}$$

■

نقطه ضعف اصلی صفحه بندی تکه تکه شدن داخلی است. (اگر احتیاج به ناحیه بسیار کوچکی از حافظه باشد، در این صورت مقداری از فضای حافظه تلف می شود، زیرا کوچکترین واحدی از حافظه که به استفاده کننده اختصاص دارد یک صفحه است.)

حافظه مجازی

حافظه مجازی تکنیکی است که موجب می شود فرایند بدون اینکه کاملاً در حافظه باشد اجرا گردد. با استفاده از این تکنیک، می توان برنامه ای بزرگتر از حافظه فیزیکی را اجرا کرد. حافظه مجازی چندبرنامگی را به صورت موثری ممکن می سازد و کاربر را از محدودیتهای حافظه اصلی رها می کند. به کمک حافظه مجازی دیگر لزومی ندارد تمام صفحه‌ها (یا قطعه‌های) یک فرایند در حال اجرا، در حافظه اصلی قرار داشته

باشند و در ابتدا یک یا چند تکه حاوی آغاز برنامه، توسط سیستم عامل به حافظه آورده می‌شود. حافظه مجازی، حافظه منطقی کاربر را از حافظه فیزیکی تفکیک می‌کند. این تفکیک موجب می‌شود در حالتی که حافظه فیزیکی کم است، حافظه مجازی بزرگی برای برنامه نویس فراهم شود. حافظه مجازی، برنامه نویسی را ساده می‌کند، زیرا برنامه نویس نگران حافظه فیزیکی و کد جایگذاری نمی‌باشد.

اگر در حین اجرا به آدرسی رجوع شود که در مجموعه مقیم در حافظه نباشد، یک وقفه ایجاد می‌شود و این فرایند مسدود می‌شود و تکه مورد نظر وارد حافظه می‌شود، سپس کنترل دوباره به سیستم عامل بر می‌گردد و فرایند مسدود را به حالت آماده برمی‌گرداند.

تذکر: به بخشی از فرایند که واقعاً داخل حافظه اصلی قرار دارد، مجموعه مقیم می‌گویند.

✍ عناصر لازم برای موثر بودن حافظه مجازی عبارتند از:

- ۱- حمایت سخت افزاری برای به کارگیری صفحه بندی (یا قطعه بندی)
 - ۲- حمایت نرم افزاری برای انتقال صفحه‌ها (یا قطعه‌ها) بین حافظه ثانوی و حافظه اصلی.
- حافظه مجازی را می‌توان به روشهای زیر پیاده سازی کرد:

۱- صفحه بندی درخواستی (Demand Paging)

۲- قطعه بندی درخواستی (Demand Segmentation)

۳- قطعه بندی صفحه بندی شده (Hybrid Paging and Segmentation)

روش اول در این فصل و روش دوم و سوم در فصل بعد بررسی می‌شوند.

صفحه بندی درخواستی

در صفحه بندی برای حافظه مجازی (صفحه بندی درخواستی)، مانند صفحه بندی ساده، حافظه اصلی به تکه‌های هم اندازه به نام قاب (fram) و برنامه به صفحه‌ها (page) تقسیم می‌شود. اما برخلاف صفحه بندی ساده، لزومی ندارد تمام صفحه‌های یک فرایند در قابهای حافظه اصلی باشند تا فرایند اجرا شود. و تنها صفحاتی از فرایند که مورد نیاز است به حافظه اصلی آورده می‌شوند. این روش تلفیقی از صفحه بندی و مبادله است.

آدرس‌هایی که توسط برنامه‌ها تولید می‌شوند (آدرس مجازی)، به واحد MMU که در درون پردازنده قرار دارد، منتقل می‌شوند. توسط این واحد، عمل ترجمه (نگاشت) آدرس مجازی به آدرس فیزیکی انجام می‌شود. این عمل مشابه مراحل ترجمه آدرس در روش صفحه بندی است، با این تفاوت که بیت حضور (Present) برای هر درایه جدول صفحه وجود دارد که حضور یا عدم حضور صفحه در حافظه اصلی را تعیین می‌کند.

اگر MMU به جدول صفحه مراجعه کند و بیت حضور را برای صفحه مورد نظر صفر ببیند، متوجه می‌شود که صفحه بر روی دیسک است. در این حالت پردازنده در تله (trap) سیستم عامل می‌افتد. به این تله، خطای نقص صفحه (Page Fault) می‌گویند که مدیریت آن بر عهده سیستم عامل است.

وقتی سعی شود مکانی از حافظه مجازی خوانده شود که موجود نیست، فقدان صفحه رخ می دهد و منجر به صدور وقفه می گردد.

جدول صفحه

واحد مدیریت حافظه یا MMU به کمک جدول صفحه (Page Table)، آدرس مجازی را به آدرس فیزیکی، نگاشت می کند. در صفحه بندی مجازی، هر فرایند جدول صفحه خود را دارد. اندازه هر درایه (e) جدول صفحه معمولا چهار بایت است. جدول صفحه بر حسب اندیس شماره صفحه چیده شده است و P# یک فیلد از جدول محسوب نمی شود. فیلدهای هر درایه از یک جدول صفحه شامل موارد زیر است:

بیت اعتبار (Valid)	اگر بیت اعتبار برای صفحه ای 1 باشد، یعنی برنامه نویس از آدرس های درون این صفحه استفاده کرده است. اگر 0 باشد، یعنی صفحه بدون استفاده است.
بیت حضور - غیاب (P/A)	اگر این بیت 1 باشد، یعنی صفحه در حافظه قرار دارد و سایر فیلدهای این درایه قابل استفاده می باشند. به این فیلد in/out نیز می گویند.
شماره قاب صفحه (F#)	مهمترین فیلد است و هدف اصلی از نگاشت صفحه، یافتن شماره قاب صفحه است.
بیت مراجعه شده (R) (Referenced)	اگر یک مراجعه، خواندن یا نوشتن به یک صفحه انجام گیرد، این بیت برابر 1 می شود.
بیت تغییر یافته (M) (Modified)	اگر عمل نوشتن بر روی یک صفحه انجام شود، سخت افزار به صورت اتوماتیک این بیت را 1 می کند.
بیت‌های حفاظت (Protection)	این بیت ها نوع دسترسی مجاز را مشخص می کند. در حالت سه بیتی (rwx)، یک بیت برای خواندن، یک بیت برای نوشتن و یک بیت برای اجرا در نظر گرفته می شود.
سن (Age)	هر چه مقدار این فیلد برای صفحه ای کمتر باشد، نشان دهنده این است که آن صفحه مدت بیشتری در حافظه بدون مراجعه بوده است.
بیت کپی در نوشتن (Copy on Write)	این فیلد زمانی مقدار می گیرد که بیش از یک فرایند در یک صفحه شریک باشند. اگر یکی از فرایندها در صفحه بنویسد، یک کپی جداگانه برای تمام فرایندهایی که در این صفحه شریک هستند، درست می شود.
بیت کش غیر فعال شده (Caching Disable)	توسط این بیت، می توان امکان استفاده از حافظه نهان را برای صفحه، غیر فعال کرد. ماشین هایی که از فضای I/O جداگانه برخوردارند و از I/O نگاشت شده با حافظه استفاده نمی کنند، نیازی به این بیت ندارند.

مثال

یک سیستم کامپیوتری مبتنی بر حافظه مجازی با اندازه صفحه 32KB مفروض است. اگر حداکثر برنامه قابل اجرا در این سیستم 4MB باشد، تعداد مدخل‌های جدول صفحه کدام است؟

پاسخ:

$$\frac{4MB}{32KB} = \frac{4 \times 2^{20}}{32 \times 2^{10}} = 2^7 = 128$$



مثال

یک سیستم کامپیوتری مبتنی بر حافظه مجازی با اندازه صفحه 32KB مفروض است. در صورتی که حجم حافظه اصلی این سیستم 512KB باشد، تعداد صفحات حافظه اصلی چه مقدار خواهد بود؟

$$\frac{512KB}{32KB} = 16 \text{ پاسخ}$$



مثال

یک سیستم حافظه صفحه بندی مجازی را در نظر بگیرید که حداکثر اندازه جدول صفحه هر فرایند در آن برابر 8 مگابایت، اندازه هر صفحه برابر 8 کیلوبایت و مدخل‌های هر جدول صفحه 16 بیتی باشند، آدرسهای ماشینی که بتواند این حافظه مجازی را مستقیماً آدرس دهی کند، باید چند بیتی باشند؟

پاسخ:

$$e = \frac{8MB}{16} = \frac{8 \times 2^{20}}{16} = 2^{19}$$

$$2^{19} \times (8kb) = 2^{19} \times (8 \times 2^{10}) = 2^{32}$$

■ بنابراین آدرس باید 32 بیتی باشد.

جدول صفحه در حافظه اصلی قرار می‌گیرد. چون برای هر فرایند یک جدول وجود دارد و هر فرایند می‌تواند مقدار زیادی از حافظه مجازی را اشغال کند، بنابراین تعداد مدخلهای صفحه برای هر فرایند زیاد خواهد شد و جدول صفحه فضای زیادی را اشغال خواهد کرد. بنابراین بهتر است جداول صفحه در حافظه مجازی ذخیره شوند، یعنی خود جداول صفحه نیز در معرض صفحه بندی هستند.

روشهای کاهش اندازه جدول صفحه

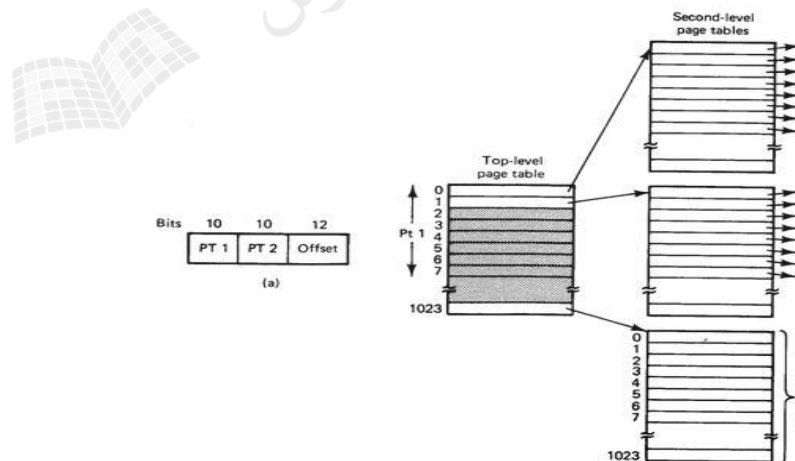
۱- جداول صفحه چند سطحی ۲- جدول صفحه وارونه (معکوس)

صفحه بندی چند سطحی

سیستم های مدرن از فضای آدرس منطقی گسترده ای پشتیبانی می کنند. در این محیط ها درایه های جدول صفحه زیاد خواهند شد و در نتیجه اندازه جدول صفحه بزرگ خواهد شد و چون نمی خواهیم جدول صفحه را به طور همجوار در حافظه اصلی تخصیص دهیم، باید آن را به فضاهای کوچکتری تقسیم کرد. برای این کار می توان از الگوی صفحه بندی دو سطحی استفاده کرد که خود جدول صفحه نیز صفحه بندی می شود. فرض کنید یک فرایند 20 کیلو بایتی در یک سیستم با آدرس های 32 بیتی و صفحات 4 کیلو بایتی (2^{12} بایت) داشته باشیم. بنابراین فرایند نیاز به 5 صفحه دارد. از طرفی چون آدرس 32 بیتی است و 12 بیت آن برای آفست است، پس 20 بیت برای شماره صفحه باقی می ماند. بنابراین جدول صفحه دارای 2^{20} درایه خواهد بود که فقط 5 درایه آن استفاده خواهد شد. پس تعداد بسیار زیادی از درایه های جدول صفحه استفاده نشده است. اگر درایه های استفاده نشده را حذف کنیم، دیگر نمی توان بر اساس P#، جدول صفحه را اندیس دهی کرد و ترجمه آدرس کند می شود. برای حذف اغلب (نه تمامی) درایه های خالی و حفظ مکانیسم نشانی دهی بر اساس P#، از جدول صفحه چند سطحی استفاده می کنیم. در این روش، اندیس دهی چند مرحله ای می شود.

مثال

سیستمی از یک جدول صفحه دو سطحی و آدرس های مجازی 32 بیتی استفاده می کند. اندازه صفحات 2^{12} بایت است. اولین 10 بیت آدرس به عنوان ایندکس به اولین جدول صفحه عمل می کند. در نتیجه جدول صفحه سطح اول دارای 2^{10} درایه است. بنابراین ایندکس سطح دوم 10 بیتی است. هر جدول صفحه در سطح دوم دارای 2^{10} درایه است.



مثال

کامپیوتری با یک فضای آدرس پذیر مجازی 18 بیتی را که صفحات آن هر یک 2^6 بایت ظرفیت دارند، در نظر بگیرید. اندازه هر مدخل جدول صفحه 4 بایت است. به دلیل آنکه هر جدول باید داخل یک صفحه جای گیرد، یک جدول صفحه چند سطحی استفاده شده است. چند سطح مورد نیاز است؟

پاسخ:

فضای آدرس مجازی 18 بیت است که 6 بیت آن مربوط به آفست است. بنابراین 12 بیت برای شماره صفحه جا داریم. حال باید ببینیم که این 12 بیت را به چند قسمت، تقسیم کنیم.

از آنجا که هر مدخل جدول صفحه چهار بایتی است، تعداد مدخل های هر جدول صفحه که می تواند در یک صفحه جای گیرد، برابر $2^4 = \frac{2^6}{4}$ است. بنابراین اندیس های PT_i برابر 4 بیت است. در نتیجه تعداد سطوح مورد

نیاز برابر $3 = \left\lceil \frac{12}{4} \right\rceil$ می باشد. آدرس منطقی :

PT1= 4bit	PT2= 4 bit	PT3= 4 bit	OFFSET= 6 bit
-----------	------------	------------	---------------

مثال

فرض کنید سیستمی از فضای آدرس دهی مجازی 2^{10} بایت پشتیبانی می کند. در این سیستم اندازه حافظه فیزیکی قابل دسترسی 2^9 بایت و طول هر قاب حافظه در این سیستم 2^4 بایت می باشد. این سیستم از روش صفحه بندی برای مدیریت حافظه استفاده کرده است. با فرض اینکه هر مدخل از جدول صفحه به 11 bit به عنوان بیت‌های کنترلی (بیت حضور- غیاب و ...) نیاز داشته باشد، در این صورت برای اینکه هر جدول صفحه جزئی، دقیقاً در یک قاب قرار گیرد، باید حداقل از جدول صفحه چند سطحی استفاده شود؟

پاسخ: طبق اطلاعات مسئله، مشخص است که فضای آدرس مجازی 10 بیت است که 4 بیت آن مربوط به آفست است. بنابراین 6 بیت برای شماره صفحه جا داریم. حال باید ببینیم که این 6 بیت را باید به چند قسمت تقسیم کنیم. تعداد قاب ها

از تقسیم اندازه فضای فیزیکی به اندازه هر قاب بدست می آید: $\frac{2^9}{2^4} = 2^5$

پس چون 32 قاب صفحه داریم، 5 بیت برای شماره قاب در جدول صفحه در نظر گرفته می شود.

همچنین 11 بیت نیز برای بیت‌های کنترلی در نظر گرفته می شود. بنابراین اندازه هر درایه جدول صفحه، برابر 16 بیت (2

بایت) است. چون هر درایه جدول صفحه 2 بایت است، پس تعداد 8 درایه $2^3 = \frac{2^4}{2}$ می تواند در یک قاب جای بگیرد.

بنابراین اندیس های PT_i برابر 3 بیت است. و در نتیجه باید از جدول صفحه 2 ($\frac{6}{3} = 2$) سطحی استفاده کرد. آدرس منطقی

به صورت زیر است:

PT1 =3 bit	PT2 =3 bit	Offset =4 bit
------------	------------	---------------

جدول صفحه وارونه (معکوس)

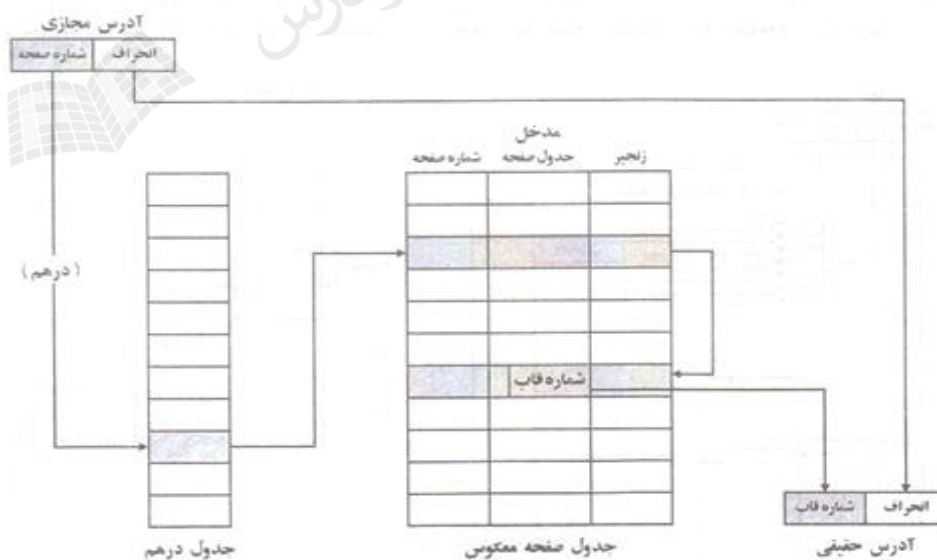
یکی از روش‌های کاهش اندازه جدول صفحه، استفاده از جدول صفحه معکوس است. جدول صفحه معکوس بر اساس شماره قاب (F#) اندیس شده است (نه شماره صفحه). هر یک از درایه‌ها جدول صفحه معکوس، مشخص می‌کند صفحه مجازی شماره چند از کدام فرایند در آن قاب صفحه قرار گرفته است. در این روش، به جای اینکه به ازای هر فرایند، یک جدول صفحه مجزا داشته باشیم، فقط یک جدول صفحه عمومی داریم.

اگر چه جدول صفحه معکوس جای کمی را اشغال می‌کند، ولی تبدیل آدرس مجازی به فیزیکی بسیار مشکل‌تر می‌باشد. برای سریع کردن جستجو، از یک جدول درهم (Hash Table) استفاده می‌شود. قسمت شماره صفحه از آدرس مجازی به کمک یک تابع درهم ساز به یک جدول درهم نگاشت می‌شود. جدول درهم دارای نشانگری به جدول صفحه معکوس است که شامل مدخلهای جدول صفحه می‌باشد. با این ساختار در جدول درهم و در جدول صفحه معکوس، برای هر صفحه حافظه حقیقی تنها یک مدخل وجود دارد. (به جای یک مدخل برای هر صفحه مجازی).

پس جدول صفحه معکوس:

- ۱- سبب کاهش اندازه حافظه فیزیکی جهت ذخیره سازی آن می‌شود.
- ۲- باید یک جدول صفحه خارجی نیز برای آن ذخیره شود.
- ۳- زمان سرویس نقص صفحه افزایش می‌یابد. (به دلیل ایجاد یک نقص صفحه دیگر)

شکل زیر ساختار جدول صفحه معکوس نشان داده شده است:



بافرهای کناری ترجمه (TLB)

روشهای پیاده سازی سخت افزاری جدول صفحه عبارتند از:

- ۱- جدول صفحه به صورت مجموعه ای از ثباتها پیاده سازی می شود.
- ۲- جدول صفحه در حافظه اصلی ذخیره شود و ثبات پایه جدول صفحه (PTBR) به آن اشاره کند.
- ۳- استفاده از TLB (یک سخت افزار جستجوی سریع، کوچک و پنهان)

TLB : Translation Lookaside Buffers

تذکر: ثباتهای انجمنی TLB را میانگیرهای دم دستی نیز می نامند.

✍ اگر جدول صفحه بزرگ باشد، روش ۱ مناسب نمی باشد.

✍ در روش ۲، برای دسترسی به هر بایت، نیاز به دو دستیابی داریم (یکبار برای ورودی جدول صفحه و دیگری برای دسترسی به بایت) که این باعث کند شدن دستیابی به حافظه می شود.

نحوه بکارگیری TLB

در صفحه بندی تعداد اندکی از ورودیهای جدول صفحه در TLB قرار می گیرد. وقتی آدرسی تولید می شود، شماره صفحه آن با شماره صفحه موجود در ثباتهای انجمنی (که حاوی شماره صفحه و شماره قابهای متناظر با آنهاست)، مقایسه می گردد. اگر شماره صفحه در ثباتهای انجمنی پیدا شود، از شماره قاب آن برای دستیابی به حافظه استفاده می شود. اگر شماره صفحه در ثباتهای انجمنی موجود نباشد، ارجاع به جدول صفحه باید انجام گرفته و شماره صفحه و شماره قاب به TLB اضافه می شود تا در مراجعات بعدی به سرعت پیدا شود. اگر TLB پر باشد، سیستم عامل باید یکی از ورودی ها را حذف کرده و ورودی جدید را به جای آن قرار می دهد.

✍ هر بار که تعویض بستر (انتخاب جدول صفحه جدید) رخ دهد، باید TLB پاک شود تا فرایند بعدی که باید اجرا شود، از اطلاعات ترجمه ای نادرست استفاده نکند.

✍ درصد تعداد دفعاتی که شماره صفحه در ثباتهای انجمنی پیدا شود، نسبت اصابت (hit ratio) نام دارد. مثلا اگر نسبت اصابت 80% باشد، یعنی که 80% از تعداد دفعاتی که به ثباتهای انجمنی مراجعه کردیم، شماره صفحه پیدا شده است. نسبت اصابت را با H نشان می دهیم.

زمان موثر دسترسی

زمان موثر دسترسی، میانگین زمان واقعی است که برای یک دسترسی به حافظه اصلی مورد نیاز است. این زمان از زمان ترجمه آدرس و زمان دسترسی به کلمه مورد نظر در حافظه اصلی پس از محاسبه آدرس فیزیکی می باشد.

زمان موثر دسترسی با فرض استفاده از TLB و جدول صفحه ساده:

$$T_{Access} = T_{Translation} + T_{Mem}$$

$$T_{Translation} = H \times T_{TLB} + (1 - H) \times (T_{TLB} + T_{Mem}) = T_{TLB} + (1 - H) \times T_{Mem}$$

(T_{TLB} : زمان دستیابی TLB، H : نسبت اصابت TLB)

با فرض اینکه آدرس در TLB نباشد (یعنی $H=0$) رابطه به صورت زیر خواهد بود:

$$T_{Access} = T_{TLB} + 2T_{Mem}$$

زمان موثر دسترسی در صورت رخ دادن نقص صفحه با احتمال P :

$$T_{Access} = T_{Translation} + T_{Mem} + P \times T_{Disk}$$

$$T_{Translation} = T_{TLB} + (1 - H) \times (T_{Mem} + P \times T_{Disk})$$

(T_{Disk} : زمان انتقال صفحه از دیسک، P : احتمال خطای صفحه)

با فرض اینکه آدرس در TLB نباشد و نقص صفحه حتما رخ دهد (یعنی $H=0$ و $P=1$) رابطه به صورت زیر خواهد بود:

$$T_{Access} = T_{TLB} + 2T_{Mem} + 2T_{Disk}$$

در این فرمول فرض شده که صفحه به علت تغییر باید در دیسک ذخیره شود. اگر نیازی به ذخیره نباشد از به جای 2 قبل از T_{Disk} باید 1 قرار داد.

زمان موثر دسترسی با فرض استفاده از TLB و جدول صفحه دو سطحی

فرمت آدرس منطقی پردازنده در این حالت به صورت زیر است:

Dir	Page	Offset
-----	------	--------

$$T = T_{TLB} + (1 - H_{TLB})(T_1 + T_2) + T_3 + P \times T_{DISK}$$

$$T_1 = T_{Cache-Dir} + (1 - H_{Cache}) \times T_{cache-Dir_Miss}$$

$$T_2 = T_{Cache-Pagetable} + (1 - H_{Cache}) \times T_{Cache-Pagetable-Miss}$$

$$T_3 = T_{Cache-Frame} + (1 - H_{Cache}) \times T_{Cache-Frame-Miss}$$

بررسی حالت های مختلف:

۱- اگر در مراجعه به داده اصلی در قاب صفحه دچار فقدان صفحه شویم، دو حالت رخ می دهد.

الف- قاب خالی وجود دارد و نیاز به جایگزینی صفحه نیست: $T \cong T_{disk}$

ب- قاب خالی وجود ندارد و نیاز به جایگزینی صفحه می باشد: $T \cong 2 \times T_{Disk}$

۲- اگر TLB اصابت کند. برای مراجعه به خود داده در حافظه دو حالت رخ می دهد:

الف- cache اصابت کند: $T = T_{Translation} + T_{cache}$

ب- cache اصابت نکند: $T = T_{translation} + T_{cache} + T_{cache-miss-penalty}$

۳- اگر TLB اصابت نکند، با توجه به سه مورد "دایرکتوری، جدول صفحه و قاب" در این صورت چهار حالت رخ

می دهد:

الف- در مراجعه به هر سه مورد، اصابت cache داشته باشیم:

$$T = T_{TLB} + T_{Cache-Dir} + T_{Cache-pagetable} + T_{cache-Frame}$$

ب- در مراجعه به یکی از سه مورد، cache اصابت نکند:

$$T = T_{old} + 1 \times T_{penalty}$$

(منظور از T_{old} ، همان T بدست آمده از حالت الف این حالت است)

ج- در مراجعه به دو تا از سه مورد، cache اصابت نکند: $T = T_{old} + 2 \times T_{penalty}$

د- در مراجعه هر سه مورد، cache اصابت نکند: $T = T_{old} + 3 \times T_{penalty}$

الگوریتم‌های جایگزینی صفحه (Page Replacement Algorithms)

در یک سیستم چند برنامه ای که از بخش بندی پویا استفاده می‌کند زمانی خواهد رسید که تمام فرایندهای موجود در حافظه اصلی در حالت مسدود قرار دارند. در این حالت سیستم عامل یکی از فرایندهای داخل حافظه اصلی را به خارج مبادله می‌کند تا فضای کافی برای فرایندی جدید یا یک فرایند آماده و معلق ایجاد شود. در این حالت سیستم عامل باید فرایندی که قرار است جایگزین شود را انتخاب کند. در واقع سیاست جایگزینی درباره انتخاب یک صفحه برای جایگزینی در زمانی که لازم است یک صفحه جدید به داخل آورده شود صحبت می‌کند.

الگوریتم‌های جایگزینی عبارتند از:

- ۱- بهینه (optimal)
- ۲- عدم استفاده در گذشته اخیر (NRU)
- ۳- خروج به ترتیب ورود (FIFO)
- ۴- دومین شانس (Second Change)
- ۵- ساعت (Clock)
- ۶- کمترین استفاده در گذشته نزدیک (LRU)

الگوریتم بهینه (optimal)

در این روش، صفحه ای جایگزین شود که به مدت طولانی مورد استفاده قرار نخواهد گرفت.

الگوریتم بهینه، قابل اجرا نیست، چون نیاز به پیش بینی آینده دارد. علت مطرح شدن این الگوریتم، استفاده از آن به عنوان شاخص مقایسه می باشد.

نام دیگر این الگوریتم، BO (Belady Optimal) می باشد.

مثال

تعداد فقدان صفحه برای مراجعات زیر با داشتن 3 قاب صفحه در صورت استفاده از روش بهینه را مشخص کنید.

7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

پاسخ:

در ابتدا به صفحه 7 مراجعه شده و چون در قابها موجود نیست، فقدان صفحه رخ می دهد و این صفحه به داخل آورده می شود. مراجعه به صفحه 0 و 1 نیز منجر به فقدان صفحه می شود.

حال به صفحه 2 مراجعه می شود. چون هر 3 قاب پر شده است، باید یکی از صفحه های 0, 7 و 1، را از قابها خارج کرد، که همان صفحه 7 است، چون در آینده دورتری به آن مراجعه شده است. بعد به صفحه 0 مراجعه می شود که در قاب موجود است و بعد به 3 مراجعه می شود که جای صفحه 1 قرار خواهد گرفت، چون صفحه 1 بین صفحه های موجود در قاب در آن لحظه (یعنی 2,0,1) در آینده دورتری به آن مراجعه شده است.

این روال را ادامه می دهیم. در نهایت 9 خطای صفحه رخ می دهد.

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
7	7	7	2	2	2	2	2	2	2	2	2	2	2	2	2	2	7	7	7
	0	0	0	0	0	0	4	4	4	0	0	0	0	0	0	0	0	0	0
		1	1	1	3	3	3	3	3	3	3	3	1	1	1	1	1	1	1
F	F	F	F		F		F		F		F		F		F		F		F

الگوریتم NRU

در روش NRU (Not Recently Used)، وقتی نقص صفحه رخ می دهد، سیستم عامل بر اساس وضعیت بیت های ارجاع (R) و اصلاح (M)، صفحات را به چهار کلاس مطابق شکل زیر، تقسیم می کند و سپس به طور تصادفی، صفحه ای را از کلاسی که در دسته با شماره کمتری باشد، را انتخاب می کند.

شماره کلاس	R	M
0	0	0
1	0	1
2	1	0
3	1	1

در واقع صفحه ای انتخاب می شود که در درجه اول از ابتدای دوره جاری، استفاده نشده است و در درجه دوم، از هنگام ورود به حافظه تغییر نکرده است. صفحه متعلق به دسته شماره 0، اخیراً مورد استفاده قرار نگرفته و تغییر نکرده است. به همین علت بهترین انتخاب می باشد.

تذکره: صفحه ای که بیت اصلاح آن یک باشد، قبل از جایگزینی باید نوشته شود.

کلاس 1، زمانی رخ می دهد که در کلاس 3، بیت R در یک وقفه ساعت، 0 شود. (وقفه ساعت بیت M را 0 نمی کند، چون این بیت نشان دهنده این است که آیا این صفحه باید دوباره در دیسک نوشته شود یا خیر.)

الگوریتم FIFO

در این روش (خروج به ترتیب ورود)، صفحه ای جایگزین شود که زمان بیشتری منتظر بوده است. برای جایگزینی، صفحه موجود در ابتدای صف را انتخاب کرده و صفحه ای که وارد حافظه شده را به انتهای صف اضافه می کنیم.

مثال

تعداد فقدان صفحه برای مراجعات زیر با داشتن 3 قاب صفحه در صورت استفاده از روش FIFO، چند است؟
7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

پاسخ: در FIFO از یک صف استفاده می شود که طبق قانون صف، حذف از ابتدا و درج به انتها صورت می گیرد. سه مراجعه اول قابها را پر کرده و 3 نقص صفحه رخ می دهد. مراجعه به صفحه 2، موجب حذف صفحه 7 (اول صف) شده و صفحه 2 به انتهای صف اضافه می شود. مراجعه به صفحه 0 تغییری را ایجاد نمی کند چون در قاب موجود است. مراجعه به صفحه 3 موجب حذف 0 از ابتدای صف و درج 3 به انتهای صف و بروز نقص صفحه خواهد شد. در نهایت 15 خطای صفحه داریم.

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
7	7	7	0	0	1	2	3	0	4	2	2	2	3	0	0	0	1	2	7
	0	0	1	1	2	3	0	4	2	3	3	3	0	1	1	1	2	7	0
		1	2	2	3	0	4	2	3	0	0	0	1	2	2	2	7	0	1
F	F	F	F		F	F	F	F	F	F			F	F		F	F	F	

الگوریتم دومین شانس

الگوریتم دومین شانس، مانند FIFO می باشد. با این تفاوت که وقتی نقص صفحه رخ می دهد، اگر بیت R قدیمی ترین صفحه، 1 بود، این بیت 0 شده و صفحه به انتهای لیست منتقل می شود. با این کار به او شانس دوباره ای داده شده است.

اگر بیت R همه صفحات 1 باشد، الگوریتم دومین شانس به FIFO تبدیل می شود.

مثال

سیستمی از الگوریتم دومین شانس استفاده می نماید. در صورتی که به شماره صفحه مجازی 2 ارجاع شود، به جای کدام صفحه قرار می گیرد؟



پاسخ:

بیت R صفحه های 4 و 5 چون 1 است، آن را صفر کرده و صفحه ها به انتهای صف منتقل می شوند. حال به صفحه 7 رسیده ایم که چون بیت R آن صفر است، صفحه را از حافظه خارج کرده و صفحه 2 را به جای آن در انتهای صف اضافه می کنیم. (بیت R صفحه دو را 1 قرار می دهیم).



الگوریتم ساعت

اگر چه الگوریتم دومین شانس، الگوریتم قابل قبولی بود، اما به علت حذف صفحه از ابتدای لیست و اضافه کردن آن به انتهای لیست، روشی پر هزینه است. برای حل این مشکل از لیست چرخشی استفاده می شود. در الگوریتم ساعت، لیست پیوندی صفحات به صورت حلقوی و به شکل یک ساعت نگهداری می شود، به طوری که عقربه ساعت به قدیمی ترین صفحه اشاره می کند. وقتی نقص صفحه رخ می دهد، به بیت R صفحه ای که توسط عقربه به آن اشاره شده، نگاه کرده که دو حالت رخ می دهد:

الف- اگر بیت R ، 0 باشد، صفحه مورد نظر خارج شده و صفحه جدید در همان مکان جایگزین شده و عقربه به جلو می رود.

ب- اگر بیت R ، 1 باشد، آنگاه 0 شده و عقربه به صفحه بعدی اشاره خواهد کرد.

مثال

سیستمی دارای 4 قاب صفحه است که مطابق جدول زیر صفحات مجازی در آن‌ها قرار دارند. اگر سیستم عامل از الگوریتم ساعت (Clock) استفاده نماید، در صورت وقوع نقص صفحه، صفحه جدید با کدام صفحه مجازی باید جایگزین شود؟

شماره صفحه	زمان بارگذاری صفحه در حافظه	R
1	50	1
2	30	0
3	20	1
4	40	0

پاسخ:

ابتدا عقربه بر روی قدیمی‌ترین صفحه یعنی صفحه 3 است. چون بیت R آن 1 است، بیت R آن 0 شده و عقربه به جلو حرکت می‌کند و بر روی صفحه 2 می‌رود (صفحه بعدی بر اساس زمان بارگذاری). چون بیت R صفحه 2، صفر است، صفحه جدید با این صفحه جایگزین می‌شود. ■

الگوریتمی به نام WS-Clock هست که در آن وقتی عقربه به صفحه ای می‌رسد، که عضوی از مجموعه کاری (Working Set) فرایند باشد، انتخاب نمی‌شود و عقربه به جلو می‌رود. برای پیاده‌سازی این روش از فیلد سن (Age) جدول صفحه استفاده می‌شود. (مجموعه صفحاتی که فرایند در حال حاضر از آنها استفاده می‌کند، مجموعه کاری نام دارد.)

الگوریتم LRU

در روش (LRU (Least Recently Used)، صفحه ای جایگزین می‌شود که برای مدت طولانی مورد استفاده قرار نگرفته است. یعنی در گذشته دورتری به آن مراجعه شده است.

مثال

تعداد فقدان صفحه برای مراجعات زیر با داشتن 3 قاب صفحه در صورت استفاده از روش LRU چند است؟

7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

پاسخ:

در ابتدا به صفحه 7 مراجعه شده و چون در قابها موجود نیست، فقدان صفحه رخ می دهد و این صفحه به داخل آورده می شود. مراجعه به صفحه 0 و 1 نیز منجر به فقدان صفحه می شود.

حال به صفحه 2 مراجعه می شود. چون هر 3 قاب پر شده است، باید یکی از صفحه های 7، 0 و 1، را از قابها خارج کرد، که همان صفحه 7 است، چون نسبت به صفحه های 0 و 1 در گذشته دورتری به آن مراجعه شده است. بعد به صفحه 0 مراجعه می شود که در قاب موجود است و بعد به 3 مراجعه می شود که جای صفحه 1 قرار خواهد گرفت، چون صفحه 1 بین صفحه های موجود در قاب در آن لحظه (یعنی 2,0,1) در گذشته دورتری به آن مراجعه شده است.

بنابراین الگوریتم LRU دارای 12 خطای صفحه است.

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
7	7	7	2	2	2	2	4	4	4	0	0	0	1	1	1	1	1	1	1
	0	0	0	0	0	0	0	0	3	3	3	3	3	3	0	0	0	0	0
		1	1	1	3	3	3	2	2	2	2	2	2	2	2	2	7	7	7
F	F	F	F		F		F	F	F	F		F		F		F			

پیاده سازی سخت افزاری LRU

پیاده سازی الگوریتم LRU، پر هزینه است. یکی از طرح های سخت افزاری برای پیاده سازی آن، استفاده از ماتریس $n \times n$ است. در این طرح، برای یک ماشین با n قاب صفحه، سخت افزار LRU، یک ماتریس $n \times n$ بیتی است، که در ابتدا همه عناصر آن صفر است. اگر به قاب صفحه شماره k مراجعه شود، همه بیت های سطر k ام را یک می کند و سپس همه بیت های ستون k ام، را صفر می کند. در هر لحظه، سطری که مقدار دودویی آن از همه سطرها کمتر باشد، همان صفحه مورد نظر است و نسبت به صفحه های دیگر، مدت طولانی تری بدون استفاده مانده است.

مثال

عملکرد این الگوریتم برای 4 قاب صفحه و دسترسی به صفحات 0 1 2 3 2 1 0 3 2 3 (از چپ به است) در شکل زیر نشان داده شده است.

چون 4 قاب داریم، پس یک ماتریس 4×4 بیتی با عناصر صفر در نظر می‌گیریم. اولین درخواست به صفحه 0 است. بنابراین ابتدا همه بیت های سطر شماره صفر را 1 می‌کنیم. سپس همه بیت های ستون شماره صفر را 0 می‌کنیم (شکل a). درخواست بعدی شماره 1 است. در ماتریس شکل a، ابتدا سطر یک را 1 کرده و سپس ستون یک را 0 می‌کنیم (شکل b).

	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3	
0	0	1	1	1	0	0	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	0	1	1	1	0	0	1	1	0	0	0	1	0	0	0	0
2	0	0	0	0	0	0	0	0	0	1	1	0	1	1	1	0	0	1	1	0	1
3	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	1	1	0	0
	(a)	(b)	(c)	(d)	(e)	(f)	(g)	(h)	(i)	(j)											

مثال

یک سیستم با 256 مگا بایت حافظه اصلی را در نظر بگیرید که از تکنیک صفحه بندی با اندازه قاب 4 کیلو بایت استفاده می‌کند. اگر بخواهیم الگوریتم کمترین استفاده در گذشته اخیر (LRU) را با روش ماتریس دو بعدی تقریب بزنییم، چقدر حافظه برای ذخیره سازی ماتریس نیاز است؟

پاسخ:

از آنجا که در این طرح، برای یک ماشین با n قاب صفحه، یک ماتریس $n \times n$ بیتی در نظر گرفته می‌شود، پس ابتدا تعداد قاب ها را محاسبه می‌کنیم:

$$\frac{256MB}{4KB} = \frac{2^8 \times 2^{20}}{2^2 \times 2^{10}} = \frac{2^{28}}{2^{12}} = 2^{16}$$

بنابراین ماتریس مورد نیاز $2^{16} \times 2^{16}$ بیتی است.

نحوه تبدیل $2^{16} \times 2^{16}$ بیت به بایت:

$$\frac{2^{16} \times 2^{16}}{8} = 2^{29} = 2^9 \times 2^{20} = 512MB$$

بنابراین به 512 مگا بایت (دو برابر اندازه حافظه اصلی)، حافظه نیاز داریم.



شبه سازی LRU در نرم افزار (الگوریتم سالمندی)

در الگوریتم سالمندی (Aging)، از فیلد سن (age) در جدول صفحه استفاده می شود که مثلاً می تواند 8 بیتی باشد. مقدار این فیلد در هنگام بارگذاری صفحه، 0 است. سیستم عامل در هر وقفه ساعت، فیلد سن همه صفحه های موجود در حافظه را یک بیت به سمت راست شیفت داده و بیت R را در بیت انتهایی سمت چپ (بیت با ارزش)، قرار می دهد. سپس فیلد R همه صفحات را 0 می کند. اگر نقص صفحه رخ دهد، صفحه ای با کمترین مقدار شمارنده برای جایگزینی انتخاب می شود.

به عنوان مثال اگر فیلد سن یک صفحه برابر 00001101 و بیت R آن برابر 1 باشد، پس از پایان پریرود زمانی، ابتدا این فیلد به اندازه یک بیت به سمت راست شیفت داده شده (0000110) و سپس بیت R در بیت انتهایی سمت چپش قرار می گیرد. پس فیلد سن برابر 10000110 خواهد شد.

به کمک فیلد سن می توان به تاریخچه مراجعات به صفحه در چند دوره اخیر پی برد. به عنوان مثال اگر فیلد سن یک صفحه 11000111 باشد، متوجه می شویم که به این صفحه در دو دوره اخیر مراجعه شده، در سه دوره قبل از آن، مراجعه ای به آن انجام نشده و قبل از این پنج دوره، در سه دوره متوالی نیز به این صفحه مراجعه شده است.

مثال

اگر فیلد سن صفحه 1 برابر 00010010 و فیلد سن صفحه 2 برابر 00010000 باشد، کدام صفحه برای جایگزینی انتخاب می شود؟

پاسخ: به هر یک از دو صفحه در سه دوره متوالی اخیر، مراجعه ای نشده است. ولی به هر دو در دوره قبل از آن مراجعه شده است. اما نمی توان تشخیص داد که به کدام صفحه در آن دوره، دیرتر مراجعه شده است. در اینصورت چون به صفحه شماره 1 در سه دوره قبل از آن مراجعه شده و در مورد صفحه 2 چنین نمی باشد، صفحه 1 برای جایگزینی انتخاب می شود. ■

یکی از تفاوت های روش سالمندی با LRU این است، که نمی توان تقدم و تاخر دسترسی ها در داخل یک دوره را تشخیص داد.

چون تعداد بیت های شمارنده در روش سالمندی محدود است، (به طور مثال 8 بیت)، اگر مقدار شمارنده برای دو صفحه 00000000 باشد، اگر به یکی از آنها در 9 دوره قبل مراجعه شده باشد و به دیگری در 100 دوره قبل، نمی توان از این موضوع آگاه شد. در اینصورت یک صفحه به صورت تصادفی انتخاب می شود.

الگوریتم‌های جایگزینی LFU و MFU نیز وجود دارد که به علت پایین بودن کارایی، در بیشتر منابع بیان نشده است. در صورت وقوع نقص صفحه، در LFU، صفحه با کمترین مقدار شمارنده و در MFU، صفحه با بیشترین مقدار شمارنده، جهت جایگزینی انتخاب می‌شود.

الگوریتم بافر کردن صفحه

در این روش، تعدادی قاب صفحه به عنوان بافر رزرو شده و صفحات درون آن از نظر جدول صفحه، غایب در نظر گرفته می‌شوند و با مراجعه به آنها نقص صفحه رخ می‌دهد. در این روش سه لیست صفحه داریم:

۱- درون حافظه ۲- درون بافر(بدون تغییر) ۳- درون بافر(تغییر یافته)

این لیست‌ها، به ترتیب ورود مرتب می‌باشند. هنگامی که صفحه‌ای با الگوریتم FIFO، از ابتدای لیست اول برای جایگزینی انتخاب می‌شود، از لیست اول خارج شده و به انتهای لیست دوم یا سوم اضافه می‌شود. این صفحه واقعا از حافظه خارج نمی‌شود(در جدول صفحه به عنوان غایب علامت گذاری می‌شود) و به جای آن یک صفحه از ابتدای لیست دوم و یا سوم خارج شده و واقعا از حافظه حذف می‌شود.

یکی از نقطه ضعف‌های الگوریتم FIFO، اخراج صفحات قدیمی پر استفاده است که در الگوریتم بافر کردن صفحه برطرف شده است. چون اگر صفحه‌ای با استفاده زیاد، به طور ظاهری اخراج شود، به زودی با نقص صفحه مواجه شده و به سرعت این صفحه از بافر بر گردانده می‌شود.

نکات طراحی سیستم‌های صفحه بندی

تعداد کل قاب صفحاتی که می‌توان در یک سیستم به مجموعه فرایندها داد، مقداری ثابت و متناسب با اندازه حافظه اصلی است. کاهش تعداد قاب‌ها باعث می‌شود که یک فرایند نتواند مجموعه صفحاتی که در طی یک زمان از آنها استفاده می‌کند را در حافظه اصلی بارگذاری کند. در نتیجه تعداد نقص صفحه‌ها زیاد می‌شود. در این وضعیت، زمان CPU به جای اجرای فرایندها، صرف مبادله صفحه‌ها می‌شود و کارایی سیستم کاهش می‌یابد. به این پدیده، کوپیدگی (Thrashing) می‌گویند.

از مشکل‌های دیگر، می‌توان به این موضوع اشاره کرد که اگر در ابتدای بارگذاری یک فرایند، نتوان مجموعه کاری را تشخیص داد، باید آنقدر نقص صفحه به وجود آورد تا مجموعه کاری اش در حافظه لود شود. که باعث کند شدن سیستم می‌شود. بنابراین به دنبال راه‌حلی هستیم تا از این نوع مشکلات به وجود نیاید.

پیش صفحه بندی (prepaging)

سیاست واکنشی در مورد تعیین زمانی که یک صفحه باید به داخل حافظه آورده شود، دو رویکرد دارد:

۱- صفحه بندی درخواستی

۲- پیش صفحه بندی

در صفحه بندی درخواستی، فقط زمانی که مراجعه ای به مکانی از یک صفحه شود، آن صفحه به حافظه اصلی آورده می شود. ولی در پیش صفحه بندی، صفحه هایی به غیر از آنچه به وسیله خطای صفحه درخواست شده نیز به داخل آورده می شوند. یکی از خصوصیات صفحه بندی درخواستی، رخ دادن تعداد زیادی خطای صفحه در شروع یک کار می باشد، که پیش صفحه بندی سعی به جلوگیری از این صفحه بندی زیاد دارد و از ابتدا تمام صفحات مورد نیاز فرایند را به صورت یکجا، به حافظه می آورد.

سیاست پیش صفحه بندی می تواند یا در زمان شروع فرایند به کار گرفته شود یا هر بار که یک خطای صفحه رخ می دهد.

مدل مجموعه کاری (working sets)

در هر لحظه زمانی (t) ، مجموعه ای از صفحات وجود دارد که در k مراجعه اخیر به حافظه، مورد استفاده واقع شده اند. به این مجموعه، مجموعه کاری می گویند و به صورت $w(k,t)$ نمایش داده می شود. سیستم عامل با نظارت به مجموعه کاری هر فرایند، به آن قاب کافی اختصاص می دهد. اگر مجموع اندازه های مجموعه کاری فرایندها، زیادتر از تعداد کل قاب ها شود، پردازش معلق شده و از کوبیدگی پیشگیری می شود.

مثال

در صورت وجود ارجاعات یک برنامه به صفحات حافظه به ترتیب زیر (از چپ به راست)

4, 2, 0, 2, 1, 5, 1, 2, 3, 2, 1, 2, 6, 2, 1, 3

مجموعه کاری $w(t,k)$ که در آن t برابر زمان بین ارجاع هشتم و نهم و k برابر چهار باشد، کدام است؟

پاسخ: از ارجاع بین دهم و یازدهم به اندازه ۴ ارجاع به عقب بر می گردیم:

4, 2, 0, 2, 1, 5, 1, 2, 3, 2, 1, 2, 6, 2, 1, 3

بنابراین داریم: $w(t,k) = \{1,2,5\}$

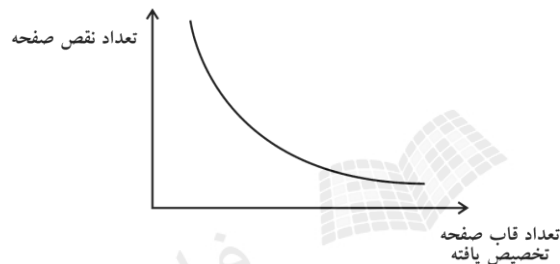


الگوریتم فرکانس نقص صفحه (PFF)

در روش PFF (Page Fault Frequency)، کران بالا و پایین برای نرخ خطای صفحه تعیین می‌شود. اگر نرخ خطای صفحه از کران بالا، بیشتر شود، قاب دیگری به آن فرایند تخصیص داده می‌شود و اگر نرخ خطای صفحه از کران پایین کمتر شود، قابی از فرایند پس گرفته می‌شود. بنابراین PFF سعی می‌کند که نرخ صفحه بندی را بین دو حد قابل قبول نگه دارد تا از تفریط (کوبیدگی) و افراط (اتلاف حافظه) جلوگیری شود.

تناقض بلیدی (Belady's anomaly)

در روش FIFO ممکن است با افزایش تعداد قابها، خطای صفحه کم نشود که به این پدیده تناقض بلیدی می‌گویند. تناقض بلیدی در الگوریتم های BO,LRU رخ نمی‌دهد. در واقع تناقض بلیدی به این معنی است که الگوریتم FIFO ممکن است از نمودار زیر، پیروی نکند:



مثال

دستیابی به صفحات { 4, 3, 2, 1, 4, 3, 5, 4, 3, 2, 1, 5 } را در نظر بگیرید. اگر اندازه حافظه سه صفحه باشد، تعداد 9 نقص صفحه رخ می‌دهد و اگر اندازه حافظه را به چهار صفحه افزایش دهیم، تعداد ده نقص صفحه رخ می‌دهد. به این موضوع تناقض FIFO می‌گویند. چون انتظار داشتیم با افزایش تعداد قابهای صفحه، تعداد فقدان صفحه ها (page fault) کمتر شود ولی این چنین نشد.

پاسخ: با انباره 3 صفحه ای، 9 فقدان صفحه رخ می‌دهد:

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

و با در نظر گرفتن انباره 4 صفحه ای، 10 فقدان صفحه رخ می دهد:

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

الگوریتم های پشته (Stack Algorithms)

الگوریتم پشته ای الگوریتمی است که در آن، مجموعه ای از صفحات موجود در حافظه برای n قاب، همیشه زیر مجموعه ای از صفحاتی است که برای n+1 قاب در حافظه خواهند بود. الگوریتم های پشته ای هیچگاه دچار تناقض بلیدی نمی شوند.

اندازه صفحه

اندازه صفحات توانی از 2 هستند. برای تعیین اندازه صفحه باید به موارد زیر توجه شود:

۱- اندازه جدول صفحه

استفاده از صفحه های کوچکتر موجب افزایش اندازه جدول صفحه می شود. چون تعداد صفحات زیادتر می شود.

۲- بهره وری

هرچه صفحه کوچکتر باشد، بهره وری بیشتر است، چون تکه تکه شدن داخلی کمتر می شود.

۳- زمان خواندن یا نوشتن یک صفحه

استفاده از صفحه های کوچکتر، موجب کاهش کل زمان I/O می شود، چون محلی بودن بهبود می یابد.

کاهش اندازه صفحه موجب:

الف- کاهش تکه تکه شدن داخلی حافظه می شود.

ب- افزایش بهره وری حافظه و افزایش زمان I/O می شود.

ج- کاهش زمان سرویس نقص صفحه می شود.

تذکر: برای کمینه کردن تعداد خطای صفحه باید از صفحات بزرگ استفاده شود.

اندازه صفحه بهینه برابر $P = \sqrt{2Se}$ می باشد. (S: اندازه فرایند) (e: اندازه یک مدخل جدول صفحه)

مثال

در محیط یک سیستم عامل که از روش حافظه مجازی برای مدیریت حافظه استفاده می کند، چنانچه اندازه هرفرایند 64 کیلوبایت و برای هر پروسس در جدول صفحه، 8 بایت اطلاعات ذخیره شود، اندازه صفحه بهینه در این سیستم چند بایت خواهد بود؟

$$P = \sqrt{2Se} = \sqrt{2 \times 64 \times 2^{10} \times 8} = \sqrt{2^{20}} = 2^{10}$$

ساختار برنامه

در بیشتر موارد، کاربر از ماهیت صفحه بندی حافظه بی اطلاع است و در بعضی موارد مطلع است که موجب بهبود کارایی سیستم می شود. به طور مثال در سیستمی که اندازه صفحات ۴ کلمه است، یک برنامه نویس پاسکال می خواهد آرایه A را با صفر مقدار دهی اولیه کند.

```
Var A : Array [1..4] [1..4] of integer;
```

می دانیم که آرایه در پاسکال به صورت سطری در حافظه ذخیره می شود. بنابراین هر سطر آرایه یک صفحه را اشغال می کند.

الف: الگوریتم ناکارا

```
for j:=1 to 4 do
  for i:=1 to 4 do
    A[i][j] := 0;
```

این کد، یک کلمه از یک صفحه و یک کلمه از صفحه دیگر را صفر می کند و این روند ادامه می یابد. اگر سیستم عامل برای کل برنامه کمتر از چهار قاب را تخصیص دهد، اجرای آن منجر به 16 خطای صفحه می گردد. ($4 \times 4 = 16$)

ب: الگوریتم کارا

```
for i=1 to 4 do
  for j=1 to 4 do
    A[i][j]=0;
```

این کد تمام کلمات یک صفحه را صفر می کند و سپس به صفحه بعدی می پردازد و به همین علت تعداد خطای صفحه برابر چهار می شود.

بنابراین، انتخاب درست ساختمان داده ها و ساختارهای برنامه نویسی می تواند موجب افزایش محلی بودن مراجعات و کاهش نرخ خطای صفحه و تعداد صفحات موجود در مجموعه کاری شود. ■

مثال

ماتریس $A[1..4][1..4]$ به صورت ردیفی (row-major) مفروض است. دستورات زیر عناصر این ماتریس را صفر می کند:

```
for j:=1 to 4 do
  for i:=1 to 4 do
    A[i][j]:=0;
```

فرض کنید این برنامه در یک سیستم با مدیریت حافظه صفحه بندی بر حسب نیاز (demand paging) که اندازه قاب صفحه آن 8 کلمه است اجرا می شود. به این برنامه ۲ قاب صفحه اختصاص داده شده است که دستورات برنامه در یکی از این قابها بار شده است. قاب دیگر که ابتدا خالی است برای داده ها منظور شده است. اگر برای جایگزینی صفحات از روش LRU استفاده شود، تعداد کل فقدان صفحات چقدر است؟
(هر خانه آرایه از نوع integer است که یک کلمه از حافظه را اشغال می کند)

حل: حلقه‌های داده شده، ماتریس را به صورت ستون به ستون صفر می‌کند. از آنجا که ماتریس داده شده دو صفحه را اشغال می‌کند ($\frac{4 \times 4}{8} = 2$)، و هر دو سطر آن یک صفحه است، در هر با خواندن صفحه (دو سطر)، فقط دو خانه از یک ستون مقدار دهی می‌شوند و برای مقدار دهی به هر دو خانه، یک نقص صفحه رخ می‌دهد و تعداد کل نقص صفحه‌ها برابر است با:

$$\blacksquare \frac{4 \times 4}{2} = 8$$

قطعه بندی

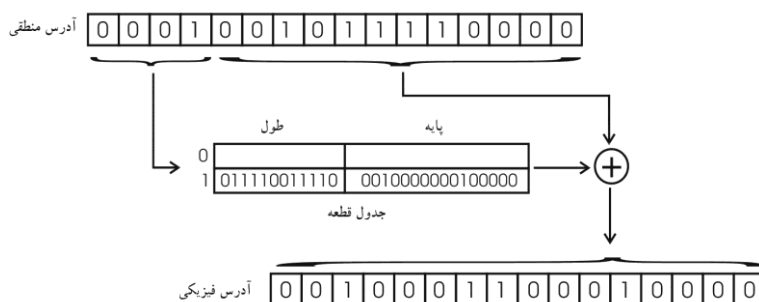
در روش قطعه بندی برای مدیریت حافظه، برنامه و داده‌ها به تعدادی قطعه (Segment) تقسیم می‌شوند و لزومی ندارد اندازه این قطعه‌ها هم اندازه باشند. هنگامی که یک فرایند به داخل آورده می‌شود، کلیه قطعه‌های آن به داخل حافظه بار می‌شوند و جدول قطعه ایجاد می‌شود. هر سطر این جدول شامل آدرس شروع قطعه مورد نظر در حافظه اصلی و طول قطعه می‌باشد.

نکاتی در رابطه با قطعه بندی:

- ۱- آدرس منطقی در قطعه بندی از دو قسمت تشکیل یافته است: شماره قطعه و آفست.
- ۲- امتیاز قطعه بندی، حفاظت از قطعات و اشتراک داده‌ها و کد می‌باشد.
- ۳- الگوی صفحه بندی نمی‌تواند حافظه فیزیکی را از دیدگاه کاربر نسبت به حافظه تفکیک کند.
- ۴- قطعه بندی به دلیل بکارگیری قطعه‌های غیرهم اندازه، مشابه بخش بندی پویا است.
- ۵- تفاوت قطعه بندی با بخش بندی در این است که یک برنامه می‌تواند بیش از یک بخش را اشغال کند و لزومی ندارد این بخشها پیوسته باشند.
- ۶- قطعه بندی تکه تکه شدن داخلی را حذف می‌کند، اما دارای تکه تکه شدن خارجی است.
- ۷- تکه تکه شدن خارجی هم در بخش بندی پویا و هم در قطعه بندی وجود دارد. اما چون یک فرایند به قطعه‌های کوچکتر شکسته می‌شود، تکه تکه شدن خارجی در قطعه بندی کمتر است.
- ۸- در حالی که صفحه بندی از دید برنامه‌ساز مخفی است، قطعه بندی قابل رویت و عامل تسهیل سازماندهی برنامه‌ها و داده‌ها می‌باشد.

مثال

آدرس منطقی 16 بیتی 0001001011110000 مفروض است. نحوه تولید آدرس فیزیکی در روش قطعه بندی را نشان دهید. (آفست 12 بیتی و شماره قطعه 4 بیتی)



تذکر: آدرس فیزیکی از جمع آفست 12 بیتی با مقدار پایه بدست می آید.

تذکر: حداکثر اندازه قطعه برابر $2^{12} = 4096$ می باشد. ■

مثال

در یک سیستم حافظه قطعه بندی ساده، با جدول قطعه زیر، کدام آدرس منطقی (0,150)، (2,700) فاقد آدرس فیزیکی هستند؟

	Base	Length
0	500	200
1	700	1000
2	400	600

پاسخ: آدرس منطقی در سیستم قطعه بندی از دو قسمت (شماره قطعه و آفست) تشکیل شده، بنابراین آدرس منطقی (2,700) آدرس فیزیکی ندارد، چون $(700 > 600)$.

آدرس منطقی (0,150)، دارای آدرس فیزیکی است، چون $150 < 200$. این آدرس برابر است با: $500 + 150 = 650$.

مزایای سازماندهی بر اساس قطعه بندی

۱- ساده شدن اداره ساختمان داده‌های رشد کننده

۲- میسر ساختن اشتراک بین فرایندها

۳- میسر شدن حفاظت

۴- امکان تغییر برنامه‌ها به صورت مستقل و ترجمه آنها بدون نیاز به پیوند و بار شدن همه مجموعه‌ها

قطعه بندی درخواستی

در قطعه بندی ساده، هر فرایند جدول قطعه خودش را دارد و هرگاه تمام قطعه‌های مربوط به آن فرایند به داخل حافظه اصلی بار شدند، جدول قطعه برای آن فرایند ایجاد و داخل حافظه بار می‌شود. اما در حافظه مجازی با قطعه بندی، لزومی ندارد تمام قطعه‌های یک فرایند در حافظه اصلی باشند و می‌توانند برحسب نیاز به داخل خوانده شوند. هر مدخل جدول قطعه شامل موارد زیر است:

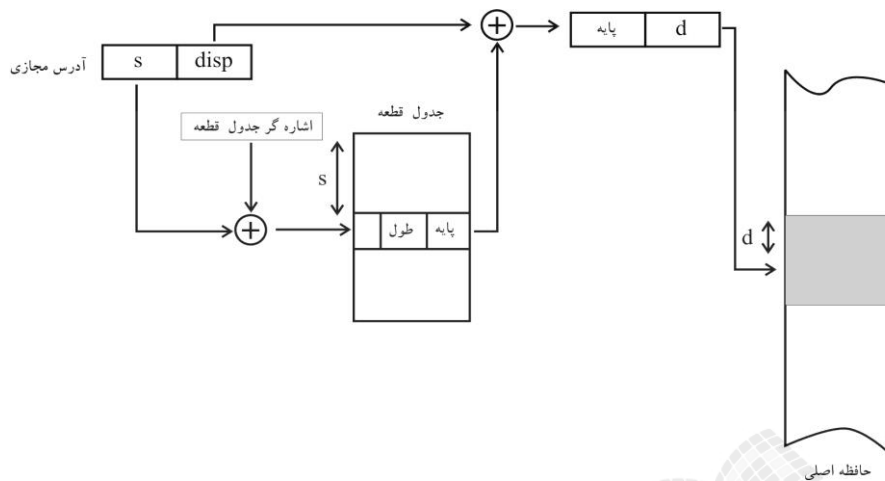
۱- طول قطعه ۲- پایه قطعه ۳- بیت حضور (P)

۴- بیت تغییر (M) ۵- بیت‌های کنترلی دیگر

اگر بیت حضور فعال باشد (قطعه مورد نظر در حافظه اصلی باشد)، آدرس شروع قطعه و طول قطعه نیز در مدخل جدول در نظر گرفته می‌شود.

هنگامی که فرایندی در حال اجرا است، آدرس شروع جدول قطعه این فرایند در یک ثبات نگهداری می‌شود.

شکل زیر نحوه ترجمه آدرس در یک سیستم قطعه بندی مجازی را نشان می‌دهد:



قطعه بندی صفحه بندی (Segmentation with paging)

در سیستم ترکیبی قطعه بندی و صفحه بندی، فضای آدرس به تعدادی قطعه تقسیم می‌شود و هر قطعه نیز به تعدادی صفحه تقسیم می‌شوند. برای هر فرایند یک جدول قطعه و چند جدول صفحه وجود دارد. آدرس مجازی در این سیستم از ۳ قسمت تشکیل شده است:

آفست درون صفحه	شماره صفحه	شماره قطعه
----------------	------------	------------

آدرس فیزیکی نیز برابر است با:

آفست درون صفحه	شماره قاب صفحه
----------------	----------------

(PTBA: Page Table Base Address)

مثال

یک حافظه به اندازه 8KB و با صفحات 512 بیتی را در نظر بگیرید که با روش قطعات صفحه بندی شده مدیریت می شود. جدول قطعه به صورت زیر است:

	PTBA	Limit
0	0070	8
1	0078	8
2	0086	8
3	0094	8

آدرس فیزیکی متناظر با آدرس مجازی زیر را مشخص کنید.

S#	P#	Offset
11	101	1010101101

(لازم است بدانید در خانه به آدرس حافظه 0099 مقدار 9H ذخیره شده است.)

پاسخ:

با توجه به آدرس مجازی داده شده، چون شماره قطعه برابر 3 (همان 11 در مبنای دو) می باشد، به اندیس 3 در جدول قطعه مراجعه کرده و مقدار PTBA یعنی 0094 مشخص می شود. این مقدار را با شماره صفحه یعنی 5 (همان 101 در مبنای دو) جمع می کنیم و حاصل برابر 0099 می شود. در این آدرس، شماره قاب نوشته شده است (9H). با قرار دادن این مقدار قبل از آفست، آدرس فیزیکی بدست می آید:

P#	offset
1001	1010101101

که این آدرس (یعنی 10011010101101) در مبنای شانزده برابر است با: 26AD

مثال

مدیریت حافظه در یک سیستم فرضی به صورت قطعه بندی صفحه بندی شده است و اندازه هر صفحه 4 کیلو بایت است، یعنی آفست 12 بیتی است. در PCB یک فرایند برای آدرس پایه جدول قطعه (STBA) مقدار 0B05H دیده می شود. اگر در این فرایند آدرس منطقی [01H , 4678H] تولید شود، آدرس فیزیکی نظیر چه خواهد بود؟

(بخش اول آدرس منطقی شماره قطعه است.) (حرف H به معنی Hex است.)

هر درایه جدول قطعه سه بیتی است. بایت اول و دوم نشان دهنده آدرس پایه جدول صفحه (PTBA) و بایت سوم مشخص کننده LIMIT می باشد.

هر درایه جدول صفحه یک بایتی است و نشان دهنده شماره قاب است.
محتویات حافظه به صورت زیر است: (0B08=0B, 0B09=07, 0B0B=0D)

پاسخ:

در مدیریت حافظه به روش "قطعه بندی صفحه بندی شده"، آدرس منطقی از سه قسمت تشکیل شده است:

آفست	شماره صفحه	شماره قطعه
------	------------	------------

در آدرس منطقی داده شده [01,4678]، قسمت اول یعنی 01، شماره قطعه می باشد. همچنین چون آفست 12 بیتی است و چون یک عدد 12 بیتی در مبنای دو، معادل یک عدد 3 رقمی در مبنای شانزده می باشد، در نتیجه آفست برابر سه رقم آخر یعنی 678 می باشد. بنابراین شماره صفحه برابر 4 است. یعنی آدرس منطقی برابر است با:

01	4	678
----	---	-----

از آنجا که آدرس شروع جدول قطعه برابر 0B05، شماره قطعه برابر 01 و هر سطر جدول قطعه 3 بایتی است، به آدرس $0B05 + 3 = 0B08$ مراجعه می کنیم. در این آدرس دو بایت اول، مشخص کننده آدرس پایه جدول صفحه (PTBA) می باشد که برابر 0B07 است. از آنجا که شماره صفحه برابر 4 و اندازه هر سطر جدول صفحه یک بایتی است، به آدرس $0B07 + 4 = 0B0B$ مراجعه کرده تا شماره قاب را پیدا کنیم، که برابر 0D است.

در نهایت با قرار دادن آفست یعنی 678 بعد از شماره قاب، آدرس فیزیکی مشخص می شود:

آفست	شماره قاب	\Rightarrow	0D	678
------	-----------	---------------	----	-----

تذکر: آفست در آدرس منطقی و فیزیکی یکسان است. (چون اندازه صفحه و اندازه قاب برابر است)



مقایسه روشهای مدیریت حافظه

در جدول زیر، چهار نوع مدیریت حافظه با یکدیگر مقایسه شده اند:

صفحه بندی ساده	صفحه بندی حافظه مجازی	قطعه بندی ساده	قطعه بندی حافظه مجازی
حافظه اصلی به تکه های هم اندازه به نام قاب تقسیم می شود.	حافظه اصلی به تکه های هم اندازه به نام قاب تقسیم می شود.	حافظه اصلی تقسیم نمی شود.	حافظه اصلی تقسیم نمی شود.
برنامه توسط مترجم یا سیستم مدیریت حافظه تقسیم می شود.	برنامه توسط مترجم یا سیستم مدیریت حافظه به صفحه ها تقسیم می شود.	قطعه های برنامه توسط برنامه ساز به مترجم اطلاع داده می شوند. یعنی تصمیم گیری با برنامه ساز است.	قطعه های برنامه توسط مترجم یا برنامه ساز به مترجم اطلاع داده می شوند. یعنی تصمیم گیری با برنامه ساز است.
بدون تکه تکه شدن خارجی	بدون تکه تکه شدن خارجی درجه چندبرنامگی بالاتر فضای مجازی بزرگ برای فرایند حفاظت	بدون تکه تکه شدن داخلی	بدون تکه تکه شدن داخلی درجه چندبرنامگی بالاتر فضای آدرس مجازی بزرگ حمایت از اشتراک و حفاظت
تکه تکه شدن داخلی درون قاب	تکه تکه شدن داخلی درون قاب سربرار پیچیدگی مدیریت حافظه	تکه تکه شدن خارجی گسترش به کار گیری حافظه کاهش سربرار نسبت به بخش بندی پویا	تکه تکه شدن داخلی درون قاب سربرار پیچیدگی مدیریت حافظه
سیستم عامل باید فهرست قابهای آزاد را نگهداری کند.	سیستم عامل باید فهرست قابهای آزاد را نگهداری کند.	سیستم عامل باید فهرست حفره های آزاد در حافظه اصلی را نگهداری کند.	سیستم عامل باید فهرست قابهای آزاد را نگهداری کند.
پردازنده از شماره صفحه و انحراف برای محاسبه آدرس مطلق استفاده می کند.	پردازنده از شماره صفحه و انحراف برای محاسبه آدرس مطلق استفاده می کند.	پردازنده از شماره قطعه و انحراف برای محاسبه آدرس مطلق استفاده می کند.	پردازنده از شماره صفحه و انحراف برای محاسبه آدرس مطلق استفاده می کند.
تمام صفحه های یک فرایند باید در حافظه اصلی باشند تا فرایند اجرا شود، مگر اینکه روی هم گذاری به کار رفته باشد.	تمام صفحه های یک فرایند باید در حافظه اصلی باشند تا فرایند اجرا شود، مگر اینکه روی هم گذاری به کار رفته باشد.	یک فرایند به تعدادی قطعه تقسیم شده و تمام قطعه ها باید در حافظه اصلی باشند تا فرایند اجرا شود، مگر اینکه روی هم گذاری به کار رفته باشد.	تمام صفحه های یک فرایند باید در حافظه اصلی باشند تا فرایند اجرا شود، مگر اینکه روی هم گذاری به کار رفته باشد.
	خواندن یک صفحه به داخل حافظه اصلی، ممکن است نیازمند نوشتن یک قطعه بر روی دیسک باشد.		خواندن یک صفحه به داخل حافظه اصلی، ممکن است نیازمند نوشتن یک صفحه بر روی دیسک باشد.

کنکور ارشد

(مهندسی کامپیوتر - دولتی ۸۶)

۱- در یک سیستم حافظه صفحه بندی ساده (Simple Paging) حافظه فیزیکی دارای 2^{24} بایت است. 256 صفحه فضای آدرس منطقی را تشکیل می دهد و اندازه صفحات 2^{10} بایت است. کدام یک از گزینه های زیر تعداد بیت های آدرس منطقی و اندازه جدول صفحه را مشخص می کند؟

(۱) 18 بیت و 256 عضو

(۲) 18 بیت و 16 کیلو عضو

(۳) 24 بیت و 256 عضو

(۴) 24 بیت و 16 کیلو عضو

پاسخ: جواب گزینه ۱ است.

تعداد درایه های جدول صفحه با تعداد صفحات فضای آدرس منطقی برابر است. بنابراین جدول صفحه دارای 256 عضو است. اندازه حافظه منطقی از حاصل ضرب تعداد صفحات در اندازه صفحه بدست می آید:

$$256 \times 2^{10} = 2^{18}$$

بنابراین تعداد بیت های آدرس منطقی برابر 18 می باشد.

(مهندسی کامپیوتر - دولتی ۷۱)

۲- در یک سیستم با مدیریت حافظه Overlay، برنامه زیر اجراء می شود. این برنامه با پردازش A شروع و در انتهای این پردازش پایان می پذیرد. اندازه پردازش ها (کیلو بایت) عبارتند از: $A=5$

$B=9, C=10, D=15, E=7, F=10$

اگر Overlay Driver احتیاج به 2 کیلو بایت حافظه داشته باشد، حداقل فضای مورد نیاز جهت اجرای این برنامه کدام است؟

<pre> procedure A; . call D; . call F; . end; </pre>	<pre> procedure B; end; </pre>	<pre> Procedure C; end; </pre>
<pre> procedure D; . call E; . call B; end; </pre>	<pre> procedure E; . . . end; </pre>	<pre> Procedure F; . call C; . end; </pre>

30 K (۴)

29 K (۳)

27 K (۲)

31 K (۱)

پاسخ: گزینه ۱ صحیح است.

در این روش هر زیر برنامه ای که صدا زده می شود، در حافظه فضائی به آن اختصاص می یابد و هنگامی که زیر برنامه پایان می یابد، حافظه آزاد می شود. با توجه به اینکه این برنامه با پردازنده A شروع می شود، در ابتدا فضای 5K از حافظه به فرایند A داده می شود. در داخل این فرایند، فرایند D اجرا شده و 15K به آن داده می شود. در داخل فرایند D، فرایند E اجرا شده و 7K حافظه به آن داده می شود. تا این لحظه 27K حافظه تخصیص داده شده است. با اتمام E، فضای داده شده به آن آزاد شده و سپس با اجرای B، 9K فضا به آن داده شده و در مجموع به 29K فضا نیاز خواهیم داشت. بعد از اتمام اجرای B، اجرای D نیز پایان یافته و F اجرا می شود. در داخل این فرایند نیز C اجرا می شود. در این حالت به 25K حافظه نیاز است. با توجه به این توضیحات، بیشترین حافظه مورد نیاز مربوط به حالت $A \rightarrow D \rightarrow B$ می باشد که به 29 کیلو بایت حافظه نیاز است. البته خود درایور نیاز به 2K دارد، بنابراین در مجموع حداقل به 31K حافظه نیاز است.

(مهندسی IT - دولتی ۸۷)

۳- در یک سیستم صفحه بندی ساده که جدول صفحه (page table) آن 512 عنصر 16 بیتی (شامل بیت نامعتبر/معتبر (valid/invalid) است و اندازه صفحات 1 کیلوبایتی است، به ترتیب اندازه فضای آدرس فیزیکی چقدر است و آدرس فیزیکی چند بیتی است؟

$$(۱) 2^{26} - 26 \quad (۲) 2^{25} - 25 \quad (۳) 2^{24} - 24 \quad (۴) 2^{16} - 16$$

پاسخ: جواب گزینه ۲ است.

چون اندازه صفحه 1KB است، افست 10 بیتی است. از 16 بیت چون 1 بیت آن برای اعتبار است، 15 بیت برای شماره قباب صفحه باقی می ماند. پس تعداد بیت های آدرس فیزیکی برابر $15+10=25$ می باشد. همچنین اندازه حافظه فیزیکی برابر 2^{25} است.

(مهندسی کامپیوتر - دولتی ۹۲)

۴- کدام گزینه زیر درباره جدول صفحه معکوس (inverted page-table) درست نیست؟

- (۱) این نوع جدول، زمان نگاشت آدرس منطقی به آدرس فیزیکی را کاهش می دهد.
- (۲) این نوع جدول صفحه، سبب کاهش اندازه حافظه فیزیکی جهت ذخیره سازی آن می شود.
- (۳) در این نوع جدول صفحه، زمان سرویس نقص صفحه (page fault) به دلیل ایجاد یک نقص صفحه دیگر افزایش می یابد.

(۴) برای این نوع جدول صفحه، می بایست یک جدول صفحه خارجی نیز ذخیره شود.

حل: گزینه ۱ جواب است.

گزینه ۱ نادرست است، چون جدول صفحه معکوس، به علت نیاز به جستجو و تابع Hash، زمان ترجمه آدرس را بالا می برد.

(مهندسی IT - آزاد ۸۸)

۵- در یک کامپیوتر از روش جدول صفحه معکوس شده استفاده شده است. این کامپیوتر دارای آدرس مجازی 32 بیتی، حافظه فیزیکی 64 مگا بیتی و صفحات 4 کیلو بیتی است. جدول صفحه چند مدخل (Table Entry) دارد؟

$$(۱) 2^8 \quad (۲) 2^{12} \quad (۳) 2^{20} \quad (۴) 2^{14}$$

پاسخ: گزینه ۴ جواب است.

در جدول صفحه وارونه، تعداد درایه های جدول صفحه برابر است با تعداد قاب های حافظه:

$$\frac{64MB}{4KB} = \frac{2^6 \times 2^{20}}{2^2 \times 2^{10}} = 2^{14}$$

(مهندسی IT – دولتی ۸۸)

۶- با فرض اینکه جدول صفحه در حافظه ذخیره شده باشد و ۸۵٪ از ارجاعات به حافظه از طریق TLB انجام شود و هزینه هر ارجاع حافظه ۲۵۰ نانو ثانیه و ارجاع به TLB با هزینه ۵ نانو ثانیه انجام شود، با فرض عدم رخداد نقصان صفحه و عدم

توازی عملیات در معماری سیستم مذکور، هر ارجاع به حافظه به طور متوسط چقدر طول می کشد؟

(۱) ۲۸۷.۵ ثانیه (۲) ۲۹۲.۵ نانو ثانیه (۳) ۲۹۱.۷۵ ثانیه (۴) ۵۰۵ نانو ثانیه

حل: جواب گزینه ۲ است.

همان حالت اول است:

$$T_{\text{Translation}} = T_{\text{TLB}} + (1 - H) \times T_{\text{Mem}} = 5 + (0.15 \times 250) = 42.5 \text{ ns}$$

$$T_{\text{Access}} = T_{\text{Translation}} + T_{\text{Mem}} = 42.5 + 250 = 292.5 \text{ ns}$$

(مهندسی IT – آزاد ۸۴)

۷- سیستمی علاوه بر ذخیره جدول صفحه در حافظه اصلی از TLB نیز استفاده می کند. اگر زمان خواندن از حافظه اصلی ۵۰ ns و زمان خواندن از TLB برابر ۲۰ ns باشد و درصد کارایی سیستم بدون استفاده از TLB نسبت به استفاده سیستم از

TLB برابر ۸۰ درصد باشد، آن گاه نرخ برخورد TLB چقدر است؟

(۱) ۱۰ درصد (۲) ۲ درصد (۳) ۸۰ درصد (۴) ۹۰ درصد

حل: گزینه ۳ جواب است.

اگر از TLB استفاده نشود:

$$T_{\text{Access}} = T_{\text{Translation}} + T_{\text{Mem}} = 2 \times T_{\text{Mem}} = 2 \times 50 \text{ ns} = 100 \text{ ns}$$

و اگر از TLB استفاده شود:

$$T_{\text{Access}} = [T_{\text{TLB}} + (1 - H) \times T_{\text{Mem}}] + T_{\text{Mem}} = [20 + (1 - H) \times 50] + 50$$

و چون درصد کارایی برابر ۸۰٪ است، داریم:

$$\frac{70 + (1 - H) \times 50}{100} = \frac{80}{100} \Rightarrow H = 0.8 \Rightarrow H = 80\%$$

(مهندسی IT - دولتی ۸۴)

۸- در یک سیستم مدیریت حافظه، اگر زمان دسترسی به حافظه 400ns و زمان دسترسی به جدول TLB، 50ns و hit ratio در جدول TLB برابر 80٪ باشد، کدام عبارت درست است؟

(۱) سیاست صفحه بندی و TLB دارای زمان دسترسی به حافظه برابر با 530ns است.

(۲) سیاست صفحه بندی دارای زمان دسترسی به حافظه برابر با 850ns است.

(۳) سیاست صفحه بندی دارای زمان دسترسی به حافظه برابر با 730ns است.

(۴) سیاست صفحه بندی و TLB دارای زمان دسترسی به حافظه برابر با 450ns است.

پاسخ: گزینه ۱ درست است.

سیاست صفحه بندی به همراه سخت افزار TLB :

$$T_{Translation} = T_{TLB} + (1 - H) \times T_{Mem} = 50 + (0.2 \times 400) = 130ns$$

$$T_{Access} = T_{Translation} + T_{Mem} = 130 + 400 = 530ns$$

گزینه ۲ و ۳ نادرست است، چون در سیاست صفحه بندی بدون سخت افزار TLB :

$$T_{Translation} = T_{Mem} = 400ns$$

$$T_{Access} = T_{Translation} + T_{Mem} = 400 + 400 = 800ns$$



زمان موثر دسترسی در صورت استفاده از حافظه پنهان و با در نظر گرفتن احتمال وقوع نقص صفحه:

$$T_{Access} = T_{Translation} + T_{CM} + P \times T_{Disk}$$

$$T_{Translation} = T_{TLB} + (1 - H) \times T_{CM}$$

$$T_{CM} = T_{Cache} + (1 - H_{Cache}) \times T_{Penalty}$$

$T_{Penalty}$: جریمه هر عدم اصابت در حافظه پنهان

(مهندسی IT - دولتی ۸۹)

۹- یک حافظه مجازی با این مشخصات در نظر بگیرید.

زمان دسترسی حافظه 50ns و زمان دستیابی TLB برابر 2ns، نسبت اصابت TLB برابر 98٪ و احتمال خطای صفحه برای

کل دسترسی ها به حافظه 2×10^{-6} است. زمان انتقال صفحه از دیسک را 10ms فرض کنید.

برای سرعت بخشیدن به این حافظه از حافظه پنهان (cache) با این مشخصات استفاده شده است:

زمان دسترسی حافظه پنهان 10ns و نسبت اصابت حافظه پنهان 90٪، جریمه هر عدم اصابت در حافظه پنهان 100ns

است.

میانگین زمان دسترسی به حافظه برای هر آدرس، به کدام یک از گزینه های زیر نزدیک تر است؟

75ns (۱) 45ns (۲) 73ns (۳) 43ns (۴)

پاسخ: جواب گزینه ۴ است.

$$T_{Translation} = T_{TLB} + (1 - H) \times T_{CM} = 2 + (1 - 0.98) \times 20 = 2.4ns$$

$$T_{CM} = T_{Cache} + (1 - H_{cache}) \times T_{Penalty} = 10 + (1 - 0.9) \times 100 = 20ns$$

$$T_{Access} = T_{Translation} + T_{CM} + P \times T_{Disk} = 2.4 + 20 + (2 \times 10^{-6})(10 \times 10^{-3}) = 43ns$$

(مهندسی IT – دولتی ۸۶)

۱۰- در یک سیستم حافظه صفحه بندی، در یک برنامه به ترتیب به صفحات زیر رجوع شده است:

0, 1, 4, 2, 0, 2, 6, 5, 1, 2, 3, 2, 1, 2, 6, 2, 1, 3, 6, 2

اگر برای این برنامه سه قاب صفحه (Page Frame) در نظر گرفته می شود و از الگوریتم جابه جایی FIFO استفاده شود، تعداد خطاهای صفحه (Page Faults) برابر است با:

10 (۱) 12 (۲) 14 (۳) 13 (۴)

پاسخ: جواب گزینه ۴ است.

0	1	4	2	0	2	6	5	1	2	3	2	1	2	6	2	1	3	6	2
0	0	0	1	4	4	2	0	6	5	1	1	1	1	2	2	3	3	3	6
	1	1	4	2	2	0	6	5	1	2	2	2	2	3	3	6	6	6	1
		4	2	0	0	6	5	1	2	3	3	3	3	6	6	1	1	1	2
F	F	F	F	F		F	F	F	F	F				F		F			F

(مهندسی کامپیوتر – دولتی ۸۶)

۱۱- حافظه اصلی کامپیوتری دارای چهار قاب صفحه می باشد. زمان بار شدن، زمان آخرین دسترسی، بیت R (Reference) و بیت M (Modify) مربوط به هر یک از صفحات در جدول زیر آمده است. اگر خطای صفحه روی صفحه مجازی شماره 4 در زمان رخ دهد، تحت الگوریتم های جایگزینی LRU, NRU به ترتیب محتویات کدام یک از قاب صفحه ها، بایستی جابجا شوند؟

قاب صفحه	شماره صفحه مجازی	زمان بار شدن	زمان آخرین دسترسی	R	M
0	2	125	278	0	1
1	1	229	239	1	0
2	0	119	271	1	0
3	3	159	318	1	1

۱) یک و دو ۲) یک و صفر ۳) دو و یک ۴) سه و صفر

پاسخ: جواب گزینه ۲ است.

در روش LRU، صفحه ای خارج می شود که در گذشته دورتری به آن مراجعه شده است. یعنی زمان آخرین دسترسی اش از همه کمتر باشد. بنابراین صفحه شماره 1، که آخرین دسترسی به آن در زمان 239 بوده است، انتخاب می شود. در روش NRU، صفحه 2 انتخاب می شود، چون در دسته با شماره کمتری قرار دارد.

تذکر: در صورت تست، شماره قاب‌ها خواسته شده است. بنابراین چون صفحه 1 در قاب 1 و صفحه 2 در قاب 0 قرار دارد، گزینه ۲ جواب است.

تذکر: چون در زمان رخ دادن خطای صفحه، یعنی 319، همه صفحه‌ها موجود هستند، تصمیم‌گیری بین همه آنها انجام گرفته است.

(مهندسی کامپیوتر – آزاد ۹۰)

۱۲- یک سیستم با 128 مگا بایت حافظه اصلی را در نظر بگیرید که از تکنیک صفحه بندی با اندازه قاب 4 کیلو بایت استفاده می‌کند. اگر بخواهیم الگوریتم کمترین استفاده در گذشته اخیر (LRU) را با روش ماتریس دو بعدی تقریب بزنییم، چقدر حافظه برای ذخیره سازی ماتریس نیاز است؟

(۱) 16 کیلو بایت (۲) 16 مگا بایت (۳) 128 مگا بایت (۴) 32 کیلو بایت

پاسخ: جواب گزینه ۳ است.

ابتدا باید تعداد قاب‌ها را مشخص کرد:

$$\frac{128\text{MB}}{4\text{KB}} = \frac{128 \times 2^{20}}{4 \times 2^{10}} = 2^{15}$$

بنابراین ماتریس مورد نیاز $2^{15} \times 2^{15}$ بیتی یا 128 مگا بیتی است.

نحوه تبدیل به بایت:

$$\frac{2^{15} \times 2^{15}}{8} = 2^{27} = 2^7 \times 2^{20} = 128\text{MB}$$

بنابراین به 128 مگا بایت (برابر اندازه حافظه اصلی)، حافظه نیاز داریم.

(مهندسی کامپیوتر – دولتی ۸۴)

۱۳- سیستمی را در نظر بگیرید که در حالت Thrashing می‌باشد. در این صورت کدام یک از شرایط زیر را لزوماً در سیستم خواهیم داشت؟

(۱) CPU در 100% اشتغال خود خواهد بود.

(۲) CPU، 100% بیکار (Idle) خواهد بود.

(۳) صفحه بندی دیسک (Paging disk) در حالت 100% (صد درصد) ظرفیت خود خواهد بود.

(۴) صفحه بندی دیسک (Paging disk) تقریباً 100% (صد درصد) فعال خواهد بود.

پاسخ: جواب گزینه ۴ است.

در حالت Thrashing، نرخ نقص صفحه بسیار بالا و بهره وری CPU بسیار پایین و نزدیک به صفر (نه الزماً صفر) است. پس گزینه های ۱ و ۲ نادرست هستند. همچنین در صد عملیات مبادله صفحات بین حافظه اصلی و دیسک بسیار بالا حتی نزدیک به 100% (نه الزماً 100%) می‌باشد. بنابراین گزینه ۴ که از عبارت "تقریباً" استفاده کرده، صحیح می‌باشد.

(مهندسی کامپیوتر – آزاد ۸۵)

۱۴- اگر فرایندی اکثر زمان هایش را به جای اجرا، اختصاص به صفحه بندی دهد، این عمل چه نام دارد؟

(۱) Prepaging (۲) Demand Paging (۳) Local Allocation (۴) Thrashing

پاسخ: گزینه ۴ جواب است.

(مهندسی IT – آزاد ۸۷)

۱۵- کدام یک از گزینه های زیر در مورد الگوریتم جایگزینی صفحه درست نیست؟

- (۱) الگوریتم بهینه با $n+1$ قاب صفحه همواره بهتر یا مساوی با الگوریتم بهینه با n قاب صفحه عمل می کند.
- (۲) الگوریتم LRU با $n+1$ قاب صفحه همواره بهتر یا مساوی با LRU با n قاب صفحه عمل می کند.
- (۳) الگوریتم FIFO با $n+1$ قاب صفحه بعضی مواقع بدتر از الگوریتم FIFO با n قاب صفحه عمل می کند.
- (۴) الگوریتم LRU همواره بدتر از الگوریتم بهینه عمل می کند.

پاسخ: جواب گزینه ۴ است.

گاهی ممکن است عملکرد الگوریتم LRU به خوبی عملکرد الگوریتم بهینه باشد.

گزینه دیگر درست می باشند، چون BO و LRU ناهنجاری بلیدی ندارند ولی FIFO، ناهنجاری بلیدی دارد. ■

(مهندسی IT – آزاد ۹۰)

۱۶- در چه صورت با پدیده Beladys Anomaly روبرو می شویم؟

- (۱) زمانی که الگوریتم جایگزینی صفحه از نوع Stack باشد.
- (۲) زمانی که رابطه مستقیم بین تعداد قاب های صفحه و تعداد نقض صفحه وجود داشته باشد.
- (۳) زمانی که از الگوریتم جایگزینی صفحه بهینه (Optimal) استفاده می کنیم.
- (۴) زمانی که مجموعه صفحات در حافظه با n قاب زیر مجموعه ای در حافظه با $n+1$ قاب باشد.

پاسخ: جواب گزینه ۲ است.

زمانی این پدیده رخ می دهد که رابطه مستقیم بین تعداد قاب های صفحه و تعداد نقض صفحه وجود داشته باشد. یعنی با

افزایش تعداد قاب، تعداد نقض صفحه نیز افزایش یابد. ■

(مهندسی کامپیوتر – آزاد ۸۸)

۱۷- کدام الگوریتم یک الگوریتم جایگزینی صفحه Stack، محسوب نمی شود؟

(۱) FIFO (۲) LRU (۳) NFU (۴) Optimal

پاسخ: گزینه ۱ جواب است. ■

(مهندسی IT - دولتی ۸۴)

۱۸- اندازه صفحه در سیستمی با مدیریت حافظه مجازی به صورت صفحه بندی درخواستی، 256 بایت است. حافظه سیستم حاوی سه قاب صفحه (در ابتدا خالی) می باشد. هر قاب صفحه می تواند به کد یا داده انتساب شود و قاب های صفحه به اشتراک بین کد و داده استفاده می شوند. اندازه کد فرایند برابر یک صفحه است و فرض کنید که حافظه فرایند فقط از دو بخش کد و داده تشکیل می شود. اگر از روش جایگزینی FIFO استفاده شود، اجرای کد زیر منجر به چند نقص صفحه خواهد شد؟

X : array [1..128][1..128] of byte

for register int i=1 to 128 do

for register int j=1 to 128 do

X[i][j]=0;

1) 65 2) 8193 3) کمتر از 65 4) بیشتر از 65 و کمتر از 100

پاسخ: گزینه ۴ درست است.

هر سطر ماتریس (128 عنصر 1 بایتی) نیاز به 128 بایت حافظه دارد. بنابراین در یک صفحه 256 بایتی، دو سطر ماتریس جا می شود. بنابراین ماتریس با 128 سطر به 64 صفحه نیاز دارد. چون برنامه به صورت ردیفی، ماتریس را مقدار دهی می کند، بنابراین 64 نقص صفحه برای داده ها رخ می دهد. از طرفی چون برای کد برنامه، قاب خاصی به صورت جداگانه در نظر گرفته نشده است، صفحه حاوی کد نیز مانند صفحات حاوی داده، طبق الگوریتم FIFO خارج می شود و چون کد به طور دائم در حال اجرا است، به محض خروج دوباره نقص صفحه رخ می دهد و به حافظه بر می گردد. بنابراین چون با ورود هر سه صفحه داده، یک کد خارج شده و نقص صفحه رخ می دهد، پس تعداد $\left\lfloor \frac{64}{3} \right\rfloor$ یعنی 22 نقص صفحه برای کد رخ می دهد. بنابراین در کل تعداد نقص صفحه ها برابر $22 + 64 = 86$ یعنی 86 می باشد که از 65 بیشتر و از 100 کمتر است.

(مهندسی IT - دولتی ۸۴)

۱۹- در مورد آدرس مجازی زیر در یک سیستم مدیریت حافظه که از ترکیب قطعه بندی و صفحه بندی با جداول صفحه دو سطحی بهره می برد، کدام عبارت نادرست است؟

Segment No	PT1	PT2	Offset
9 Bit	5 Bit	7 Bit	13 Bit

۱) اندازه صفحه 8K و اندازه حافظه مجازی هر فرایند 16 G است.

۲) حداکثر تعداد جداول صفحه سطح دو در هر قطعه برابر 128 است.

۳) حداکثر تعداد جداول صفحه سطح یک در هر فرایند برابر 512 است.

۴) حداکثر اندازه هر قطعه برابر 32M، حداکثر تعداد صفحات در هر قطعه برابر 4096 (4K) و حداکثر تعداد صفحات در هر فرایند برابر 2M است.

پاسخ: جواب گزینه ۲ است.

گزینه ۲ نادرست است، چون حداکثر تعداد جداول صفحه سطح دو، در هر قطعه برابر $2^5 = 32$ است.

علت درستی گزینه های دیگر:

$$\text{اندازه صفحه} = 2^{13} = 8\text{K}$$

$$\text{اندازه حافظه مجازی هر فرایند} = 2^{(9+5+7+13)} = 2^{34} = 2^4 \times 2^{30} = 16\text{G}$$

$$\text{حداکثر اندازه هر قطعه} = 2^{(5+7+13)} = 2^{25} = 2^5 \times 2^{20} = 32\text{M}$$

$$\text{حداکثر تعداد جداول صفحه سطح یک در هر فرایند} = 2^9 = 512$$

$$\text{حداکثر تعداد صفحات در هر قطعه} = 2^{(5+7)} = 2^{12} = 2^2 \times 2^{10} = 4\text{K}$$

$$\text{حداکثر تعداد صفحات در هر فرایند} = 2^{(9+5+7)} = 2^{21} = 2 \times 2^{20} = 2\text{M}$$



فصل ۷:

مدیریت ورودی و خروجی - مدیریت دیسک

در این فصل به نحوه اداره I/O توسط سیستم عامل می پردازیم. همچنین زمان بندی دیسک ها را مورد بررسی قرار می دهیم.

نرم افزار I/O

نرم افزار I/O دارای چهار لایه می باشد که بر روی سخت افزار قرار دارند. این لایه ها در شکل زیر نشان داده شده اند:

فرایند کاربر
نرم افزار مستقل از دستگاه
گرداننده دستگاه
اداره کننده وقفه
سخت افزار

نحوه عملکرد لایه ها در هنگام خواندن بلوکی از فایل توسط فرایند کاربر

- ۱- توسط فرایند کاربر، سیستم عامل برای خواندن بلوک فراخوانی می شود. (فراخوانی سیستمی)
- ۲- بلوک در حافظه پنهان بافر توسط نرم افزار مستقل از دستگاه جستجو می شود.
- ۳- اگر بلوک در حافظه پنهان پیدا نشد، گرداننده دستگاه برای صدور فرمان به سخت افزار برای آوردن بلوک از دیسک فراخوانی می شود. (فرایند تا کامل شدن عملیات دیسک، بلوکه می شود.)
- ۴- سخت افزار بعد از پایان کار دیسک، وقفه ای ایجاد کرده و اداره کننده وقفه اجرا می شود تا وضعیت دستگاه را چک کرده و گرداننده دستگاه را بیدار سازد. سپس نرم افزار مستقل از دستگاه و بعد فرایند کاربر، بیدار شده و نتیجه عملیات را دریافت می کنند.

مدیریت دیسک

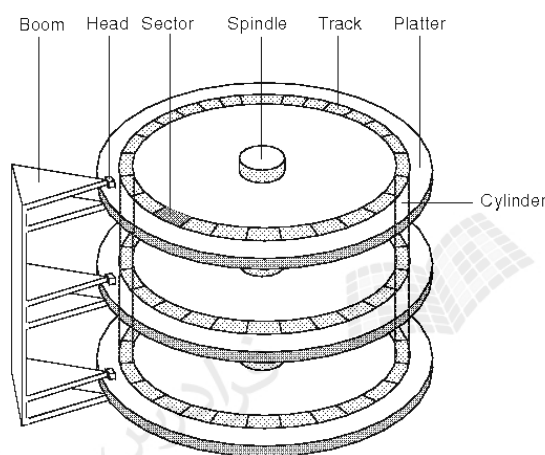
یکی از وظایف سیستم عامل، مدیریت دیسک می باشد. ابتدا دیسک مغناطیسی را تشریح می کنیم.

دیسک مغناطیسی

دیسک مغناطیسی رسانه ای گردان با امکان دستیابی مستقیم به داده های ذخیره شده می باشد. دیسک از صفحه ای مدور و مغناطیس شونده که حول یک محور عمودی می چرخد، تشکیل شده است. رویه‌های این صفحه از غشاء فرو مغناطیسی پوشیده شده که بر روی آنها شیارهایی به صورت دایره‌های متحدالمرکز وجود دارد. شیارها از بیرون به درون با شروع از صفر شماره گذاری شده اند.

تمام شیارهای هم شعاع، تشکیل یک استوانه را می دهند.

شکل زیر یک دیسک پک ۳ صفحه ای را نشان می دهد. هدها به یک بازوی دیسک متصل اند که در راستای شعاع دایره حرکت می کنند.



نرم افزار دیسک

در بخش نرم افزار دیسک، به نحوه محاسبه زمان دستیابی اطلاعات و الگوریتم های زمانبندی دیسک می پردازیم.

زمان استوانه جویی (Seek time)

زمانی که طول می کشد تا نوک خواندن/نوشتن به استوانه ای که داده مورد نظر در آن قرار دارد برسد. متوسط این زمان را با S نمایش می دهند و واحد آن میلی ثانیه است.

زمان درنگ دوران (Rotational latency time)

زمانی که طول می کشد تا ابتدای داده مورد نظر در اثر دوران دیسک به زیر نوک R/W برسد. که واحد آن میلی ثانیه است.

زمان یک دور چرخش دیسک

زمان یک دور کامل چرخش دیسک (2r) از رابطه زیر محاسبه می شود. واحد این زمان میلی ثانیه است:

$$2r = \frac{60000}{\text{rpm}}$$

که rpm سرعت چرخش دیسک است و واحد آن دور در دقیقه است. متوسط زمان درنگ دورانی از رابطه زیر محاسبه می شود:

$$r = \frac{30000}{\text{rpm}}$$

زمان دستیابی

زمان دستیابی به دیسک از مجموع زمان های استوانه جویی، درنگ دورانی و انتقال محاسبه می شود.

زمان دسترسی به فایل

الف - فایل بر روی n سکتورهای پراکنده ذخیره شده است.

$$n(s + r + t)$$

S : متوسط زمان استوانه جویی

r : متوسط زمان درنگ دورانی

t : زمان خواندن یک سکتور

ب - فایل بر روی k شیار پشت سرهم ذخیره شده است.

$$(s + r + 2r) + (k - 1)(r + 2r)$$

این زمان تشکیل شده است از مجموع زمان خواندن شیار اول (s+r+2r) و زمان خواندن شیارهای بعدی

تذکر: متوسط زمان جستجو یعنی S ، فقط برای شیار اول در نظر گرفته می شود.

الگوریتم های زمان بندی بازوی دیسک

۱- خروج به ترتیب ورود (FCFS (First Come First Serviced)

در خواست ها به ترتیب ورود به صف، اجرا می‌شوند.

۲- ابتدا کوتاهترین زمان جستجو (SSTF (Shortest Seek Time First)

بازو به سمت درخواستی حرکت می‌کند که نزدیک ترین درخواست بعدی به مکان فعلی باشد. به عبارتی درخواستی به کمترین زمان برای حرکت بازو نیاز دارد.

۳- مرور (آسانسور) : Scan:

در ابتدا بازو به جهتی حرکت می‌کند که کوتاهترین زمان استوانه جویی را برای دستیابی نیاز دارد. اگر در جهت انتخاب شده به همه درخواستها پاسخ داده شد، جهت حرکت عوض می‌شود.

۴- مرور مدور: C-Scan

مانند روش Scan است، با این تفاوت که پس از پاسخ به آخرین درخواست در یک جهت (مثلا رو به بالا)، بازو بلافاصله به پایین ترین شماره سیلندر رفته و به سمت بالا حرکت می‌کند. تذکر: نام دیگر روش SCAN، روش LOOK می‌باشد.

مثال

صف درخواستهای سیلندر به صورت ۱۴، ۲۰، ۹، ۵، ۱۲ می‌باشد و هد بر روی سیلندر ۱۰ قرار دارد. در صورت استفاده از هر یک از الگوریتم های کنترل حرکت بازو، ترتیب حرکت هد را مشخص کنید؟

FCFS : 10, 12, 5, 9, 20, 14

SCAN : 10, 9, 5, 12, 14, 20

SSTF : 10, 9, 12, 14, 20, 5

C-SCAN : 10, 9, 5, 20, 14, 12



مثال

در یک دیسک سخت، نوک I/O HEAD روی سیلندر ۲۰ قرار دارد. اگر تقاضا برای خواندن سیلندرهایی به ترتیب ۱۰، ۲۲، ۲۰، ۲، ۴۰، ۶ و ۳۸ به Driver آن وارد شود و چنانچه حرکت هد I/O بین دو سیلندر مجاور ۶ میلی ثانیه طول بکشد، در صورت استفاده از الگوریتم SSTF برای خواندن سیلندرهایی، کل seek time مورد نیاز چقدر خواهد بود؟

حل: در SSTF همواره به سمت نزدیکترین سیلندر حرکت می‌شود:

20 → 22 → 10 → 6 → 2 → 38 → 40

مجموعه فاصله‌های برابر است با:

$$2 + 12 + 4 + 4 + 36 + 2 = 60$$


و چون هر حرکت ۶ میلی ثانیه طول می‌کشد، پس در کل 60×6 میلی ثانیه طول خواهد کشید. ■


مثال

فرض کنید در سیستمی، مدیریت دیسک از زمانبندی SSTF استفاده کند. در صورتی که جابجایی بین هر دو شیار مجاور زمانی ثابت (4ms) طول بکشد و نوک خواندن - نوشتن روی شیار 40 قرار داشته باشد، زمان جابجایی بین شیارها برای سرویس دهی به درخواستهای زیر چند میلی ثانیه است؟ ترتیب درخواستها برای شیارها (از راست به چپ): 41, 44, 7, 14, 5, 35, 55, 97, 100 است.

40 $\xrightarrow{1}$ 41 $\xrightarrow{3}$ 44 $\xrightarrow{9}$ 35 $\xrightarrow{20}$ 55 $\xrightarrow{41}$ 14 $\xrightarrow{7}$ 7 $\xrightarrow{2}$ 5 $\xrightarrow{92}$ 97 $\xrightarrow{3}$ 100

که با جمع اعداد روی فلش‌ها، و ضرب در 4 حاصل 712 می‌شود. ■

عدالت فقط در روش FCFS رعایت می‌شود. 


روشهای FCFS، Scan و C-Scan، بدون قحطی هستند. 

کارایی (میانگین زمان جستجو) در FCFS پایین است. 

مراجعات در FCFS محلی می‌باشد. 

حداکثر زمان پاسخ در روش C-Scan نسبت به Scan کاهش داشته است. 

الگوریتمی به نام N-Setp_Scan وجود دارد که در آن از چند صف با طول N استفاده شده و به درخواست‌های هر صف به روش Scan پاسخ داده می‌شود. در زمانی که صفی پردازش می‌شود، درخواست‌های جدید به صف‌های دیگر وارد می‌شوند.

الگوریتم N-Setp_Scan با دو صف را الگوریتم F-Scan می‌گویند. 

یکی از تکنیکهای کاهش زمان درنگ دوران، استفاده از روش تداخل بلاکها (Interleaving) می‌باشد. در این تکنیک، بلاکها به صورت n در میان روی شیار چیده می‌شوند.

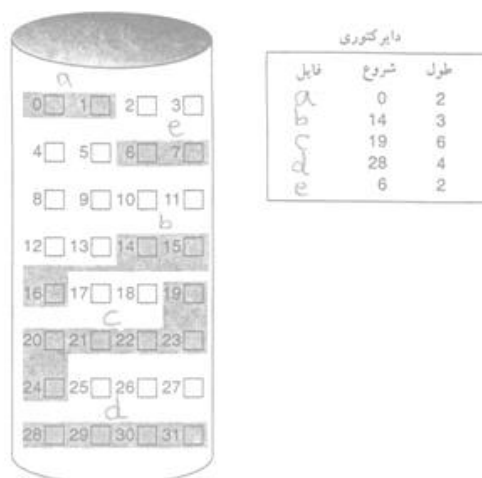
روش‌های تخصیص فضای دیسک به فایل

فضای دیسک به سه روش "پیوسته"، "پیوندی و شاخصی" می‌تواند به فایل تخصیص داده شود. هدف روش‌های مختلف، رسیدن به حالتی است که هم از فضای دیسک به خوبی استفاده شود و هم دستیابی به فایل به سرعت صورت گیرد.

۱- تخصیص پیوسته

در این روش هر فایل تعدادی بلاک پیوسته روی دیسک را اشغال می‌کند و کافی است که شماره بلاک اول روی دیسک و تعداد بلاک‌های فایل را ذخیره کرد.

شکل زیر این نوع تخصیص را نشان می‌دهد:



۲- تخصیص پیوندی

در این روش هر فایل، یک لیست پیوندی از بلاک‌های روی دیسک است. بلاک‌ها ممکن است در هر کجای دیسک پراکنده باشند. در فهرست راهنما برای هر فایل اشاره‌گری به اولین بلاک فایل قرار دارد.

۳- تخصیص شاخصی

در روش شاخصی، اشاره‌گرها به بلاک‌های فایل روی دیسک در یک مکان که به آن بلاک شاخص می‌گویند، جمع‌آوری می‌شوند. هر فایل دارای بلاک شاخص خود است که یک ماتریس از آدرس‌های بلاک‌ها است. ورودی Iام در بلاک شاخص به بلاک Iام فایل اشاره می‌کند.

در فهرست راهنما تنها آدرس بلاک شاخص حفظ می‌شود. در این روش براحتی می‌توان دستیابی مستقیم را حمایت کرد. البته فضای بلاک شاخص تلف می‌شود و در بسیاری از مواقع به تمامی بلاک شاخص نیاز نمی‌باشد. علت پر طرفدار بودن روش شاخصی، رابطه نزدیک آن با مدیریت حافظه قطعه‌بندی-صفحه‌بندی شده است. بلاک شاخص می‌تواند یک جدول صفحه‌بندی و بلاک‌های فایل همانا صفحات فایل (در واقع یک قطعه) هستند.

مشکلات تخصیص پیوسته:

۱- یافتن فضای خالی برای یک فایل جدید. (برای یک فایل n بلاکی باید به دنبال n بلاک آزاد پشت سرهم گشت.)

۲- تعیین مقدار فضای مورد نیاز یک فایل

مشکلات تخصیص پیوندی

۱- عدم حمایت از دستیابی مستقیم

این روش فقط در رابطه با دستیابی ترتیبی خوب عمل می‌کند. زیرا برای رسیدن به بلاک i ، باید بلاکهای قبل از آن دستیابی شوند که هر کدام از اینها به یک خواندن از دیسک نیاز دارد.

۲- اتلاف فضا توسط پیوندها

۳- عدم قابلیت اطمینان سیستم

با از بین رفتن تنها یکی از اشاره گرها، صدمات جدی به فایل و فضای روی دیسک وارد می‌گردد.

دستیابی به فایل در تخصیص پیوسته

۱- ترتیبی

سیستم فایل برای دستیابی ترتیبی، شماره بلاک روی دیسک، آخرین بلاک فایل که مورد دستیابی قرار گرفته را به خاطر می‌سپارد و در صورت لزوم سراغ بلاکهای بعدی می‌رود.

۲- مستقیم

برای دستیابی مستقیم به بلاک i فایل، کافی است که به بلاک $b+i$ دستیابی صورت گیرد. (با این فرض که فایل از بلاک b دیسک آغاز شده باشد).

مثال

فایلی دارای 5 بلوک از شماره 1 تا 5 می‌باشد. می‌خواهیم بلوک شماره 4 را حذف کنیم. تعداد کل نقل و انتقال دیسک در سه حالت تخصیص دیسک به فایل را مشخص کنید. (در ابتدا فایل باز است).

حل: الف- پیوسته: چون همه بلوک های فایل به صورت پشت سر هم قرار دارند، بلوک 5 خوانده شده و بر روی بلوک 4 نوشته می‌شود. بنابراین به 2 دسترسی دیسک نیاز است.

ب- پیوندی: چون هر بلوک فایل حاوی اشاره گری است که آدرس بلوک بعدی را مشخص می‌کند، از بلوک 1 تا 4 خوانده تا آدرس بلوک 5 را به دست آوریم. سپس این آدرس را جایگزین آدرس بلوک 3 می‌کنیم. بنابراین به 5 دسترسی دیسک نیاز است.

ج- شاخصی (اندیسی): چون آدرس همه بلوک ها در یک جدول شاخص بر روی دیسک ذخیره شده، بلوک حاوی اندیسی ها را خوانده و آدرس بلوک 4 را حذف کرده و بعد از به روز رسانی، بر روی دیسک باز نویسی می‌کنیم. بنابراین به 2 دسترسی دیسک نیاز دارد. ■

سطوح در یک حافظه سه سطحی

سطوح در یک حافظه سه سطحی عبارتند از:

۱- دیسک

۲- حافظه بزرگ (وسیع ولی آهسته)

۳- حافظه کوچک (سریع ولی با گنجایش محدود)

مثال

اگر اندازه صفحه حافظه بزرگ 4 کیلو بایت و اندازه صفحه حافظه کوچک 1 کیلو بایت باشد و زمان انتقال یک کیلو بایت برابر 0.5 میلی ثانیه باشد، آنگاه:

الف- زمان انتقال 4 صفحه یک کیلو بایتی متوالی از دیسک به حافظه کوچک چند میلی ثانیه است؟

(چهار انتقال از حافظه بزرگ به حافظه کوچک) + (یک انتقال از دیسک به حافظه بزرگ) =

$$= (5 + 2) + (4 \times 0.5) = 9\text{ms}$$

ب- اگر مستقیماً از دیسک به حافظه کوچک منتقل می‌کردیم و با توجه به اینکه اندازه صفحه باید یک

کیلو بایت باشد، زمان مورد نیاز چند میلی ثانیه خواهد بود؟ (زمان درنگ دورانی = 5 میلی ثانیه)

$$4 \times (5 + 0.5) = 22\text{ms}$$

■

کنکور ارشد

(مهندسی کامپیوتر - آزاد ۹۰)

۱- دیسکی با سرعت چرخش 12000 دور در دقیقه و متوسط زمان جستجوی 8 میلی ثانیه را در نظر بگیرید. در هر شیار 256 سکتور و در هر سکتور 512 بایت وجود دارد. فایلی به اندازه 1 مگا بایت بر روی شیارها و سکتورهای پراکنده و تصادفی ذخیره شده است. کل زمان دسترسی به این فایل چند ثانیه است؟

21.5 (۱) 68 (۲) 50 (۳) 55 (۴)

پاسخ: جواب گزینه ۱ است.

زمان میانگین تاخیر چرخشی (r):

$$2r = \frac{60000}{\text{RPM}} \Rightarrow 2r = \frac{60000}{12000} \Rightarrow r = 2.5 \text{ msec}$$

تعداد سکتورهای مورد نیاز برای ذخیره فایل:

$$\frac{1\text{MB}}{512} = 2048$$

زمان خواندن یک سکتور: (زمان خواندن یک شیار (2r) تقسیم بر تعداد سکتور در هر شیار)

$$\frac{5}{256} = 0.02 \text{ msec}$$

در نهایت زمان خواندن یک فایل با n سکتور به صورت تصادفی برابر است با:

$$2048 \times (8 + 2.5 + 0.02) = 21.5 \text{ sec}$$

(مهندسی کامپیوتر - دولتی ۹۰)

۲- یک دیسک را در نظر بگیرید که شامل 100 سیلندر است (0 تا 99). زمان لازم برای عبور هد از یک سیلندر به سیلندر مجاور یک واحد زمانی است. در زمان صفر هد بر روی سیلندر صفر است و درخواستی از گذشته وجود ندارد. شش درخواست در زمان های مختلف مطابق جدول زیر وارد می شوند.

90	80	70	20	10	0	زمان ورود درخواست
17	2	68	16	75	21	سیلندر درخواست شده

در زمان حرکت هد به سمت یک سیلندر، ورود درخواست جدید تاثیری بر حرکت ندارد. ترتیب اجرای درخواست ها برای الگوریتم SCAN (آسانسور) چیست؟

68 2 17 16 75 21 0 (۲)

68 17 2 16 75 21 0 (۱)

17 2 16 68 75 21 0 (۴)

17 2 68 16 75 21 0 (۳)

پاسخ: جواب گزینه ۴ است.

نحوه پاسخ به درخواست‌ها برابر است با:

0 → 21 → 75 → 68 → 16 → 2 → 17

در لحظه صفر هد بر روی سیلندر صفر قرار دارد و تنها درخواست موجود در این لحظه، درخواست برای سیلندر 21 می باشد. بنابراین هد از سیلندر 0 به سمت سیلندر 21 می رود. در ثانیه 21، درخواست برای سیلندر 75 و 16 رسیده، که بر طبق الگوریتم scan، هد به سمت سیلندر 75 می رود (حرکت به سمت بالای سیلندر 21). زمانی که هد بر روی سیلندر 75 قرار دارد، درخواست برای سیلندره‌های 68 و 16 وجود دارد که بر طبق الگوریتم scan، هد به سیلندر 68 می رود (نزدیکترین سیلندر به 75). سپس به سیلندر 16 می رود. زمانی که بر روی سیلندر 16 قرار دارد، درخواست سیلندر 17 نرسیده است، بنابراین هد به سیلندر 2 می رود. در نهایت هد به سیلندر 17 می رود.

(مهندسی IT - دولتی ۸۳)

۳- به نظر شما کدام الگوریتم زمانبندی دیسک سخت، کارایی یک RAM Disk را بهینه تر می کند؟

(۱) FIFO

(۲) تمام الگوریتم‌ها منجر به کارایی یکسانی می شوند.

(۳) Elevator/SCAN

(۴) SSTF

پاسخ: جواب گزینه ۲ است.

در RAM Disk که از RAM برای شبیه سازی دیسک استفاده می شود، به علت نداشتن حرکت مکانیکی، زمان جستجو صفر است و بهینه تر کردن کارایی بی معنی است.

دسته بندی موضوعی آموزش های فرادرس، در ادامه آمده است:

 <p>مهندسی برق الکترونیک و رباتیک</p>	 <p>هوش مصنوعی</p>	 <p>آموزش‌های دانشگاهی و تخصصی</p>	 <p>برنامه‌نویسی</p>
<p>مهندسی برق الکترونیک و رباتیک - کلیک (+)</p>	<p>هوش مصنوعی - کلیک (+)</p>	<p>آموزش‌های دانشگاهی و تخصصی - کلیک (+)</p>	<p>برنامه‌نویسی - کلیک (+)</p>
 <p>مهارت‌های دانشگاهی</p>	 <p>مباحث مشترک</p>	 <p>دروس دانشگاهی</p>	 <p>مهندسی نرم‌افزار</p>
<p>مهارت‌های دانشگاهی - کلیک (+)</p>	<p>مباحث مشترک - کلیک (+)</p>	<p>دروس دانشگاهی - کلیک (+)</p>	<p>مهندسی نرم‌افزار - کلیک (+)</p>
 <p>آموزش‌های عمومی</p>	 <p>طراحی و توسعه وب</p>	 <p>نرم‌افزارهای عمومی</p>	 <p>نرم‌افزارهای تخصصی</p>
<p>آموزش‌های عمومی - کلیک (+)</p>	<p>طراحی و توسعه وب - کلیک (+)</p>	<p>نرم‌افزارهای عمومی - کلیک (+)</p>	<p>نرم‌افزارهای تخصصی - کلیک (+)</p>

منبع مطالعاتی تکمیلی مرتبط با این کتاب

آموزش ویدئویی سیستم‌های عامل

سیستم عامل یا سامانه عامل (Operating System) بدون شک مهمترین نرم افزار در کامپیوتر است. سیستم های عامل از اولین نرم افزارهایی است که پس از روشن کردن کامپیوتر مشاهده می شود، و همچنین آخرین نرم افزاری خواهد بود که قبل از خاموش کردن کامپیوتر مشاهده می شود.

سیستم عامل نرم افزاری است که مدیریت برنامه ها را به عهده گرفته و با کنترل، مدیریت و سازماندهی منابع سخت افزاری امکان استفاده بهینه و هدفمند آن ها را فراهم کرده و بستری را برای اجرای نرم افزارهای کاربردی فراهم می کند.

اگر کامپیوتری دارای عملیات محدود و ساده باشد به وجود سیستم عامل نیازی نخواهد بود. در این گونه سیستم ها اطلاعات ورودی و خروجی از طریق صفحه کلید، نمایشگر و ... در اختیار سیستم قرار می گیرد و لذا در این سیستم ها همواره یک برنامه در تمام حالت اجرا خواهد شد.

مدرس: مهندس فرشید شیرافکن

مدت زمان: ۱۱ ساعت

جهت مشاهده آموزش ویدئویی این آموزش [کلیک](#) کنید.

faradars.org/fvsft103