سیستم عامل

مؤلف : فرشید شیرافکن

دانشجوی دکترای بیوانفورماتیک دانشگاه تهران

ناشر: سازمان علمی آموزش فرادرس

بزرگترین پلتفرم آموزش آنلاین ایران

وب: www.faradars.org

تقدیم به:

روح پاک پدرم

- فرشید شیرافکن

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 سؤالات باشد؛ ارائه نشده است. همین موضوع، کنکور را به ویژه کنکور کارشناسی ارشد به عنوان یک آزمون متمرکز و سراسری، از اهمیت دوچندانی برخوردار می‌کند.

یکی از آسیب‌های همراه با این آزمون سراسری این است که فضای رقابتی آن با ایجاد مؤسسات 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 گام های دیگر خود را در راستای آرمان فرادرس برداشتیم. کتاب حاضر که حاصل نزدیک به یک 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 که 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، می توانند 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.

سازمان علمی آموزش فرادرس

بزرگترین پلتفرم آموزش آنلاین ایران

وب: www.faradars.org

منبع مطالعاتی تکمیلی مرتبط با این کتاب

آموزش سیستم‌های عامل

سیستم عامل یا سامانه عامل (Operating System) بدون شک مهمترین نرم‌افزار در کامپیوتر است. سیستم عامل اولین نرم‌افزاری است که پس از روشن کردن کامپیوتر مشاهده می‌شود و همچنین آخرین نرم‌افزاری خواهد بود که قبل از خاموش کردن کامپیوتر مشاهده می‌شود. سیستم عامل نرم‌افزاری است که مدیریت برنامه‌ها را به عهده گرفته و با کنترل، مدیریت و سازماندهی منابع سخت‌افزاری امکان استفاده بهینه و هدفمند آنها را فراهم کرده و بستری را برای اجرای نرم‌افزارهای کاربردی فراهم می‌کند.

آموزش سیستم عامل، توسط مهندس فرشید شیرافکن، یکی از بهترین مدرسین مسلط به این مباحث، ارائه شده است.

مدرس: مهندس فرشید شیر افکن

مدت زمان: ۱۱ ساعت

[جهت مشاهده آموزش ویدئویی این آموزش – کلیک کنید](#) **faradars.org /fvsft103**

درباره مدرس

مهندس فرشید شیرافکن کارشناس ارشد مهندسی کامپیوتر گرایش نرم‌افزار است و در حال حاضر دانشجوی دکترای بیوانفورماتیک دانشگاه تهران هستند. ایشان از مدرسین نمونه در زمینه ارائه و آموزش دروس دانشگاهی انتخاب شده‌اند.

ایشان مشاور کنکور هستند و بیش از ۳۰ کتاب در زمینه کنکور رشته کامپیوتر تألیف نموده‌اند. 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 کرده‌اند. این مجموعه دروس تا کنون مورد استفاده ده‌ها هزار دانشجوی سراسر کشور قرار گرفته اند.

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

کتب رایگان دیگر از این مجموعه آموزشی

۱. آموزش برنامه نویسی C++ - کلیک کنید (+)
۲. آموزش شیء گرایی در سی پلاس پلاس – کلیک کنید (+)
۳. آموزش نظریه زبان‌ها و ماشین – کلیک کنید (+)
۴. آموزش پایگاه داده‌ها – کلیک کنید (+)
۵. آموزش ساختمان داده‌ها – 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

دسته‌بندی موضوعی آموزش‌های فرادرس، در ادامه آمده است:

			
برنامه نویسی – کلیک (+)	آموزش های دانشگاهی و تخصصی – کلیک (+)	هوش مصنوعی و یادگیری ماشین – کلیک (+)	مهندسی برق الکترونیک و روباتیک – کلیک (+)
			
دروس دانشگاهی – کلیک (+)	مباحث مشترک – کلیک (+)	مهارت های دانشگاهی – کلیک (+)	نرم افزارهای تخصصی – کلیک (+)
			
مهندسی نرم افزار – کلیک (+)	نرم افزارهای عمومی – کلیک (+)	طراحی و توسعه وب – کلیک (+)	آموزش های عمومی – کلیک (+)

فهرست مطالب

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

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 درحال اجرا می باشد. تفاوت برنامه و فرایند در این است که برنامه یک نهاد غیرفعال و فرایند یک نهاد فعال می باشد. معمولا 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 می‌شود، کار CPU متوقف شده و روال پاسخگو به وقفه (interrupt routine) اجرا می‌شود. بعد از پایان اجرای این روتین، CPU کار قبلی‌اش را ادامه می‌دهد. 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 درجریان است، مشغول اجرای دستور العملهای 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 می کند. همچنین نوع وقفه را نیز تعیین می کند.
- بردار وقفه، جدولی از اشاره گرها می باشد که هر اشاره گر به یک روال وقفه اشاره می کند.

انواع وقفه‌ها

وقفه ها بر چهار نوع می باشند:

۱- برنامه

وقفه هایی که به دلیل بعضی شرایط حاصل از اجرای یک دستورالعمل بروز می کند:

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

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

۳- ورودی/خروجی

وقفه هایی که به وسیله کنترل کننده I/O تولید می شود، تا کامل شدن طبیعی یک عمل یا شرایط خطا را اعمال کند.

۴- نقص سخت افزار

وقفه هایی که با نقص سخت افزاری تولید می شود، مثل نقص برق یا خطای توازن حافظه.

پردازش وقفه

هنگامی که یک دستگاه I/O ، یک عمل I/O را کامل می کند، دنباله حوادث زیر اتفاق می افتد:

۱- دستگاه یک علامت وقفه برای پردازنده می فرستد.

۲- پردازنده اجرای دستورالعمل جاری را قبل از پاسخ به این وقفه، به پایان می رساند.

۳- پردازنده بروز وقفه را بررسی کرده، در می یابد که وقفه ای آمده است و علامتی مبنی بر دریافت وقفه برای دستگاه وقفه دهنده می فرستد.

۴- پردازنده برای انتقال کنترل به روال خدماتی مربوط آماده می شود. در ابتدا اطلاعات مورد نیاز برای از سرگیری برنامه جاری را ذخیره می کند. حداقل اطلاعات، محتویات ثبات وضعیت برنامه(PSW) و محل دستورالعمل بعدی(PC) می باشد. این اطلاعات می تواند 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 می کند. در این حالت پردازنده به چرخه واکشی رفته و کنترل به برنامه گرداننده وقفه منتقل می شود.

۶- در این لحظه محتوای PC و PSW مربوط به برنامه وقفه داده شده، در پشته سیستم ذخیره است.

۷- روال وقفه را پردازش می کند.

۸- بعد از کامل شدن پردازش وقفه، مقادیر ثباتها از پشته POP شده و در ثباتها گذاشته می شود.

۹- بار کردن مجدد مقادیر PSW وPC از پشته، که موجب اجرای دستورالعمل بعدی از برنامه وقفه داده شده می شود.

تعویض متن (Context Switch)

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

فراخوانی های سیستم (System Calls)

فراخوانیهای سیستم واسطی بین فرایند و سیستم عامل فراهم می کند که به صورت دستورات زبان اسمبلی می باشند. در بعضی از سیستم ها، فراخوانی ها مستقیما از برنامه های زبان سطح بالا ساخته شده و مانند زیر برنامه می باشند. وقتی برنامه ای اجرا می شود از فراخوانی های سیستمی به وفور استفاده می

کند و کاربر جزئیات آن را نمی بیند.

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، احتمالا 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)

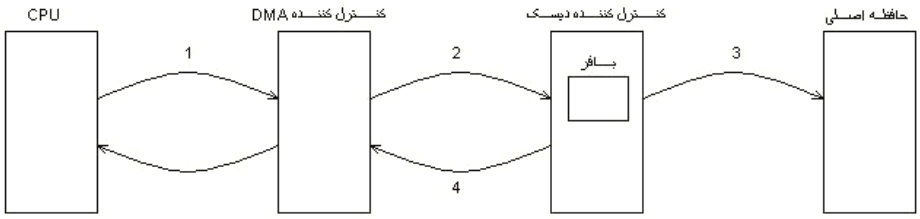
پردازنده یک فرمان I/O را از جانب فرایندی به یک مولفه I/O صادر می کند. سپس آن فرایند قبل از ادامه، تا کامل شدن عمل I/O ، به انتظار مشغولی می گذراند.

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 داده ها بین حافظه اصلی و مولفه I/O را کنترل می کند. پردازنده تقاضایی برای انتقال یک بلوک از داده ها را به مولفه DMA می فرستد و فقط پس از کنترل کل بلوک، مورد وقفه قرار می گیرد.



تذکر: در روش ورودی/خروجی مبتنی بر وقفه، دستورالعملهای بعدی می توانند 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 می گردد و کار دیگری انجام می گیرد.

- روش دوم کارامدتر از روش اول است.
- برای انتقال I/O چند کلمه ای، روش DMA بسیار کارآمدتر از روشهای اول و دوم است.
- در روش اول، مولفه I/O به پردازنده وقفه نمی دهد و این مسئولیت پردازنده است که وضعیت مولفه I/O را متناوبا بررسی کند تا اتمام آن عمل را دریابد. بنابراین روش اول را می توان ورودی/خروجی بر اساس سرکشی(polling) نامید.

- مشکل ورودی/خروجی برنامه سازی شده این است که پردازنده باید برای مدت طولانی منتظر بماند تا مولفهI/O برای دریافت یا ارسال آماده شود. در مدت انتظار، پردازنده به علت اینکه مکررا وضعیت را سئوال می کند، کارایی سیستم به شدت پایین می آید.
- روش I/O Programed و I/O Interrupt پردازنده اصلی را درگیر عملیات I/O می کند.

نگاهی کلی به سیستم عامل

سیستم عامل برنامه ای است که اجرای برنامه های کاربردی را کنترل و به صورت رابط کاربر و سخت افـزار 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 کار آمد 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 پردازنـده را بـرای اسـتفاده از سـایر 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 زیـر) 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، ادیتورهـا و ..)، بخشـی از سیسـتم عامـل 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 درحافظه ثانویه و تهیه پشتیبان.

۵- مدیریت سیستم I/O

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

تاریخچه سیستم های عامل

نسل اول: سیستم های ردیفی

در اولین کامپیوترها(حدودا ۶۰ سال قبل)، برنامه ساز مستقیما با سخت افـزار در تـراکنش بـود و سیسـتم 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 ایـن عملیـات، کـاربران بـه صـورت ردیفی و یکی پس از دیگری به کامپیوتر دسترسی داشتند، آن را پردازش ردیفی می خوانیم.

نسل دوم : سیستم های دسته‌ای ساده

زمانی که به خاطر زمانبندی و نصب در سیستمهای ردیفی به هدر می رفت قابل قبول نبود. بـرای افـزایش 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 اسـتفاده از سیسـتم عامـل دسـته ای، دیگر کاربر دسترسی مستقیم به ماشین ندارد و کار خود را روی کارت یا نوار به متصدی کـامپیوتر مـی دهد. اپراتور کارها را به صورت ردیفی دسته کرده و همگی را روی یک دستگاه ورودی می گـذارد تـا مـورد 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 برنامـه نویسی برای فرمان دادن به ناظر می باشد.

JCL : Job Control Language

اسپولینگ

در سیستم های اولیه،CPU گرانترین جزء کامپیوتر بود و افزایش راندمان CPU مهمترین محرک در تکامـل سیستم های عامل بوده است. برای همین منظور، تکنیک Spooling مطرح شد. اسپولینگ I/O یک کـار را با محاسبات کار دیگر به طور همزمان انجام می‌دهد. حتی می‌تواند در حین خواندن ورودی‌هـای یـک کـار، خروجی های کار دیگر را انجام دهد. اسـپولینگ موجـب مـی شـود تـا CPU و دسـتگاههایI/O بـا سـرعت بیشتری کار کنند. استفاده از spooling در سیستم عامل دسته ای باعث افزایش سرعت می شود.

سیستم های دسته ای Offline Spooling

در این سیستم ها، 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 توسـط

اپراتور به CPU اصلی منتقل شده تا اجرا شروع شود. نتیجه اجرا بر روی نوار دیگری نوشته می شـد. بعـد از اجرای تمام دسته کارها، اپراتور نوار خروجی را به یک پردازنده آهسته منتقل می کرد تا عمل چـاپ انجـام شود. در این روش ارتباط پردازنده اصلی محاسباتی با کارت خوان و چاپگر، غیر مستقیم بود.

فواید سیستم های دسته ای offline Spooling نسبت به سیستم های قبلی

۱- افزایش راندمان پردازنده گران قیمت

۲- عملیات ساده‌تر

۳- ساده شدن استفاده از سیستم از راه دور.

معایب سیستم های 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

سیستم های دسته ای Online Spooling

با ظهور دیسک های سخت، سیستم های دسته ای متحول شدند. در این سیستم ها بلافاصـله بعـد از ورود کارها، کارت ها خوانده شده و به دیسک منتقل می شدند. زمانی که اجرای یک کار تمام می شود، سیسـتم عامل با سرعت بیشتری یک کار جدید را از روی دیسک برداشته و به حافظه اصلی لود کرده و سپس آن را اجرا می کند.

در خروجی نیز از اسپولینگ استفاده می شود. اسناد خروجی ابتدا بر روی دیسک ذخیره شده و زمـانی کـه خروجی قبلی چاپگر تمام شد، این اسناد از دیسک به بافر چاپگر منتقل می شوند.

در این تکنیک که Spooling (عملیات پیوسته، مستقیم و همزمان دستگاه های جانبی) نامیـده مـی شـود، ارتباط CPU با دستگاه های جانبی مستقیم است و نیازی به پردازنده های آهسته و نوار گردان های اضافی و حمل نوارها توسط اپراتورها نمی باشد.

SPOOL : **S**imulataneonus **P**eripheral **O**peration **O**n-**L**ine

مزایای اسپولینگ online نسبت به offline عبارتند از :

۱- دسترسی با اولویت

۲- گردش سریع کارها

۳- بالا بودن راندمان CPU ودستگاههای I/O

۴- امکان داشتن همزمان چند مدرک ورودی/خروجی

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

نرم افزار spooler کارها را دسته بندی می کند. این نرم افزار اجزاء یک کار(داده، برنامه وJCL) 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 یکJob می باشد.

نسل سوم: سیستم های دسته‌ای چند برنامگی

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 ازآنها را انتخاب می‌کند و CPU را به آن تخصیص می دهد. (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 برنامه ها ایستا می باشد)

- ممکن است حافظه را برای نگهداری ۳،۴ یا تعداد بیشتری از برنامه ها گسترش دهیم و پردازنده به تمام آنها بپردازد. این عمل را چند برنامه ای یا چند وظیفه ای می گویند که موضوع اصلی سیستمهای عامل امروزی است.

نسل چهارم : سیستم های اشتراک زمانی (Time sharing)

سیستم های اشتراک زمانی(چندبرنامگی تعاملی) توسعه منطقی سیستم های چند برنامه‌ای می‌باشند. در سیستم عامل اشتراک زمانی، وقت پردازنده بین کارها به اشتراک گذاشته می شود. برای اجرای چند کار، پردازنده بین آنها سوئیچ می‌کند و کاربران می‌توانند با برنامه‌های در حال اجرا تعامل داشته باشند. در

سیستم های اشتراک زمانی ارتباط کاربر با سیستم به صورت محاوره ای (interactive یا online) است.
سیستم های اشتراک زمانی، هزینه استفاده محاوره‌ای از سیستم های 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 سیستم های 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 برنامگی را فراهم می سازد.
۴- وجود یک سیستم فایل ضروری است.
۵- در main frame قابلیت چند کاربره و چند برنامگی را فراهم می کند.

چندبرنامه ای دسته ای و اشتراک زمانی هر دو از چندبرنامگی استفاده می کنند.

تفاوتهای اصلی چندبرنامه ای دسته ای با اشتراک زمانی

	چندبرنامه ای دسته ای	اشتراک زمانی
هدف اصلی	حداکثر استفاده از پردازنده	حداقل زمان پاسخ
منبع دستورات به سیستم عامل	دستورالعملهای JCL که همراه کار ارائه شده است.	فرمانهایی که از پایانه وارد می شود.

تذکر: سیستم های زیر در کتاب سیلبرشاتس آورده شده است:

۱- سیستم های موازی (یا چند پردازنده ای)

۲- سیستم های عامل توزیع شده (گسترده)

۳- سیستم های عامل بی‌درنگ (Real time)

سیستم های موازی (یا چند پردازنده ای)

سیستم هایی 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 سیستم های چند پردازنده‌ای

الف- چندپردازنده‌ای متقارن

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

ب- چندپردازنده‌ای 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 سیستم های سخت افزاری مختلف

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

دلایل ساخت سیستم های 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 نمی شود بلکه از سرعت آن می کاهد.)

۳- مقرون به صرفه بودن از نظر اقتصادی

سیستم های عامل توزیع شده (گسترده)

در این سیستم ها، 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. سیستم های توزیعی را سیستم های ارتباط ضعیف (Loselycoupled) نیز می‌گویند.

دلایل ساخت سیستم های توزیعی

الف- اشتراک منابع : کاربری دریک 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

سیستم های عامل بی‌درنگ (Real time)

سیستم عامل بی‌درنگ(بلادرنگ)، نوعی ازسیستم 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.

سیستم های نظامی، تزریق سوخت اتومبیل، کنترل کننده‌های لوازم خانگی،کنترل صنعتی و تصویرسازی پزشکی نمونه هایی از سیستم های بی درنگ می باشند. سیستم های بی درنگ بر دو نوع نرم و سخت می باشند.

ویژگی های سیستم های بی درنگ نرم (Soft Real Time)

۱- دارای محدودیت زمانی نسبتا دقیقی می باشند.

۲- در پروژه های علمی پیشرفته استفاده می‌شوند.

۳- با سیستم های دیگر قابل ترکیب می باشند.

۴- مهلت زمانی را پشتیبانی نمی کنند. (از آنها در کنترل صنعتی و روبات‌ها استفاده نمی شود.)

ویژگی های سیستم های بی درنگ سخت (Hard Real Time)

۱- دارای محدودیت زمانی دقیقی می باشند و کارهای بحرانی به موقع انجام می‌شوند.

۲- تمام تاخیرهای موجود درسیستم باید از بین بروند.

۳- از ویژگیهای پیشرفته سیستم عامل استفاده نمی‌شود.

۴- با سیستم های اشتراک زمانی برخورد (Conflict) دارند و با یکدیگر ترکیب نمی شوند.

۵- هیچکدام از سیستم های همه منظوره موجود از عملکرد بی‌درنگ سخت پشتیبانی نمی‌کنند.

انواع سیستم عامل از نظرساختاری

انواع سیستم عامل از نظرساختاری عبارتند از:

۱- یکپارچه (Monolitic)

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 مستقیما بر روی مونیتور یا دیسک بنویسند.

ساختار لایه‌ای

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 خطا یابی می شود. اگر در حین خطایابی لایه ای، خطایی یافت شود، می دانیم که خطا در همان لایه است، زیرا لایه های پایینی قبلا خطا یابی 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.

فرایند اول برای انجام محاسباتش به اندازه 200 سیکل به CPU نیاز دارد و فرایند 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 فرایند دوم می شود. البته در هنگام کار با DMA و انجام I/O ، گاهی گذرگاه در اختیار DMA قرار می گیرد و زمان اجرای فرایند اول بیشتر از 200 سیکل خواهد شد. بنابراین کل عملیات به بیش از 210 سیکل نیاز دارد.

(مهندسی کامپوتر- 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

۱) 27 ۲) 10 ۳) 20 ۴) 17

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 زمان5 تا 7 نیز job2 از ورودی دریافت می شود. اجرایjob1 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 واکشی عملوندها ۴) پس از ذخیره نتایج

پاسخ: گزینه ۳ جواب است.

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

فصل ۲

فرآیند

در این فصل مفهوم فرایند و حالات فرایند بررسی می شوند. همچنین نخ نیز توضیح داده می شود.

فرایند و حالات آن

به برنامه در حال اجرا، فرآیند(process) می گویند. یک فرایند می تواند در یکی از حالتهای زیر باشد:

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

۲- اجرا (Running) : فرایندی که هم اکنون در حال اجر می باشد.

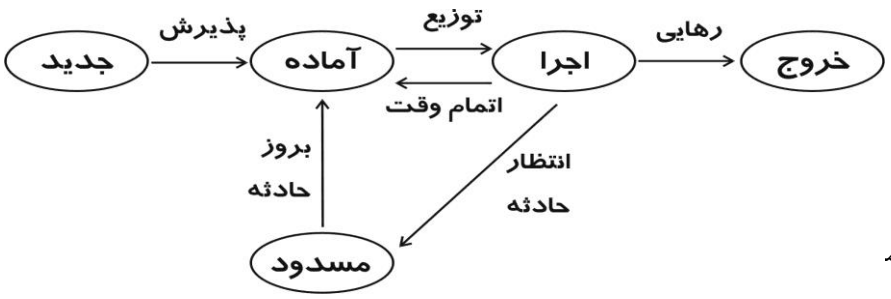
۳- مسدود (Blocked) : فرایندی که تا بروز حادثه‌ای (مثل اتمام یک عمل I/O)، نمی‌تواند اجرا شود.

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

۵- خروج (Terminated) : فرایندی که اجرای آن پایان یافته است و یا اجرای آن قطع شده و از مجموعه فرایندهای قابل اجرای سیستم عامل خارج شده است.

تذکر : به حالت مسدود، حالت بسته یا انتظار نیز می گویند.

در شکل زیر این حالات، نشان داده شده است:



تغییر حالات ممکن

۱- جدید به آماده

اگر سیستم عامل آمادگی گرفتن یک فرایند دیگر را داشته باشد، فرایند 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) می گویند.

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 بخواهد که به خاطر آن باید منتظر بماند، سیستم عامل آن فرایند را از حالت اجرا به مسدود می‌برد. به این عمل بلوکه شدن می گویند. 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 می گویند.

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 می باشد.

۷- آماده به خروج

در بعضی سیستمها، یک پدر می تواند 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 می کند.

وجود وضعیت مسدود باعث افزایش بهره وری پردازنده می شود. چون وقتی فرایند در حال اجرا نیاز بـه I/O پیـدا مـی کند، به حالت Blocking منتقل شده و فرایند آماده دیگری به قسمت اجرا منتقل مـی شـود، تـا در حـد امکـان 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 محاوره ای

کاربر از طریق پایانه با سیستم ارتباط برقرار می کند.

۲- ارائه سرویس به وسیله سیستم عامل

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

۳- زایش توسط یک فرایند موجود

به منظور بهره گیری از توازی با تفکیک، برنامه کاربر می تواند ایجاد فرایندهای جدیدی را دیکته کند.

۴-کار دسته ای جدید

سیستم عامل با جریانی از کارهای دسته ای روبه رو می باشد. وقتی برای گرفتن یک کار جدید آماده است، دنباله بعدی از فرمانهای کنترل کار را می خواند.

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 داده های ممکن دیگر.

دلایل پایان یک فرایند

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

بلوک کنترل فرایند (PCB)

هر فرایند در سیستم بوسیله یک ساختاری به نام بلوک کنترل فرایند (Process Control Bolock) مشخص می‌شود. این بلوک مهمترین و محوریترین 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، محدویتهای 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 مضنون

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 می‌برد.

نمودار تغییرحالت 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.

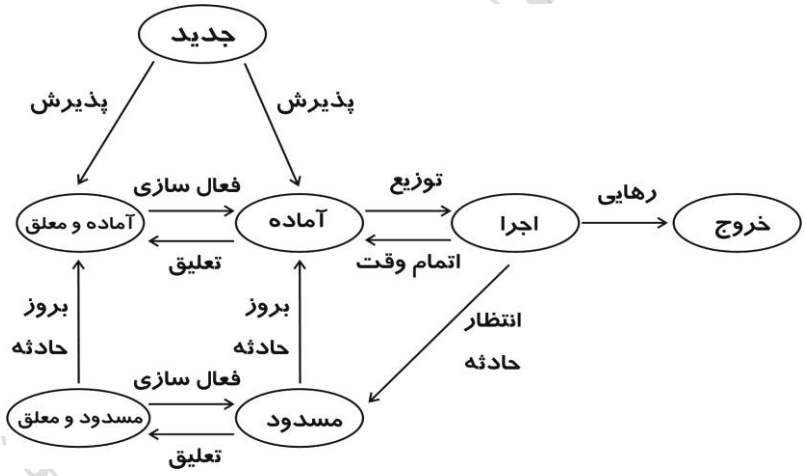
بنابراین حالت معلق را به حالتهای گفته شده، اضافه می کنیم:

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 اینحالت 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 برنامگی 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)

تصمیم گیری در مورد افزودن به مجموعه فرایندها برای اجرا.

۲- زمانبند میان مدت(Middle Term Scheduler)

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

۳- **زمانبند کوتاه مدت** (Short Term Scheduler)

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

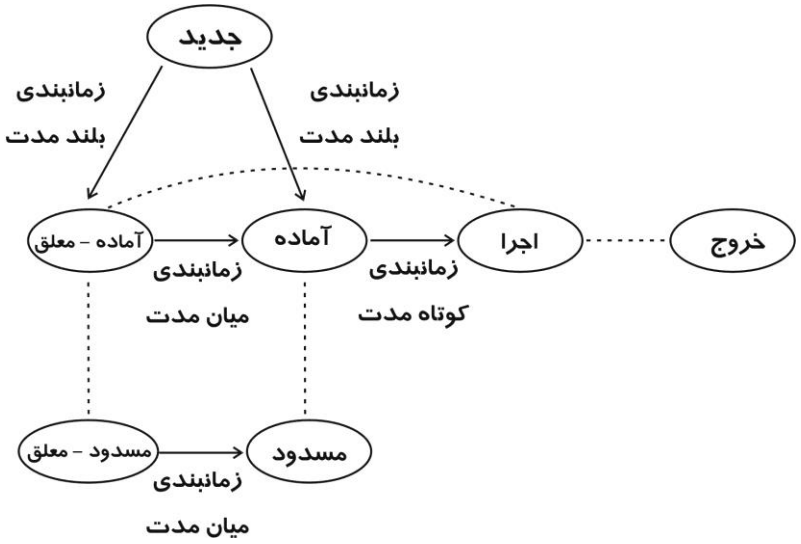
۴- زمانبندی ورودی/خروجی

تصمیم می گیرد که کدام درخواست I/O فرایندها به وسیله یک دستگاه I/O موجود انجام بگیرد.

- وظیفه فعال سازی و تعلیق فرآیندها بر عهده زمانبند میان مدت(Medium-term scheduler) می باشد.
- زمانبند میان مدت، فرایندی را از حافظه اصلی حذف و به حافظه جانبی می برد. این فرایند بعدا می تواند به حافظه اصلی لود شود. این الگو را مبادله (swapping) می گویند.
- ایده اصلی زمانبندی میان مدت، این است که می تواند فرایندی را از حافظه حذف کند و درجه چند برنامگی را کاهش دهد.
- زمانبند بلند مدت، ترکیب خوبی از فرایندهای I/O limited و CPU Limited ، انتخاب می کند.
- 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.

تذکر: اکثر سیستم های اشتراک زمانی فاقد زمانبند بلند مدت می باشند.

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 مستقلا 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 نخی، هر گاه فراخوان

سیستمی مسدود کننده ای اجرا شود، یا یک نقص صفحه رخ دهد، کل فرایند مسدود می شود. بـه عنـوان مثال یک برنامه صفحه گسترده را در نظر بگیرید و فرض کنید کاربری می خواهد به طور دائمـی و تعـاملی مقادیر را تغییر دهد. می دانیم که در برنامه صفحه گسترده، وابستگی تابعی بین سلول هـای مختلـف بایـد حفظ شود. بنابراین اگر سلولی تغییر کند، محاسبات زیادی انجام می گیرد. اگر فقط یک نـخ وجـود داشـته باشد، وقتی برنامه منتظر دریافت ورودی است، محاسبات نمی تواند ادامه یابـد، چـون فراخـوان سیسـتمی دریافت ورودی، کل فرایند را مسدود کرده است. راه حل این است که از یک نخ برای خواندن ورودی کـاربر و از نخ دیگری برای بهنگام سازی صفحه گسترده استفاده شود.

مثال

بسیاری از صفحات وب، شامل چند تصویر کوچک هستند که مرورگـر بایـد یـک اتصـال جداگانـه را بـرای دریافت هر تصویر برقرار کند که زمان زیادی لازم دارد. اگر در مرورگر از روش چند نخی استفاده شود، هـر 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 پـرداز را طـوری طراحی کرد که هر یک دقیقه بافر RAM را روی دیسک بنویسد. یک نـخ مـی توانـد تنهـا بـرای گـرفتن پشتیبان دوره ای ایجاد شود و خودش را مستقیما با سیستم عامل زمانبندی کند.

نخها و فرایندها می توانند ۴ حالت را بوجود آورند:

۱- یک فرایند- یک نخ

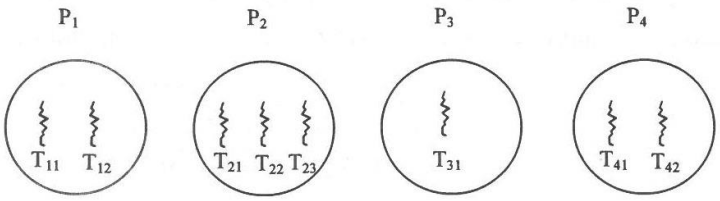
۲- یک فرایند – چند نخ

۳- چند فرایند- یک نخ در هر فرایند

۴- چند فرایند– چند نخ در هر فرایند

مثال

شکل زیر سیستمی با چهار فرآیند 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 نخ های یک فرایند، در حالت و منابع آن فرایند شریک هستند.
- اگر نخ یک فرآیند در حال اجرا باشد و آن فرآیند به حالت خروج برود، نخ نمی تواند 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 فرآیند جداگانه از نوع تعویض متن فرآیندی است.
- نخ در برنامه نویسی، بخشی از یک فرایند یا برنامه بزرگتر می باشد.
- ساختار نخ گاهی در محیط تک پردازنده ای، برای ساده کردن ساختار برنامه ای که منطقا اعمال متعددی را انجام می دهد، نیز مفید است.
- با تقسیم یک کار به چند نخ، برنامه ساز می تواند 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- نخ های سطح کاربر

۲- نخ های سطح هسته

۳- روش های ترکیبی

نخ های سطح کاربر

در این روش تمام عملیات راهبری در فضای آدرس کاربر انجام می شود. در این روش هزینه ایجاد نخ با هزینه تخصیص حافظه برای بر پا سازی پشته (stack) نخ تعیین می شود. تعویض متن نخ اغلب با تعداد کمی از دستورات العمل ها انجام می شود و فقط کافی است محتوای ثبات های پردازنده برای نخ فعلی، ذخیره شود و سپس مقادیر قبلا ذخیره شده مربوط به نخی که به آن سوئیچ می شود، بار گذاری شوند. در تعویض متن نخ ها نیازی به فلاش کردن TLB ، تغییر نگاشت های حافظه و حسابداری پردازنده نمی باشد.

مزایای کتابخانه نخ سطح کاربر

۱- ایجاد و حذف سریع نخ ها

۲- همگام سازی سریع نخ ها

۳- تعویض متن کم هزینه و سریع

نقاط ضعف نخ های سطح کاربر

۱- اگر یک نقص صفحه (page fault) برای یک نخ رخ دهد، کل فرایند و در نتیجه همه نخ های درون آن به اشتباه مسدود می شود.

۲- اگر نخی یک فراخوان سیستمی بلوکه کننده را صدا بزند، کل فرایند و همه نخ های درون آن نیز به اشتباه مسدود می شوند.

۳- چون سیستم عامل از وجود نخ ها آگاه نمی باشد، نخ ها را بین چندین پردازنده به خوبی پخش و زمان بندی نمی کند.

نخ های سطح هسته

اگر نخ ها در هسته سیستم عامل پیاده سازی شوند، مشکلات روش قبل رخ نمی دهند. اما چون هر عملیات نخ باید توسط هسته انجام شود، نیاز به یک فراخوانی سیستمی دارد و هزینه بالا می رود. تعویض متن نخ در این روش ممکن است به اندازه تعویض متن فرایند پر هزینه باشد.

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

روش ترکیبی

برای غلبه بر مشکلات دو روش قبل، نخ های سطح کاربر و سطح هسته را با تلفیق می کنیم. برای این کار فرایندهای سبک وزن(LWP) ایجاد می کنیم. 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 نخ ها به برنامه های کاربردی ارائه می دهد. بسته نخ کاملا در فضای کاربر پیاده سازی می شود و تمام عملیات روی نخ ها بدون دخالت هسته انجام می شود. برنامه های کاربردی چند نخی با ایجاد نخ ها و سپس انتساب هر نخ به یک LWP ساخته می شوند.

در این روش اگر نخی فراخوان سیستمی مسدود کننده ای را اجرا کند، اجرا از مد کاربر به مد هسته تغییر می کند ولی همچنان در متن LWP جاری ادامه می یابد. در جایی که LWP جاری نتواند ادامه دهد، متن به LWP دیگری تعویض می شود که باعث یک تعویض متن برای بازگشت به مد کاربر نیز می شود.

- در روش ترکیبی، فراخوانی های سیستمی از نوع مسدود با تامین هم روندی حمایت می شوند.

- تغییر متن (context switch) در نخ های یک فرایند، اشاره گر پشته (SP) را تغییر می دهد ولی ثبات ها و جداول مدیریت حافظه را تغییر نمی دهد و ارتباطی با TLB ندارد.

- TLB یک سخت افزار جستجوی سریع، کوچک و پنهان است. در صفحه بندی تعداد اندکی از ورودیهای جدول صفحه در TLB قرار می گیرد.

(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 درحافظه)

۱) از sleep swapped به ready to run in memory

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

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

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:

sleep swapped : مسدود و معلق

sleep in memory : مسدود

ready to run in memory : آماده

ready to run swapped : آماده و معلق

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- کدم گزینه درباره مدل های چند نخی 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 نخ ها وجود ندارد.

توضیح بیشتر: بین نخ ها و فرایندها می تواند ۴ رابطه زیر برقرار باشد:

یک به یک	UNIX	هر نخ اجرا، یک فرایند یکتا با فضای آدرسها و منابع خویش است.
چند به یک	NT , Solris,OS/2	یک فرایند، یک فضای آدرس و مالکیت پویای منابع را تعریف می کند. ممکن است نخهای متعدد در داخل آن فرایند ایجاد و اجرا گردد.
یک به چند	Clouds	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 می سازد.
چند به چند	TRIX	ترکیب خصوصیات موارد چند به یک و یک به چند

در رابطه با این مدل ها می توان گفت که:

الف- توانایی بکارگیری پردازنده ها در مدل های 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 می باشد.)

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

۲) یک نخ عادی در طول حیات خود ممکن است در LWP های متفاوتی، بخش هایی از اجرای خود را بگذراند.

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

۴) نخ مستقیما زیر نظر سیستم عامل اجرا می شود و مدیریت آن نمی تواند در سطح کاربر باشد.

پاسخ: گزینه ۲ درست است.

فرایند سبک وزن(LWP) ، نگاشتی بین نخ سطح کاربر و نخ سطح هسته می باشد. هر LWP از یک نخ سطح کاریر یـا بیشـتر حمایـت می کند و به یک نخ سطح هسته می نگارد. LWP ها مستقلا توسط هسته زمانبندی شده و ممکن است به صورت موازی روی پردازنـده های متعدد اجرا شوند.

یک نخ عادی در طول حیات خود ممکن است در LWP های متفاوتی، بخش هایی از اجرای خود را بگذراند. بنـابراین گزینـه ۲ درسـت است. گزینه ۳ نادرست است، چون تعداد LWP ها به اندازه تعداد نخ هایی است که هم اکنون با هسته درگیر هستند و سیستم عامل به ازای هر نخ، یک LWP ایجاد نمی کند. به طور نمونه اگر دو LWP موجود باشد و نخ سوم فرایند، درخواستی از هسته داشته باشد، باید منتظر بماند تا یکی از LWP های قبلی کارش با هسته پایان یابد.

منتخبی از عناوین آموزشی منتشر شده بر روی فرادرس

برنامه‌نویسی	
عنوان آموزش	مدت زمان تقریبی
مبانی برنامه نویسی - کلیک کنید (+)	۳ ساعت
برنامه نویسی C - کلیک کنید (+)	۱۳ ساعت
آموزش برنامه نویسیC++	۲۰ ساعت
برنامه نویسی کاربردی سی شارپ - کلیک کنید (+)	۱۴ ساعت
آموزش جامع شی گرایی در سی شارپ - کلیک کنید (+)	۱۴ ساعت
برنامه نویسی جاوا - کلیک کنید (+)	۲۳ ساعت
آموزش برنامه نویسی - PHP کلیک کنید (+)	۲۸ ساعت
آموزش فریمورک PHP کدایگنایتر - (CodeIgniter) کلیک کنید (+)	۷ ساعت
آموزش اسکریپت برنامه نویسی - jQuery کلیک کنید (+)	۷ ساعت
آموزش ویژوال بیسیک دات نت - (Visual Basic.NET) کلیک کنید (+)	۱۳ ساعت
آموزش تکمیلی ویژوال بیسیک دات نت - (Visual Basic.NET) کلیک کنید (+)	۱۶ ساعت
آموزش برنامه نویسی با روش سه لایه به زبان - VB.Net کلیک کنید (+)	۴ ساعت
برنامه نویسی اسمال بیسیک یا - Small Basic کلیک کنید (+)	۱۶ ساعت
آموزش ساخت بازی ساده در ویژوال بیسیک - کلیک کنید (+)	۲ ساعت
آموزش کاربردی - SQL Server کلیک کنید (+)	۱۱ ساعت
آموزش آشنایی با LINQ to SQL در - C# کلیک کنید (+)	۲ ساعت
آموزش برنامه نویسی با روش سه لایه به زبان سی شارپ - کلیک کنید (+)	۴ ساعت
آموزش برنامه نویسی تحت شبکه با سی شارپ در قالب پروژه - کلیک کنید (+)	۱ ساعت
آموزش Cryptography در دات نت - کلیک کنید (+)	۳ ساعت

برنامه‌نویسی (ادامه از صفحه قبل)	
عنوان آموزش	**مدت زمان تقریبی**
آموزش قفل نرم افزاری در سی شارپ از طریق رجیستری	۴ ساعت
آموزش ساخت اپلیکیشن کتاب و کار با داده‌ها در اندروید - کلیک کنید (+)	۱۳ ساعت
آموزش ارتباط با دیتابیس سمت سرور در اندروید - کلیک کنید (+)	۱۴ ساعت
آموزش ساخت روبات و کنترل آن با اندروید - کلیک کنید (+)	۱۶ ساعت
آموزش ساخت اپلیکیشن دیکشنری صوتی دو زبانه با قابلیت تشخیص صوت کاربر - کلیک کنید (+)	۷ ساعت
آموزش مدیریت بانک اطلاعاتی اوراکل - کلیک کنید (+)	۹ ساعت
آموزش مدیریت بانک اطلاعاتی اوراکل پیشرفته - کلیک کنید (+)	۷ ساعت
آموزش راه اندازی اوراکل ۱۲ cدر لینوکس	۱ ساعت
آموزش دیتاگارد در اوراکل - کلیک کنید (+)	۳ ساعت
برنامه نویسی متلب - کلیک کنید (+)	۹ ساعت
متلب برای علوم و مهندسی - کلیک کنید (+)	۱۴ ساعت
برنامه نویسی متلب پیشرفته - کلیک کنید (+)	۷ ساعت
طراحی رابط های گرافیکی (GUI) در متلب - کلیک کنید (+)	۸ ساعت
آموزش برنامه نویسی R و نرم‌افزار - RStudioکلیک کنید (+)	۷ ساعت
آموزش تکمیلی برنامه نویسی R و نرم‌افزار - RStudio کلیک کنید(+)	۵ ساعت
آموزش برنامه نویسی پایتون ۱ - کلیک کنید (+)	۲۰ ساعت
آموزش برنامه نویسی پایتون ۲ - کلیک کنید (+)	۵ ساعت
آموزش گرافیک کامپیوتری با - OpenGL کلیک کنید(+)	۱۶ ساعت

راه اندازی و مدیریت وبسایت‌ها و سرورها	
عنوان فرادرس	**مدت زمان تقریبی**
آموزش برنامه نویسی - PHP کلیک کنید (+)	۲۸ ساعت
آموزش فریمورک PHP کدایگنایتر - (CodeIgniter) کلیک کنید (+)	۷ ساعت
آموزش طراحی وب با - HTML کلیک کنید (+)	۳ ساعت
آموزش طراحی وب با - CSS کلیک کنید (+)	۵ ساعت
آموزش پروژه محور HTML و - CSS کلیک کنید - کلیک کنید (+)	۴ ساعت
آموزش جاوا اسکریپت - (JavaScript) کلیک کنید (+)	۹ ساعت
آموزش کار با - cPanel کلیک کنید (+)	۱ ساعت
آموزش مدیریت هاست با - DirectAdmin کلیک کنید - کلیک کنید (+)	۱ ساعت
راه اندازی سایت و کار با وردپرس - کلیک کنید (+)	۷ ساعت
راه اندازی فروشگاه دیجیتال با وردپرس و - Easy Digital Downloads کلیک کنید (+)	۱ ساعت
آموزش راه اندازی سایت شخصی با وردپرس - کلیک کنید (+)	۱ ساعت
آموزش ترجمه قالب وردپرس - کلیک کنید (+)	۲ ساعت
آموزش راه اندازی سایت خبری با وردپرس - کلیک کنید (+)	۲ ساعت

علوم کامپیوتر	
عنوان آموزش	مدت زمان تقریبی
ساختمان داده‌ها -کلیک کنید (+)	۱۰ ساعت
آموزش ساختمان داده‌ها (مرور – تست کنکور ارشد) - کلیک کنید (+)	۲۰ ساعت
آموزش نظریه زبان‌ها و ماشین‌ها -کلیک کنید (+)	۹ ساعت
آموزش نظریه زبان‌ها و ماشین (مرور – تست کنکور ارشد) - کلیک کنید (+)	۸ ساعت
آموزش سیستم های عامل - کلیک کنید (+)	۱۱ ساعت
آموزش سیستم عامل (مرور اجمالی و تست کنکور) - کلیک کنید (+)	۱۲ ساعت
آموزش پایگاه داده‌ها -کلیک کنید (+)	۸ ساعت
آموزش پایگاه داده‌ها (مرور – تست کنکور ارشد) - کلیک کنید (+)	۵ ساعت
آموزش طراحی و پیاده سازی زبان های برنامه سازی - کلیک کنید (+)	۱۰ ساعت
آموزش طراحی و پیاده سازی زبان های برنامه سازی (مرور – تست کنکور ارشد) - کلیک کنید (+)	۱۲ ساعت
آموزش روش های حل روابط بازگشتی - کلیک کنید (+)	۴ ساعت
آموزش روش تقسیم و حل در طراحی الگوریتم - کلیک کنید (+)	۲ ساعت
آموزش ذخیره و بازیابی اطلاعات - کلیک کنید (+)	۸ ساعت
آموزش ساختمان گسسته با رویکرد حل مساله - کلیک کنید (+)	۱۶ ساعت
آموزش جامع مدارهای منطقی - کلیک کنید (+)	۱۰ ساعت
آموزش معماری کامپیوتر با رویکرد حل مسأله - کلیک کنید (+)	۲۰ ساعت
آموزش ساختمان گسسته (مرور و حل تست های کنکور کارشناسی ارشد) - کلیک کنید (+)	۱۲ ساعت
آموزش طراحی الگوریتم - کلیک کنید (+)	۸ ساعت
آموزش شبکه های کامپیوتری ۱ - کلیک کنید (+)	۱۹ ساعت

علوم کامپیوتر (ادامه از صفحه قبل)	
عنوان آموزش	مدت زمان تقریبی
آموزش نظریه گراف و کاربردها - کلیک کنید (+)	۱۴ ساعت
آموزش نتورک پلاس - (Network+) کلیک کنید (+)	۱۰ ساعت
آموزش مدل سازی UML با نرم‌افزار - Rational Roseکلیک کنید (+)	۳ ساعت
آموزش پردازش ویدئو - کلیک کنید (+)	۳ ساعت
پردازش تصویر در متلب - کلیک کنید (+)	۱۶ ساعت
آموزش پردازش تصویر با - OpenCV کلیک کنید (+)	۱۰ ساعت

هوش مصنوعی	
عنوان آموزش	مدت زمان تقریبی
الگوریتم ژنتیک در متلب - کلیک کنید (+)	۱۴ ساعت
الگوریتم PSO در متلب - کلیک کنید (+)	۱۰ ساعت
الگوریتم ازدحام ذرات (PSO) گسسته باینری - کلیک کنید (+)	۲ ساعت
ترکیب الگوریتم ژنتیک و PSO در متلب - کلیک کنید (+)	۱ ساعت
حل مسأله فروشنده دوره گرد با استفاده از الگوریتم ژنتیک - کلیک کنید (+)	۲ ساعت
الگوریتم مورچگان در متلب - کلیک کنید (+)	۶ ساعت
الگوریتم رقابت استعماری در متلب - کلیک کنید (+)	۱۳ ساعت
طراحی سیستم های فازی عصبی یا ANFIS با استفاده از الگوریتم های فرا ابتکاری و تکاملی - کلیک کنید (+)	۲ ساعت
الگوریتم فرهنگی یا Cultural Algorithm در متلب - کلیک کنید (+)	۲ ساعت

هوش مصنوعی (ادامه از صفحه قبل)	
عنوان آموزش	مدت زمان تقریبی
شبیه سازی تبرید یا Simulated Annealing در متلب - کلیک کنید (+)	۴ ساعت
جستجوی ممنوع یا Tabu Search در متلب - کلیک کنید (+)	۲ ساعت
الگوریتم کرم شب تاب یا Firefly Algorithm در متلب - کلیک کنید (+)	۱ ساعت
بهینه سازی مبتنی بر جغرافیای زیستی یا BBO در متلب - کلیک کنید (+)	۲ ساعت
جستجوی هارمونی یا Harmony Search در متلب - کلیک کنید (+)	۲ ساعت
کلونی زنبور مصنوعی یا Artificial Bee Colony در متلب - کلیک کنید (+)	۳ ساعت
الگوریتم زنبورها یا Bees Algorithm در متلب - کلیک کنید (+)	۲ ساعت
الگوریتم تکامل تفاضلی - کلیک کنید (+)	۱ ساعت
الگوریتم بهینه سازی علف هرز مهاجم یا IWO در متلب - کلیک کنید (+)	۲ ساعت
الگوریتم بهینه سازی مبتنی بر و یادگیری یا - TLBO کلیک کنید (+)	۱ ساعت
الگوریتم بهینه سازی جهش قورباغه یا SFLA در متلب - کلیک کنید (+)	۴ ساعت
بهینه سازی چند هدفه در متلب - کلیک کنید (+)	۱۹ ساعت
بهینه سازی مقید در متلب - کلیک کنید (+)	۹ ساعت
شبکه های عصبی مصنوعی در متلب - کلیک کنید (+)	۲۸ ساعت
آموزش کاربردی شبکه های عصبی مصنوعی - کلیک کنید (+)	۹ ساعت
آموزش استفاده از شبکه عصبی مصنوعی با نروسولوشن - کلیک کنید (+)	۳ ساعت
شبکه عصبی GMDH در متلب - کلیک کنید (+)	۴ ساعت
شبکه های عصبی گازی به همراه پیاده سازی عملی در متلب - کلیک کنید (+)	۳ ساعت
طبقه بندی و بازشناسی الگو با شبکه های عصبی LVQ در متلب - کلیک کنید (+)	۳ ساعت
آموزش پیاده سازی الگوریتم های تکاملی و فراابتکاری در سی شارپ - کلیک کنید (+)	۸ ساعت

آمار و داده کاوی	
عنوان آموزش	مدت زمان تقریبی
گنجینه فرادرس های یادگیری ماشین و داده کاوی - کلیک کنید (+)	۸۸ ساعت
گنجینه فرادرس های محاسبات هوشمند - کلیک کنید (+)	۷۱ ساعت
آموزش یادگیری ماشین - کلیک کنید (+)	۲۴ ساعت
داده کاوی یا Data Mining در متلب - کلیک کنید (+)	۲۴ ساعت
آموزش داده کاوی در - RapidMiner کلیک کنید (+)	۲ ساعت
آموزش وب کاوی - کلیک کنید (+)	۱۷ ساعت
شبکه های عصبی مصنوعی در متلب - کلیک کنید (+)	۲۸ ساعت
آموزش کاربردی شبکه های عصبی مصنوعی - کلیک کنید (+)	۹ ساعت
شبکه عصبی GMDH در متلب - کلیک کنید (+)	۴ ساعت
شبکه های عصبی گازی به همراه پیاده سازی عملی در متلب - کلیک کنید (+)	۳ ساعت
طبقه بندی و بازشناسی الگو با شبکه های عصبی LVQ در متلب - کلیک کنید (+)	۳ ساعت
خوشه بندی با استفاده از الگوریتم های تکاملی و فراابتکاری - کلیک کنید (+)	۳ ساعت
تخمین خطای کلاسیفایر یا - Classifier کلیک کنید (+)	۲ ساعت
انتخاب ویژگی یا - Feature Selection کلیک کنید (+)	۲ ساعت
انتخاب ویژگی با استفاده از الگوریتم های فرا ابتکاری و تکاملی - کلیک کنید (+)	۴ ساعت
کاهش تعداد رنگ تصاویر با استفاده از روش های خوشه بندی هوشمند - کلیک کنید (+)	۱ ساعت
آموزش پردازش سیگنال های واقعی در متلب - کلیک کنید (+)	۴ ساعت
مبانی و کاربردهای راهبرد تلفیق داده یا - Data Fusion کلیک کنید (+)	۹ ساعت
آمار و احتمال مهندسی - کلیک کنید (+)	۱۳ ساعت
آزمون‌های فرض مربوط به میانگین جامعه نرمال در - SPSS کلیک کنید (+)	۳ ساعت

آمار و داده کاوی (ادامه از صفحه قبل)	
عنوان آموزش	مدت زمان تقریبی
آموزش محاسبات آماری در اکسل - کلیک کنید (+)	۲ ساعت
آموزش کنترل کیفیت آماری - کلیک کنید (+)	۵ ساعت
آموزش کنترل کیفیت آماری با - SPSS کلیک کنید (+)	۲ ساعت
آموزش مدل سازی معادلات ساختاری با - Amos کلیک کنید (+)	۷ ساعت
تجزیه و تحلیل اطلاعات با نرم‌افزار - SASکلیک کنید (+)	۴ ساعت

مهندسی برق	
عنوان آموزش	مدت زمان تقریبی
طراحی دیجیتال با استفاده از وریلوگ یا - Verilog کلیک کنید (+)	۷ ساعت
آموزش جامع مدارهای منطقی - کلیک کنید (+)	۱۰ ساعت
آموزش مروری طراحی و پیاده سازی مدارات منطقی - کلیک کنید (+)	۴ ساعت
آموزش میکروکنترلر AVR و نرم‌افزار - CodevisionAVRکلیک کنید (+)	۴ ساعت
آموزش تکمیلی میکروکنترلر AVR و نرم‌افزار - CodevisionAVRکلیک کنید (+)	۴ ساعت
آشنایی باPLC های ساخت شرکت های Omron و - Keyence کلیک کنید (+)	۶ ساعت
میکروکنترلر PIC با کامپایلر - CCS کلیک کنید (+)	۹ ساعت
آموزش تحلیل و طراحی مدارات الکترونیکی با - Proteus کلیک کنید (+)	۳ ساعت
آموزش شبیه سازی و تحلیل مدارهای الکتریکی و الکترونیکی با پی اسپایس (PSpice) -کلیک کنید (+)	۳ ساعت
آموزش مقدماتی - ADS کلیک کنید (+)	۳ ساعت
آموزش تکمیلی آنالیز مدار با نرم‌افزار - ADSکلیک کنید (+)	۲ ساعت

مهندسی برق (ادامه از صفحه قبل)	
عنوان آموزش	مدت زمان تقریبی
آموزش تحلیل ریاضی مدارات الکتریکی با - OrCAD کلیک کنید (+)	۲ ساعت
آموزش شبیه سازی مدارات الکترونیکی با - Orcad Capture کلیک کنید (+)	۲ ساعت
آموزش برنامه نویسی آردوینو (- (Arduinoکلیک کنید (+)	۸ ساعت
آموزش تکمیلی برنامه نویسی آردوینو - (Arduino) کلیک کنید (+)	۷ ساعت
آموزش طراحی برد مدار چاپی به کمک نرم‌افزار - Altium Designer کلیک کنید (+)	۷ ساعت
آموزش مبانی ربات های برنامه پذیر - کلیک کنید (+)	۵ ساعت
آموزش ساخت روبات و کنترل آن با اندروید - کلیک کنید (+)	۱۶ ساعت
آموزش مدارهای الکتریکی ۱ - کلیک کنید (+)	۹ ساعت
آموزش مدارهای الکتریکی ۲ - کلیک کنید (+)	۱۱ ساعت
آموزش سیستم های کنترل خطی - کلیک کنید (+)	۱۰ ساعت
آموزش مکاترونیک کاربردی ۱ - کلیک کنید (+)	۱۳ ساعت
آموزش کامسول (مباحث منتخب) - کلیک کنید (+)	۳ ساعت
آموزش سینماتیک مستقیم و معکوس ربات‌ها -کلیک کنید (+)	۳ ساعت
آموزش تجزیه و تحلیل سیگنال‌ها و سیستم ها - کلیک کنید (+)	۲۷ ساعت
آموزش متلب با نگرش تحلیل آماری، تحلیل سری های زمانی و داده های مکانی - کلیک کنید (+)	۸ ساعت
پردازش سیگنال های دیجیتال با استفاده از نرم‌افزار متلب - کلیک کنید (+)	۴ ساعت
شبیه سازی سیستم با سیمیولینک - کلیک کنید (+)	۴ ساعت
آموزش سیستم های قدرت در سیمولینک و متلب - کلیک کنید (+)	۱۱ ساعت
آنالیز پایداری و کنترل سیستم های قدرت با استفاده از جعبه ابزارهای نرم‌افزار متلب - کلیک کنید (+)	۲ ساعت
آشنایی با SimPowerSystems در شبیه سازی سیستم های قدرت - کلیک کنید (+)	۳ ساعت

مهندسی برق (ادامه از صفحه قبل)	
عنوان آموزش	مدت زمان تقریبی
شبیه سازی ماشین های الکتریکی در تولباکس های Simulink و SimPowerSystem در نرم‌افزار متلب - کلیک کنید (+)	۴ ساعت
آموزش الکترونیک قدرت – شبیه سازی در متلب و سیمولینک - کلیک کنید (+)	۸ ساعت
آموزش شبیه سازی عملکرد انواع ماشین های الکتریکی در سیمولینک متلب - کلیک کنید (+)	۱۰ ساعت
برنامه های پاسخگویی بار - کلیک کنید (+)	۴ ساعت
آموزش نرم‌افزار ETAP برای تحلیل سیستم های قدرت - کلیک کنید (+)	۲۱ ساعت
آموزش مقدماتی نرم‌افزار GAMSبرای حل مسائل بازار برق - کلیک کنید (+)	۵ ساعت
آموزش پخش بار اقتصادی (دیسپاچینگ اقتصادی) در GAMS - کلیک کنید (+)	۲ ساعت
کاربرد فازی در سیستم های قدرت - کلیک کنید (+)	۲ ساعت
آموزش نرم‌افزار HFSS - کلیک کنید (+)	۵ ساعت
طراحی آنتن مایکرواستریپ به کمک نرم‌افزار HFSS - کلیک کنید (+)	۱ ساعت
آموزش طراحی و شبیه سازی آنتن های SIW با HFSS - کلیک کنید (+)	۱ ساعت
آموزش بررسی کامل آنتن های مایکرواستریپ و طراحی آن توسط CST - کلیک کنید (+)	۳ ساعت
آموزش تجزیه سیگنال به مولفه های مود ذاتی یا Empirical Mode Decomposition -کلیک کنید (+)	۴۰ دقیقه
نمونه برداری و بازسازی اطلاعات در سیستم های کنترل دیجیتال - کلیک کنید (+)	۳ ساعت
بررسی پاسخ ورودی پله در شناسایی فرآیندهای صنعتی - کلیک کنید (+)	۱ ساعت
مدل سازی و شناسایی سیستم های دینامیکی با استفاده از مدل ARX و شبکه فازی عصبی - ANFIS کلیک کنید (+)	۱ ساعت
طراحی و تنظیم ضرایب کنترل کننده PID با منطق فازی - کلیک کنید (+)	۲ ساعت
آموزش کنترل سیستم چهار تانک - کلیک کنید (+)	۲ ساعت

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 می‌شوند. 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) عبارتنداز :

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 Brust 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 نمی‌باشد.

دسته بندی سیاست های زمان بندی

۱- بدون قبضه کردن (non preemptive)

در حالت بدون قبضه کردن (انحصاری)، همین که یک فرایند درحالت 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 سربار 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 (SPN یا SJF)

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.

۱- MLFQ ۲- MLQ ۳- Priority

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 دستگاههای 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 می شود.

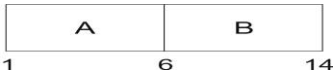
مثال

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

	زمان ورود	زمان اجرا
A	1	5

B	4	8

حل:

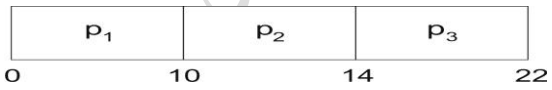


این نمودار مشخص می کند که در لحظه یک اجرای فرایند A ، بدون هیچ انتظاری شروع شده و بعد از 5 میلی ثانیه در لحظه 6 ، پایان یافته است. سپس اجرای فرایند B با 2 میلی ثانیه تاخیر شروع شده و بعد از 8 ثانیه دیگر در لحظه 14 پایان یافته است. از گانت می توان زمان پاسخ را نیز بدست آورد. کافی است که زمان ورود (که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

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

مثال

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

الف- میانگین زمان اجرا

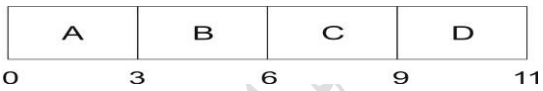
ب- میانگین زمان انتظار

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 وCPU حداکثر به مدت یک کوانتوم زمانی به هر فرایند تخصیص داده می‌شود. اگر در یک

کوآنتوم 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 ان کم است.

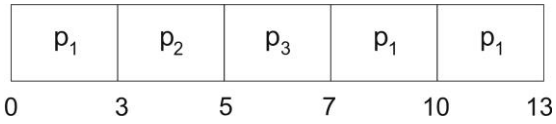
۷- برای فرایندهای کوتاه، زمان برگشت خوبی ارائه می کند.

۸- از مشکلات آن، رفتار با فرایندهای در تنگنای پردازنده در مقایسه با فرایندهای در تنگنای I/O می باشد.

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:



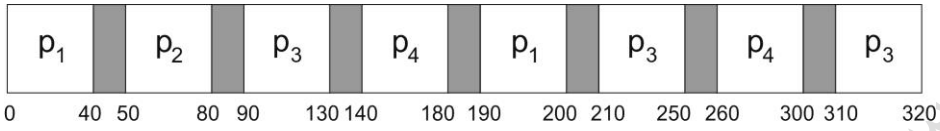
اجرای پروسس P1 در اولین برش زمانی تمام نمی‌شود و به انتهای صف می‌رود و پروسس P2 و P3 قبل از پایان برش زمانی، به پایان می‌رسند. میانگین زمان انتظار برابر است با:

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

مثال

اگر زمان اجرای پروسسهای P4,P3,P2,P1 به ترتیب 80,90,30,50 باشد و روش RR بـا کوانتـوم زمـانی 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 می رساند و فرایند ورودی در لحظـه t، اولویت با فرایند قبلی موجود در سیستم است و در شرایط کاملا یکسان بین دو فرایند، اولویت با فراینـد با شماره کوچکتر است.

حل:

گانت به صورت زیر می باشد:

P_1	P_2	P_1	P_3	P_4	P_2	P_5
0 1	2	3	4	5	6	7

در لحظه 0 ، P1 و P2 وارد می شوند. پردازنده به P1 داده می شود و P2 در صف قرار می گیرد. در لحظه 1 ، پردازنده از P1 گرفته شده و بعد از P2 در صف قرار می گیرد. همچنین P3 و P4 که در همین لحظه وارد

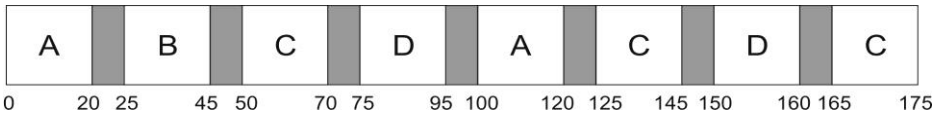
شده اند، در صف پشت سر P1 قرار می گیرند. پردازنده به P2 داده می شود. در لحظه 2 ، پردازنده از P2 گرفته شده و به انتهای صف بعد از P4 رفته و P5 که در همین لحظه وارد شده در انتهای صف بعد از P2 قرار می گیرد. پردازنده هم به P1 (فرایند ابتدای صف)داده می شود. در لحظه 3 ، اجرای P1 تمام شده و پردازنده به P3 که اول صف است داده می شود. در لحظه 4 ، اجرای P3 تمام شده و پردازنده به فرایند اول صف یعنی P4 داده می شود. در لحظه 5 ، اجرای P4 تمام شده و پردازنده به P2 داده می شود. در لحظه 6 ، اجرای P2 تمام شده و پردازنده به P5 داده می شود.

مثال

به کمک روش زمان بندی نوبه‌ای با کوانتوم زمانی 20 میلی ثانیه، متوسط زمان انتظار پردازه‌ها 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 :

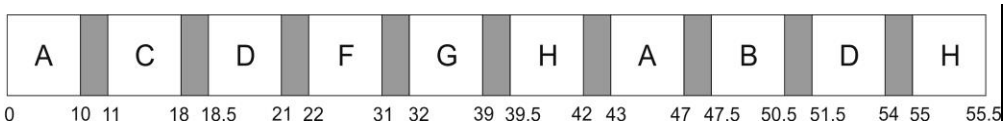
$$\frac{(120-40)+(45-20)+(175-50)+(160-30)}{4}=\frac{80+25+125+130}{4}=90$$

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:

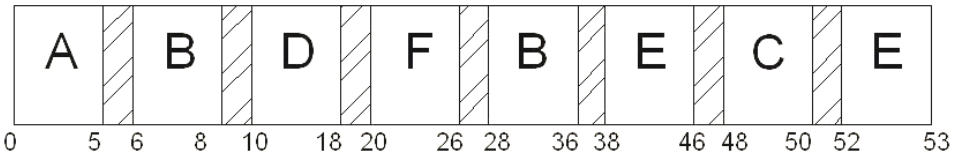


مثال

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 نخ های 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 (SPN یا SJF)

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.

ویژگی های 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 محیط های اشتراک زمانی، مناسب نیست.

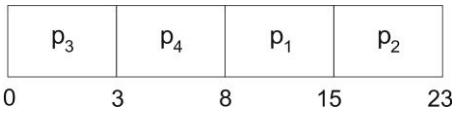
۵- میانگین زمان انتظار، کمینه است.

مثال

میانگین زمان انتظار را برای فرایندهای زیر به روش SJF بدست آورید؟(ورود همه در لحظه صفر)

(P1=7 , P2=8 , P3=3 , P4=5)

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



میانگین زمان انتظار برابر است با:

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

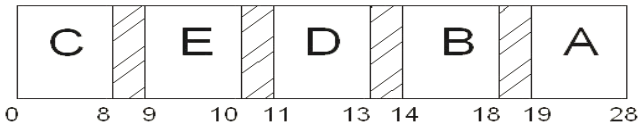
مثال

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:

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

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. $(0 \le \alpha \le 1)$

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 سیکلn 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: $A \to B \to C$

(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 می‌شود که انتظار می‌رود کوتاهترین زمان پردازش باقیمانده را داشته باشد. اگر فرایند جدیدی وارد صف آماده شود و زمان باقیمانده کمتری نسبت به فرایندی که درحال اجراست داشته باشد، فرایند درحال اجرا قبضه می‌شود و فرایند جدید اجرا می‌شود.

ویژگی های الگوریتم SRT

۱- غیر انحصاری است.

۲- امکان گرسنگی برای کارهای طولانی زیاد است.

۳- بر خلاف RR، وقفه های اضافی بوجود نمی‌آید، بنابراین سربار کاهش می یابد ولی از طرف دیگر، زمان خدمت سپری شده باید ثبت شود، که ایجاد سربار می کند.

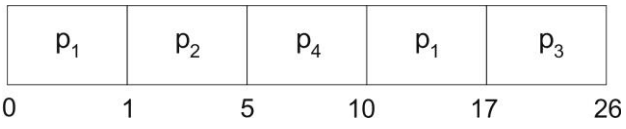
۴- زمان کل SRT نسبت به SPN بهتر است، چون کار کوتاه اولویت بیشتری نسبت به کار بلند در حال اجرا دارد.

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:

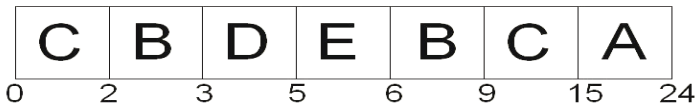
$$\frac{(17-8-0)+(5-4-1)+(26-9-2)+(10-5-3)}{4}=6.5$$

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 انتظاربرای 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 قدیمی‌تر و بلند تر ارجحیت دارند. در صف با کمترین اولویت از سیاست RR و درصفهای دیگر از سیاست FCFS استفاده می‌شود. (چون فرایند درصف کمترین اولویت، نمی‌تواند به صف‌های پایین تر برود.)

زمان بندی FB غیر انحصاری می باشد، چون فرایندی از صفی به صف دیگر می تواند 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 اخر از روش FCFS استفاده می کند. زمانبند ابتدا تمامی فرایندهای موجود درصف 0 را اجرا می‌کند، اگر صف 0 خالی بود فرایندهای صف 1 اجرا می‌شود و اگر صف 0 و1 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.

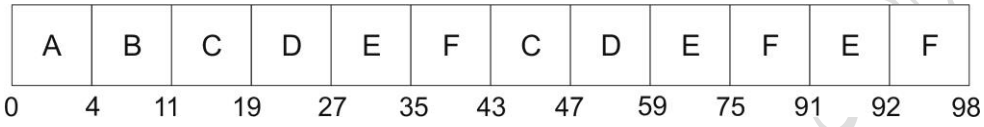
مثال

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 می شود. همچنین صف سوم با روش FCFS زمان بندی می شود. 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 برنامه ها وارد صف اول می شوند و در صورت نیاز به بیش از 8 میکروثانیه به صف دوم وارد می شوند(فرایندهای F,E,D,C) و در صف دوم در صورت نیاز به بیش از 16 میکروثانیه به صف سوم منتقل می شوند (فرایندهای F,E) و در صف سوم به روش FCFS به آنها رسیدگی می شود:



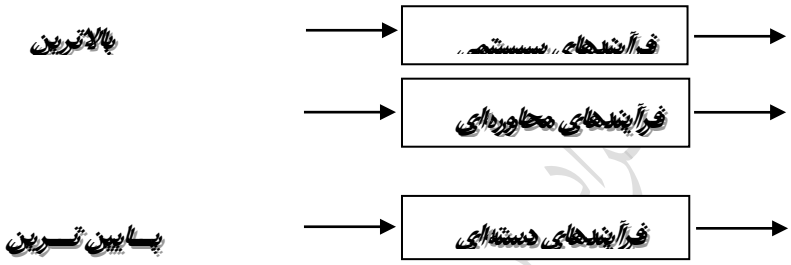
میانگین زمان انتظار برابر است با :

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

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 می شوند. اولویت فرایندهای پیش‌زمینه ممکن است از پس‌زمینه بالاتر باشد. در صف پیش‌زمینه از الگوریتم RR و در صف پس زمینه 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 می دهد:



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 بالاتراست.

زمان بندی اولویت (Priority)

در این الگوریتم به هر فرایند اولویتی 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

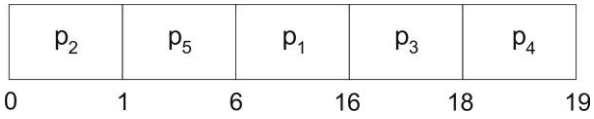
میانگین زمان انتظار برای پنج فرایند زیر را در روش زمان بندی اولویت بدست آورید.

(اعداد کوچک، اولویت بالا را نشان می‌دهد.)

فرایند	زمان اجرا	اولویت

P1	10	3
P2	1	1
P3	2	3
P4	1	4
P5	5	2

حل: گانت فرایند ها به صورت زیر است:



میانگین زمان انتظار :

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

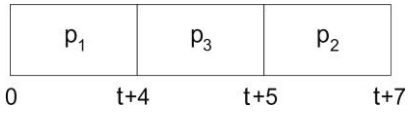
مثال

برای سه پردازه زیر، زمان متوسط پاسخگویی در روش اولویت کدام است؟

(عدد کمتر در ستون اولویت، نشان دهنده اولویت بیشتر است.)

پردازه	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.

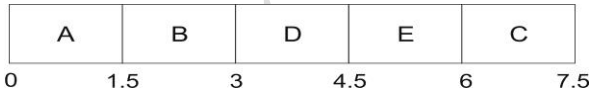
LCFS : Last-Come First-Served

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:



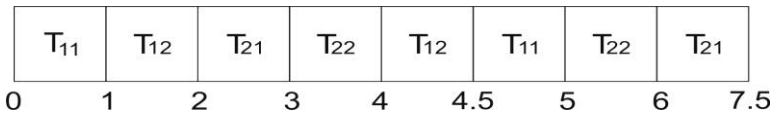
پردازنده در لحظه 0 به تنها فرایند موجود یعنی A داده می شود و چون قبضه شدنی نمی باشد، تا پایان اجرای فرایند A ، پردازنده را در اختیار دارد. در زمان 1.5 فقط فرایند B رسیده و پردازنده را می گیرد. در زمان 3 که C و D حاضرند، ابتدا پردازنده به D داده می شود، چون آخرین ورودی است. بعد از اجرای D ، چون E نیز رسیده است، پردازنده به E داده می شود و درنهایت 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
P1	T11	1.5	0
	T12	1.5	1
P2	T21	2.5	2
	T22	2	3

حل: نمودار گانت به صورت زیر می باشد:



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:

۱- LPT ۲- RPT ۳- SPT

الگوریتم LPT

این الگوریتم از بین کارهای باقیمانده، طولانی ترین کار را برای اجرا انتخاب می کند. این الگوریتم بهینه نیست ولی معمولا 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 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 کـارش را انجـام داد، کـار بعـدی را اجـرا 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 می دهد. در واقع RPT مانند LPT ای است که ترتیب تکالیف هر پردازنده معکوس شده است.

مثال

نتیجه استفاده از RPT برای $\{\tau_1\}=\{13,8,7,6,4,2,2,1\}$ 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 چندپردازنده ای، ابتدا کارها را بر اساس زمان اجرای افزایشی مرتب کرده و سـپس m تـا کـار اول را برای اجرا در m پردازنده موجود زمانبندی می کند (یک کار برای هر یک از پردازنده هـا)، سـپس m تا کار بعدی زمانبندی می گردند و بهمین ترتیب تا آخر.

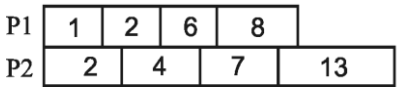
مثال

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

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

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

سپس کارها را به ترتیب به پردازنده ها داده و هر پردازنده ای که کـارش را انجـام داد، کـار بعـدی را اجـرا خواهد کرد:



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

■

مشخصات سیاست های زمان بندی

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

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 می‌شود)

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:

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

حل: جواب گزینه ۲ است. چون میانگین زمان اجرای فرایندها قبل از ورودی/خروجی کمتر از زمان برش زمانی است، اجـرای فرایندها قبل از پایان کوانتوم زمانی، به اتمام می رسد. بنابراین کارایی پردازنده، یعنی نسبت زمـان مفیـد (r) بـه کـل زمـان (r+c) برابر است با: $\frac{r}{r+c}$

(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؟

۱) FCFS ۲) SJF ۳) SRT ۴) RR

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 الگوریتم های زمان بندی انحصاری، دارای میانگین زمان برگشـت کمتری است.

گزینه ۲ درست است ، چون در این الگوریتم کارهای کوتاه تر، زودتر اجرا می شوند، بنابراین تعـداد کارهـای انجـام شـده در واحد زمان (توان عملیاتی) بیشتر است.

گزینه ۳ درست است، چون الگوریتم SJF اولویت را به کارهای کوتاه تر می دهد، یک الگوریتم اولویت است.

گزینه ۴ نادرست است، چون SJF سعی به کم کردن تعداد تعویض متن ها ندارد، بنابراین بهره وری CPU را بالا نمی برد.

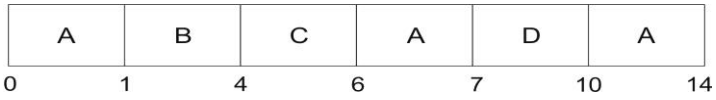
(مهندسی کامپیوتر- آزاد ۸۳)

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؟

۱) 1.5 ۲) 0.25 ۳) 2.5 ۴) 2.25

حل: جواب گزینه ۴ است.

نمودار گانت به صورت زیر می باشد:



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 صف های چند گانه، اولویت لزوما 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

۱) 12.75 ۲) 13.75 ۳) 14 ۴) 13

پاسخ: جواب گزینه ۱ است.

در زمان صفر فقط فرایند P1 در سیستم وجود دارد و پردازنده به آن داده می شود تا اجرایش به پایان برسد. در این زمان (یعنی 7) ، اولویت فرایندها طبق فرمول الگوریتم HRRN یعنی $\frac{w+s}{s}$ محاسبه می شود:

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

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

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

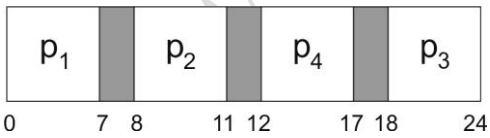
حال چون اولویت P2 از فرایندهای P3 و P4 بیشتر است ، پردازنده در زمان 8 (بعد از یکی میلی ثانیه از پایان اجرای P1 به علت تویض متن) به آن داده می شود. اجرای P2 در لحظه 11 به پایان می رسد. در زمان 11 ، مجددا اولویت فرایندهای باقی مانده را حساب می کنیم:

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

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

بنابراین پردازنده به P4 داده می شود. و در نهایت بعد از اجرای P4 به P3 داده می شود.

نمودار گانت به صورت زیر است:



بنابراین میانگین زمان برگشت برابر است با:

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

فصل ۴

هم روندی: انحصار متقابل و همگام سازی

عملکرد چند برنامه ای به خاطر این ابداع شد که زمان پردازش کامپیوتر به صورت پویا بین تعدادی کار بتواند تقسیم گردد. برای زمینه های چند برنامه ای، چند پردازشی و پردازش توزیعی و همچنین طراحی سیستم عامل، موضوع همزمانی اساسی است. فرایندهای هم روند (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- همگام سازی (Synchronization)

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

۲- تبادل اطلاعات (Communication)

فرایندها می توانند با مکانیسم هایی چون "حافظه مشترک، تبادل پیام، فایل مشترک و لوله" با یکدیگر تبادل اطلاعات کنند.

۳- رقابت فرایندها

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

حالت های 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 گوییم. 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. اگرP3 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 ایـن عمـل بـین 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 پردازه P1 و P2 به صورت هم روند وجود دارند. نحوه ایجاد رشته $A(CD)^*B$ چگونه می باشد؟

P1	P2

while(TRUE){
cout<< "A";
cout<< "B";
}
	while(TRUE){
	cout<<"C";
	cout<<"D";
	}

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
while (x=0); cout<< "A"; cout<< "D"; y=1;	cout<<" B"; x=1; **while** (y=0); cout<<"C";

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 دستورالعمل های عادبی برنامه خود را اجرا می کند، نباید در تصمیم گیری 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 حل ها را به صورت چهار رویکرد زیر، دسته بندی می کنیم:

۱- نرم افزاری

۲- با حمایت سخت افزار (با کمک دستورالعمل های خاص CPU)

۳- با حمایت سیستم عامل (با کمک فراخوان های سیستمی خاص)

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

رویکردهای نرم افزاری انحصار متقابل

راه حل های نرم افزاری مستقیما توسط برنامه ها استفاده می شوند و وجود حافظه اشتراکی ضروری می باشد. در این راه حل ها از دستورالعمل های خاص توسط سخت افزار استفاده نمی شود و حمایتی از سیستم عامل و زبان های 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 تلاش ها 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 می تواند بگیرد. مقدار اولیه turn برابر 0 است. بنابراین ابتدا P0 می تواند وارد ناحیه بحرانی شود. در این حالت P1 در حلقه while منتظر می ماند تا P0 از ناحیه بحرانی خارج شده و turn را 1 کند. در این صورت P1 می تواند وارد ناحیه بحرانی شود. برنامه فرایندها به صورت زیر است:

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

```
P1(void)
{
  while(TRUE)
   {
     while( turn != 1) ;  /*wait*/
     critical–section( );
     turn = 0;
     non-critical- section( );
    }
```

}	}

بررسی شرط ها در تلاش اول:

۱- انحصار متقابل

راه حل گفته شده انحصار متقابل را تضمین می کند و امکان ندارد که P0 و P1 با هم وارد ناحیه بحرانی شوند. چون P0 در صورتی وارد ناحیه بحرانی می شود که مقدار turn برابر 0 باشد و P1 با مقدار 1 وارد می شود. بنابراین امکان ندارد که هر دو به turn نگاه کنند و هر دو با هم مقدارهای 0 و1 را ببینند.

۲- پیشرفت

این روش شرط پیشرفت را رعایت نمی کند. چون اگر P1 وارد ناحیه بحرانی شود و کارش تمام شده و به بخش غیر بحرانی برود، در این حالت turn برابر 0 می باشد. حال نوبت P0 است که وارد ناحیه بحرانی شود، ولی می خواهد 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 می ماند 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 می شوند. همچنین بن بست نیز نداریم. بنابراین شرط انتظار محدود رعایت می شود.

- یکی از معایب این روش این است که سرعت عملیات توسط فرایند کندتر تعیین می شود، 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 می شود که هر کدام متعلق به یک فرایند است. هر فرایندی که قصد ورود به ناحیه بحرانی خود را دارد، پرچم خود را TRUE می کند.

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};

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

بررسی شرط ها در تلاش دوم

۱- انحصار متقابل

این روش انحصار متقابل رعایت نمی شود. یعنی حتی از روش اول هم بدتر است.

سناریو: فرض کنید که P0 ، flag[1] را خوانده و آن را FALSE می بیند، اما قبل از TRUE کردن flag[0]، P1 اجرا شود و flag[0] را خوانده و آن را FALSE می بیند و برای ورود به ناحیه بحرانی flag[1] را TRUE کرده و وارد می شود. حال در این زمان به P0 سوئیچ شده و flag[0] را TRUE کرده و این فرایند هم وارد ناحیه بحرانی می شود! بنابراین چون هر دو فرایند در یک زمان وارد ناحیه بحرانی خود شده اند، شرط انحصار متقابل برقرار نمی باشد.

۲- پیشرفت

این روش شرط پیشرفت را رعایت می کند. اگر P0 در ناحیه غیر بحرانی خود باشد، flag[0] را FALSE نگه می دارد 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 می تواند وارد ناحیه بحرانی شود و سپس خارج شود. یعنی P0 جلوی پیشرفت P1 را نمی گیرد.

۳- انتظار محدود

این روش شرط انتظار محدود را رعایت نمی کند، 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 می شود. چون flag[0] برابر TRUE است، P1 نمی تواند وارد ناحیه بحرانی شود. بعد از پایان کوانتوم، پردازنده به P0 داده شده و این فرایند سریعا 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 مجددا 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};
```

```
P0(void){
  while(TRUE)   {
     flag[0] = TRUE ;
     while( flag[1] );
     critical-section( );
     flag[0]= FALSE;
     non-critical- section( );
   }
}
```

```
P1(void){
  while(TRUE)   {
     flag[1] = TRUE;
     while( flag[0] );
     critical-section( );
     flag[1]= FALSE;
     non-critical- section( );
   }
}
```

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، falg[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، falg خود را برای مدت کوتاهی FALSE کرده تا فرایند مقابل بتواند وارد شود.

برنامه فرایندها به صورت زیر است:

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

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( );
}
}
	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( );
	}
	}

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 می دهیم:

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 می کنند. سپس با هم بر می گردند و مراحل بالا را تکرار می کنند. اگر این دنباله به طور نامحدود تکرار شود، ممکن است هیچ کدام از فرایندها نتوانند وارد ناحیه بحرانی شوند. البته این تکرار بن بست نمی باشد، چون با تغییر در سرعت نسبی فرایندها، این چرخه شکسته می شود.

تلاش پنجم (راه حل صحیح)

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

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

```
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 ( );
  }
}
```

```
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( );
  }
}
```

هنگامی که P0 بخواهد وارد بخش بحرانی خود شود، در flag مربوط به خود مقدار TRUE می گذارد. سپس flag مربوط به P1 را بررسی می کند که دو حالت رخ می دهد:

الف- flag[1] برابر TRUE باشد:

اگر turn برابر یک باشد، P0 به P1 احترام گذاشته و با FALSE کردن پرچم اش منتظـر مـی مانـد. در ایـن هنگام P0 کاری انجام نمی دهد تا turn برابر صفر شود و سپس flag خودش را TRUE می کند.

ب- flag[1] برابر FALSE باشد:

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

خلاصه ای از وضعیت تلاش های Decker

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

تلاش های Decker	انحصار متقابل	پیشرفت	انتظار محدود
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;
```

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

- 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 وP1 به طور تقریبا 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 برابر1 می شود. حال زمانی که P0 وP1 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، عبارتنداز:

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- دستور العمل 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 سوئچ 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.

دستور العمل TSL

بسیاری از کامپیوترها دارای دستورالعمل TSL(Test and Set Lock) می باشند. این دستور 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 دستورالعملی 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 ابـزاری بـه نام سمافور را معرفی می کنیم که دارای قدرت زیادی در برقراری انحصار متقابل مـی باشـد و همچنـین 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 ایـن کـار 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 وsignal به صورت زیر است:

```
void wait(semaphore s)
{
  s.count = s.count –1;
  if (s.count < 0)
  {
   place this process in
s.queue;
   block this process;
  }
}
```

```
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;
  }
}
```


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 می کند، اگر مقدار شمارنده بیشتر از صفر نشود، یک فرایند از قبل مسدود شده در صف، آزاد می شود.

- در بعضی از متون از حرف P به جای wait و از حرف V به جای signal استفاده می شود.
- در بعضی از متون از down به جای wait و از up به جای signal استفاده می شود.

سمافور بر دو نوع عمومی و دودویی می باشد. در سمافور عمومی که آن را بررسی کردیم، شمارنده می تواند مثبت، صفر و یا منفی باشد. اما شمارنده در سمافور دودویی، فقط مقادیر 0 و1 را دریافت می کند.
توابع wait وsignal برای سمافور باینری به صورت زیر است:

```
void wait(semaphore s)
{
  if (s.count =1)
     s.count = 0;
  else{
    place this process in
s.queue;
    block this process;
  }
}
```

```
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;
  }
}
```

- قدرت سمافور دودویی معادل با سمافور عمومی است.

در سمافورها صفی برای نگهداری فرایندهای بلوکه شده استفاده می شود. اگر خروج از ایـن صـف بـه ترتیب ورود باشد(FIFO) ،به آن سمافور قوی می گویند و اگر این چنین نباشد، به آن سمافور ضعیف می گویند. در سمافور ضعیف، امکان گرسنگی وجود دارد. در این کتاب سمافورها از نوع قوی فرض می شوند.

انحصار متقابل با استفاده از سمافورها

می خواهیم 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 signal و wait در کجای برنامه باید روی سمافور انجام شود.

مسئله انحصار متقابل برای دو فرایند p1 و p2 توسط برنامه زیر پیاده سازی 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 می کند و وارد ناحیه بحرانی می شود. 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 می گیرد. 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 می کند و P2 می تواند وارد ناحیه بحرانی شود.

- می توان با استفاده از سمافور عمومی، انحصار متقابل را برای بیش از دو فرایند نیز پیاده سازی کرد.
- سمافوری که ترتیب خروج در آن مشخص نباشد، سمافور ضعیف نامیده می شود. در این نوع سمافور، امکان گرسنگی وجود دارد.(در سمافور قوی، منطق صف FIFO است.)

پیاده سازی انحصار متقابل به کمک سمافور دارای مزایایی است از جمله: ارضای شرط های انحصار متقابل،پیشرفت و انتظار محدود . این روش عادلانه است،CPU را تلف نمی کند 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 دستورS1 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; signal(S); .	. wait(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

.
.
signal(S);
	wait(Q);
	.
	.
		wait(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(S);
cout<<"C";
cout<<"D";
	wait(Q);
	cout<<"A";
	signal(S);
	cout<<"B";

	cout<<"E";

پاسخ:

در زیر ترتیب اجرا با شماره مشخص شده است:

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

پاسخ: به علت استفاده از دستور wait(S) در ابتدای P2 ، و صفر بودن S ، اجرا نمی تواند با P2 شروع شود. بنابراین ابتدا P1 اجرا شده و عدد 1 چاپ می شود. سپس توسط signal(S) مقدار سمافور S برابر یک می شود. به علت رسیدن به wait(Q) ، و صفر بودن Q ، نمی توان ادامه داد . با تعویض متن به p2 ، چون S برابر یک شده از wait(S) رد شده و S صفر می شود. سپس 3 چاپ می شود. حال دستور signal(Q) مقدار Q را برابر یک کرده و با رسیدن به wait(S) ، چون S ، صفر است، نمی توان 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 می شود. 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 می شود. 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;
b=b+1;
signal(s);
b=b+1;
	a=3;
	wait(s);
	b=a+b;
	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

مجددا 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 وسمافور 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 دستور wait(empty) از تولید کننده و دستور 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 می شود، سعی می کند دو چنگال سمت چپ و راست خود را بردارد. اگر موفق شد، برای مدتی غذا می خورد و سپس چنگال ها را زمین می گذارد و به فکر کردن ادامه می دهد. مسأله تغذیه فیلسوفان علاوه بر **انحصار متقابل** (که در یک زمان دو فیلسوف نمی توانند از یک چنگال استفاده کنند)، باید جوابگوی **بن بست** و **گرسنگی** نیز باشد.

راه حل: هرفیلسوف ابتدا چنگال چپ و سپس چنگال راست را بر می دارد. بعد از تغذیه یک فیلسوف، دو چنگالی که استفاده می کرد را روی میز گذاشته و دیگران می توانند 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

نمی شد، و اجازه ورود همزمان هر پنج نفر، داده می شد، همه آنها چنگالهای چپ خود را برداشته و دیگر چنگال اضافی نمی ماند که کسی بتواند چنگال راست خود را بردارد. بنابراین بن بست رخ می داد.

در کتاب تننباوم، راه حل زیر داده شده است. در این راه حل، از یک آرایه به نام state استفاده می کند که وضعیت جاری فیلسوف(فکر کردن(0) ،گرسنگی(1) و خوردن(2)) را نگهداری می کند. یک فیلسوف در صورتی که هیچ یک از فیلسوفان چپ و راستش، در حال خوردن نباشند، می تواند 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 اولیت دارند. (تا زمانی که خواننده ای وجود دارد، به خواننده اجازه ورود می دهیم)

۲) نویسندگان اولویت دارند.

۳) خوانندگان و نویسندگان بدون اولیت هستند.

حالتی را بررسی می کنیم 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 می شود. در این روال، تا هنگامی که نویسنده ای به فایل دسترسی دارد، هیچ نویسنده ای و خواننده دیگری نمی تواند 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 می کند.

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 می شود،البته در صورتی که نویسنده ای فعال نباشد.

توسط if پایانی، بررسی می کنیم که اگر خواننده فعال دیگری وجود نداشته باشد، در صورت اینکه نویسنده بلوکـه شـده ای داشته باشیم، به آن اجازه داده شود.

در این روش شرط انحصار متقابل و شرط پیشرفت رعایت می شود ولی شرط انتظـار محـدود رعایـت نمی شود. (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 می باشد. 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 می دهد و کنترل آن هم ساده تر است. سمافور اغلب توسط سیستم عامل پشتیبانی می شود و می تواند 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 می دهد 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 می شود.

۳ – در هر زمان تنها یک فرایند می تواند 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 هـایی 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 فورا 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. فرخوان recevie، 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 مستقیما 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 recive را اجرا می کند، دو امکان وجود دارد:

الف- اگر پیامی قبلا 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 شونده یا مسدود نشونده باشند.

پیاده سازی انحصار متقابل توسط تبادل پیام

از تبادل پیام می توان برای اعمال انحصار متقابل استفاده کرد. فرض می کنیم کهsend 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 کـه پیـامی فـراهم مـی 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 پستیmc 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:

```
void producer( )
{
   message m1;
   while(TRUE)
    {
       receive(mp,m1);
       m1=produce( );
      send(mc,m1);
    }
}
```

```
void consumer( )
{
   message m2;
   while(TRUE)
    {
      receive(mc,m2);
      consume(m2);
      send(mp,null);
    }
}
```

زمانی که فرایندهای فرستنده و گیرنده پیام بر روی یک ماشین اجرا می شوند، مسئله کارایی پیش می اید.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 وb وc یک می باشد.

۲) مقادیر b وc 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 وb 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 می کند. کدام گزینه صحیح است؟

```
#define   FALSE  0
# define  TRUE    1
```

```
# define  N   2
int   turn;
int   intersted[N];   /*  all  values initially  0(FALSE)   */
void  enter-region(int  process){
  int  other;
 other =1 - process;
  interested[process] = TRUE;
  turn = process;
  while (turn == process  &&  interseted[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 می شود. ۴) انحصار متقابل تأمین نمی شود.

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 تامین می کند. 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 می کند.

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 flase 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( flag[(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 p0 مقدار 1 و برای p1 مقدار 0). که این کار باعث نادرست شدن روش می شود.

فرض کنید P0 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 P1 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 $P_i : (i = 1,2)$ درنظر 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 کاملا صحیح است.

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

۳) شرط انحصار متقابل (multual exclusion) تنها شرطی است که نقض می گردد.

۴) هر دو شرط انحصار متقابل و پیشرفت نقض می شوند.

پاسخ: گزینه ۴ جواب است.

مقدار flag ها در این راه حل بی اثرند. چون در بدنه هر فرایند، به جای بررسی flag فرایند مقابل، flag خود فرایند چک می شود. بعد از حذف flag ها، کد هر دو فرآیند به صورت زیر می باشد:

P1	P2

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

اگر p1 اجرا شود، مقدار turn را به 2 تغییر می دهد و در حلقه while منتظر می ماند. اگر در این لحظه به p2 ، سوئیچ شود p2، نیز مقدار turn را به 1 تغییر می دهد و در حلقه while منتظر می ماند. بنابراین بن بست رخ داده و شرط انتظار محدود نقض می شود. شرط پیشرفت نیز نقض می شود، 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 می شود. با این شرایط بدیهی است که شرط انحصار متقابل نیز نقض می شود. بنابراین هر ۳ شرط "انحصار متقابل، پیشرفت و انتظار محدود" نقض می شود.

تذکر: بهتر بود طراح محترم، در گزینه ۴ ، شرط انتظار محدود را نیز اضافه می کرد، تا داوطلب فکر نکند که منظور این است که فقط آن دو شرط که در گزینه ۴ آمده، نقض می شود.

(مهندسی IT – دولتی ۹۱)

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

```
proc ( int  i ){
    while(TRUE){
      computeation;
      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;
```

wait(semaphore ***S**)	**signal**(semaphore ***S**)
{	{

S −> value--;
if (S−> value < 0)
{
add this process to S−>List;
block(-(S −>value));
}
}
	S −>value++;
	if (S−>value <= 0)
	{
	remove a process P from S−>List;
	wakeup(P);
	}
	}

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 می گیرد و به ترتیب n (از کوچک به بزرگ) توسط wakeup(P) از حالت بلوک خارج می شوند.

$S->value=1$

برنامه های 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؟

تولید کننده	مصرف کننده
while (1) {	**while** (1) {

wait (z) **wait** (y) حذف قطعه از بافر **signal**(y) **signal** (x) استفاده از قطعه }	تولید قطعه **wait** (x) **wait** (y) اضافه کردن قطعه به بافر **signal** (y) **signal** (z) }

۱) x=0 , y=1 , z=n ۲) x=n , y=n , z=1

۳) x=n , y=1 , z=0 ۴) x=0 , y=n , z=1

پاسخ: جواب گزینه ۳ است. از x برای شمارش تعداد خانه های خالی با مقدار اولیه n ، از z برای شمارش تعداد خانه های پـر 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 ترتیب0,1 باشد، چنانچه دو زیر روال به طور هم روند اجرا شوند، کدام یـک از گزینـه های زیر صحیح است؟

procedure **producer**
begin
repeat
produce;
wait(s);
append;
signal(n);
signal(s);
forvere
end;
	procedure **consumer**
	begin
	repeat
	wait(s);
	wait(n);
	take;
	signal(s);
	signal(n);
	forever
	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 اسـت. 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 شـود.

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 وگرنه عمل بررسی را تکرار می کند.

روال put-fork(i) چنگال شماره i را می گذارد و از روال خارج می شود.

```
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 نمی شود، بنابراین بن بست رخ نمی دهد.

در این روش امکان گرسنگی وجود دارد، چون همه چیز تصادفی است و نوبت و عدالت رعایت نمی شود 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 می تواند فعال باشد.

۲) یک مانیتور هیچ گاه در بخش بحرانی خود مسدود نمی شود.

۳) پیاده سازی مانیتور در سطح کامپایلر انجام می شود.

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

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 مجموعه ای از فرایندها که برای منابع سیستم رقابت می کنند یا با یکدیگر در ارتباط هستند. این مسدود بودن ادامه می‌یابد تا سیستم عامل عمل فوق العاده‌ای 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 فعلا 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)

مجموعه ای از فرایندهای منتظر $\{P_0, P_1, ..., P_n\}$وجود دارد که P_0 منتظر منبعی باشد که در اختیار P_1 است و P_1منتظر منبعی باشد که در اختیار P_2است و به همین ترتیب، P_{n-1}منتظر منبعی باشد که در اختیار P_n است و در نهایت P_nمنتظر منبعی است که در اختیار P_0 است. در واقع چرخه ای 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 می‌گردد.

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 می‌دهد.

اگر در سیستمی n فرایند و m منبع (از یک نوع) موجود باشد، در صورت برقرار بودن شرط $\sum_{i=1}^{n} \mathrm{Re}\, quest(i) < m + n$ 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:

$$\sum_{i=1}^{n} \text{Re}\,quest(i) < m+n \Rightarrow 2n < m+n \Rightarrow n < m$$

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 پردازه برای دستیابی به آنها رقابت می‌کنند. هر پردازه به دو درایو نیاز دارد. این سیستم به ازای حداکثر چه ارزشهایی از n فاقد بن بست است؟

$$\sum_{i=1}^{n} \text{Re}\,quest(i) < m+n \Rightarrow 2n < 6+n \Rightarrow n < 6$$

بنابراین حداکثر پنج فرایند می‌تواند وجود داشته باشد.در این حالت اگر هرکدام از فرایندها یک منبع را در اختیار گیرند، یک منبع آزاد باقی می‌ماند که یکی از فرایندها با گرفتن آن کامل می‌شود و به همین ترتیب بقیه فرایندها کامل خواهند شد و بن بست رخ نمی‌دهد.

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.

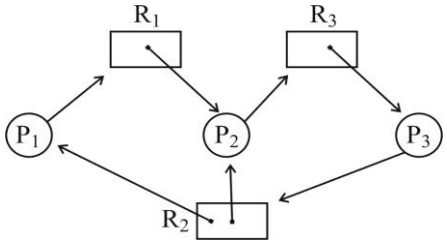
۲- یال تخصیص $R_j \rightarrow P_i$

نمونه ای 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 P1 نمونه‌ای از منبع R2 را در اختیار دارد و منتظر نمونه‌ای از منبع R1 است.

۲- فرایند P2 نمونه‌ای از منبعR1 و منبع R2 را در اختیار دارد و منتظر نمونه‌ای از منبع R3 است.

۳- فرایند P3 نمونه‌ای از منبع R3 را در اختیار دارد و منتظر نمونه‌ای از منبع R2 است.

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

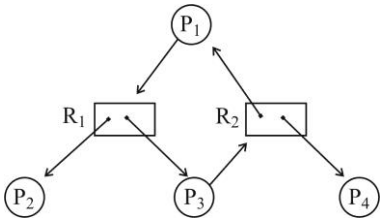
1) $P1 \to R1 \to P2 \to R3 \to P3 \to R2 \to P1$

2) $P2 \to R3 \to P3 \to R2 \to P2$

■

مثال

در گراف زیر وضعیت بن بست را بررسی نمایید.

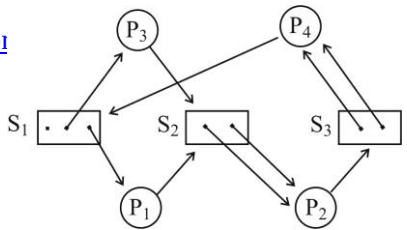


حل: با وجود چرخه $P_1 \to R_1 \to P_3 \to R_2 \to P_1$، بن بست رخ نمی دهد. چون فرایند P_4 می‌تواند منبع نمونه R_2 را آزاد کند و آنگاه منبع R_2 به فرایند P_3 داده می‌شود و چرخه از بین برود.■

اگر چرخه ای در گراف وجود داشته باشد، ممکن است بن بست وجود داشته باشد.

مثال

سیستمی متشکل از چهار پردازه P1 وP2 وP3 وP4 و سه نوع منبع قابل استفاده مجدد S1 و S2 وS3 را در



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

حل: تعداد واحدهای هر منبع به ترتیب 2,2,3 است. P1 یک واحد از منبع S1 را در اختیار دارد و تقاضای یک واحد از S2 را کرده است. P2 دو واحد از S2 را در اختیار دارد و تقاضای یک واحد از S3 را کرده است. P3 یک واحد از S1 را در اختیار دارد و تقاضای یک واحد از S2 را کرده است. P4 نیز دو واحد از S3 را دراختیار دارد و تقاضای یک واحد از S1 کرده است. فرایند P4 یک واحد از S1 که موجود است را در اختیار گرفته و کامل می‌شود. بعد از کامل شدن P4، منابعی را که در اختیار داشت (دو تا s3 و یک s1) را آزاد می کند. سپس فرایند P2 کامل می‌شود، چون منابع درخواستی موجود است. بعد از P2 فرایند P1 و P3 کامل می‌شوند. یعنی هیچ فرایندی درحالت بن بست نمی‌باشد.

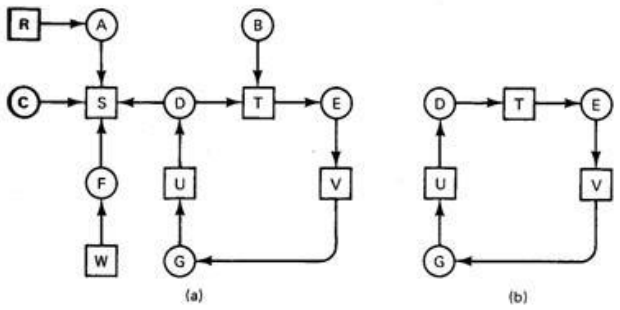
اگر هر نوع منبع دارای چند نمونه باشد، وجود چرخه الزاماً به معنای بن بست نمی‌باشد. یعنی در این حالت وجود چرخه شرط لازم برای وجود بن بست است ولی شرط کافی نیست.

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 می‌شود. سیستم عامل به طور متناوب الگوریتمی را دنبال می‌کند 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 Tapeویک 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 توانـد اجرا شود، چون نیاز به یک پرینتر دارد که موجود نیست. ولی فرایند سوم می تواند اجرا شود، چـون بـه 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 تاجایی که دیگر بن بست وجود نداشته باشد.

د- قبضه کردن پی درپی 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 نمی توان 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 Ri قبل از Rj است. اگر فرایند P1 منبع Ri را در اختیار داشته باشد و Rj را تقاضا کند و فرایند P2 منبع Rj را در اختیار داشته باشد و Ri را تقاضا کند، این حالت غیر ممکن است چون مستلزم i<j و j<i می باشد.

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 نمی تواند 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 تیستند، 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 R1 ، دو منبع R2 ، پنج منبع R3 تخصیص داده شده اند. بنابراین با توجه به منابع اولیه، منابع بعد از تخصیص به صورت (R1=0,R2=1,R3=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 حالـت P1 می‌تواند اجرا شود. بعد از اجرای P1،منابع در اختیار او به بردار Available اضافه می شود:

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 می شود و بردار Availble مانند بردار Resorce خواهد شد.

بنابراین می توان 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		
طرح	**مزایا**	**معایب**
دستکاری برای یافتن حداقل یک مسیر امن	عدم نیاز به قبضه کردن	ضرورت اطلاع از منابع مورد نیاز آینده
		امکان مسدود شدن طولانی فرایندها

کشف		
طرح	**مزایا**	**معایب**
احضار دوره ای برای بررسی بن بست	عدم تاخیر در آغاز فرایند	ضررهای ذاتی قبضه کردن
	تسهیل پردازش در حین کار	

پیشگیری		
طرح ها	**مزایا**	**معایب**
درخواست یکباره تمام منبع	در مورد فرایندهایی که فعالیت شایعی را انجام می دهند خوب کار می کند.	نا کارآمدی
	عدم نیاز به قبضه کردن	تاخیر شروع فرایند

قبضه کردن	سهولت بکارگیری منابعی که بتوان وضعیت آنها را به سادگی ذخیره کرد.	قبضه کردن بیش از تعداد لازم
		در معرض شروع شدنهای مجدد مدور
مرتب کردن منابع	امکان اعمال کنترلهای زمان ترجمه	قبضه کردن نه چندان مفید
	عدم نیاز به محاسبه در زمان اجرا ، به دلیل اینکه مساله در طرح سیستم حل شده است.	اجازه ندادن به درخواست منابع به صورت افزایشی

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 فرآیند 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؟

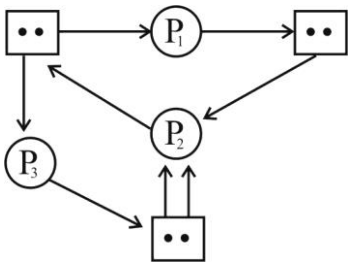
۱) 10 ۲) 5 ۳) 0 ۴) 6

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؟



۱) P_1 و P_3 و P_2 ۲) P_1 و P_2 و P_3

۳) P_2 و P_1 و P_3 ۴) هیچ کدام. زیرا سیستم در وضعیت بن بست قرار دارد.

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 می نامیم. دو نمونه منبع R_1 در اختیار فرایندهای P_1 و P_3 می باشد. فرایند P_2 و P_3 نمی تواند ابتدا اجرا شوند، چون منبع درخواستی آنها آزاد نیست. اما P_1 یک نمونه از R_2 را که آزاد است گرفته و اجرا می شود. بعد از پایان اجرایش، منبع R_1 که در اختیار داشت را رها کرده و P_2 با گرفتن آن اجرا می شود. 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 می شود. بنابراین P_1 ، P_2 و در نهایت P_3 اجرا می شود.

(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 گزینه های زیر صحیح است؟

۱) این روش از بن بست جلوگیری می کند ولی احتمال گرسنگی وجود دارد.

۲) این روش موسوم به درخواست افزایش است و جهت پیشگیری از بن بست به کار می رود.

۳) این روش مبتنی بر کشف بن بست و بدین ترتیب عامل های بن بست تشخیص داده می شوند.

۴) این روش موسوم به درخواست افزایش است و به صورت دینامیکی از بن بست اجتناب می کند.

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 می دهد.

۲) نیاز به پیش بینی آینده دارد.

۳) همه منابع قابل Spool نیست و خود فضای Spool بر روی دیسک نیز موجب بن بست می شود.

۴) هر دو مورد ۱و۲ 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	9	5
P_2	2	6
P_3	2	2
P_4	5	0

حداکثر نیاز

	R_1	R_2
P_1	7	2
P_2	1	3
P_3	1	1
P_4	3	0

مقادیر اختصاص داده شده

۱) امن ۲) نا 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	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

	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

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 بایت ها است. یکی از وظایف سیستم عامل، مدیریت حافظه است. مدیریت حافظه، در سیستم چند برنامه ای باید بسیار کارآمد باشد و حافظه به نحوی باید تخصیص یابد که فرایندهای بیشتری در آن قرار بگیرند.

انواع حافظه عبارتند از: ثبات، حافظه نهان، حافظه اصلی، حافظه دیسک که در فصل اول بررسی شدند. مدیریت ثبات بر عهده کامپایلر، مدیریت حافظه نهان، سخت افزاری، مدیریت حافظه اصلی و مدیریت حافظه دیسک بر عهده سیستم عامل است.حافظه هایی که مدیریت آنها بر عهده سیستم عامل است را در این فصل و فصل های بعد بررسی خواهیم کرد.

مدیریت حافظه ابتدایی

از آنجا که در بعضی سیستم های امروزی، از طرح های ساده مدیریت حافظه، استفاده می شود، این طرح ها را مطالعه می کنیم.

تک برنامگی ساده

ساده ترین طرح مدیریت حافظه این است که در هر لحظه، فقط یک برنامه در حال اجرا باشد. امروزه از طرح تک برنامگی بیشتر در سیستم های توکار استفاده می شود.

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. گرداننده های دستگاه در بالای حافظه و در ROM باشند. (در MS-DOS از این مدل استفاده می شود.)

- از معایب مدیریت حافظه به روش تک برنامگی، اتلاف شدید حافظه و سایر منابع است. همچنین امکان اجرای فرایند بزرگتر از حافظه اصلی وجود ندارد.

چند برنامگی با پارتیشن ثابت و بدون مبادله

برای اجرای چندبرنامگی می توان حافظه را به n پارتیشن تقسیم کرد. اندازه این پارتیشن ها می تواند یکسان نباشد.وقتی کاری وارد می شود، در یک صف ورودی قرار می گیرد تا در کوچکترین پارتیشنی که در آن جا می شود، قرار داده شود. مقداری از پارتیشن که توسط کار درون آن استفاده نمی شود، هدر می رود که به آن حفره (Hole) می گویند.

در یک سیستم با پارتیشن ثابت، هر پارتیشن می تواند، صف مربوط به خود را داشته باشد و یا یک صف مشترک برای همه پارتیشن ها 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 چندبرنامگی با یک تکنیک ساده را فراهم می کند و در مقایسه با تک برنامگی از حافظه و پردازنده بهتر استفاده می شود. اما دارای معایب زیر است:

۱- تعیین تعداد و اندازه پارتیشن ها، مشکل است.

۲- درجه چند برنامگی به تعداد پارتیشن ها محدود است.

۳- نمی توان فرایند بزرگتر از بزرگترین پارتیشن را اجرا کرد.

۴- نمی توان یک فرایند را دو تکه کرد و در دو پارتیشن مجزا قرار داد.

۵- اتلاف حافظه به علت بارگذاری قسمت هایی از یک فرایند بزرگ که فعلا به آنها نیاز نیست.

۶- تکه تکه شدن داخلی

تذکر: به این علت به این تکه تکه شدن، داخلی می گویند که از حفره **داخـل** فضـای تخصـیص یافتـه بـه فرایند ناشی می شود.

جا به جایی و حفاظت

در چندبرنامگی دو مشکل جا به جایی و حفاظت رخ می دهد که باید حل شوند.

جا به جایی(Relocation)

وقتی که برنامه اصلی با زیر برنامه های نوشته شده توسط کاربر و رویـه هـای کتابخانـه ای در یـک فضـای آدرس ترکیب می شوند، مشکلی که به وجود می آید این است که پیوند دهنده از کجـا بایـد بدانـد کـه آن برنامه در زمان اجرا از چه آدرسی در حافظه شروع می شود. برای حل این مشکل، سیستم عامل و سـخت افزار باید بتوانند مراجعات به حافظه را که در کد برنامه مطرح مـی شـوند را بـه آدرس هـای فیزیکـی کـه 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 آدرس ها اصلاح شوند.

حفاظت (Protection)

در سیستم های 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 بلوک های 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

می توان از دو رجیستر پایه(Base) و حد(Limit) واقع در MMU برای حل این دو مشکل استفاده کرد. هنگامی که پردازنده به فرایندی داده می شود، آدرس شروع پارتیشن در Base و طول پارتیشن در Limit قرار می گیرد. برای جلوگیری از مشکل جا به جایی، هر آدرس حافظه ای که تولید می شود، قبل از ارسال به حافظه، مقدارش به صورت خودکار با محتوای Base جمع می شود. برای مشکل حفاظت، آدرس ها با محتوای Limit مقایسه می شوند تا به فضای خارج از پارتیشن دسترسی نشود.

اگر ثبات پایه حاوی 100 و ثبات حد شامل 20 باشد، آن گاه برنامه می تواند به آدرسهایی از 100 تا خود 120 ، دستیابی داشته باشد.

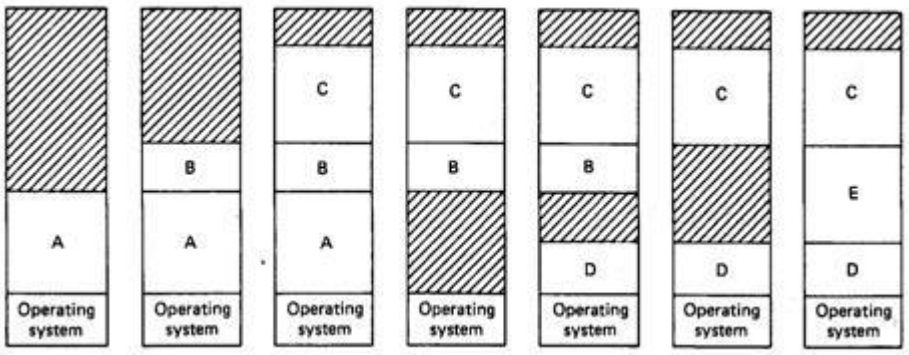
مبادله (swapping)

در سیستم های اشتراک زمانی در گاهی اوقات نمی توان همه فرایندهای فعال را در حافظه اصلی جای داد و باید فرایندهای اضافی به دیسک منتقل شده و بعدا به صورت پویا به داخل حافظه آورده شوند. مبادله (پارتیشن بندی پویا)، ساده ترین روشی است که اجازه مبادله بین حافظه و دیسک را می دهد. در این روش هر فرایند به طور کامل به حافظه اصلی آورده می شود(Swap in) و بعد از مدتی اجرا، به دیسک برگردانده می شود(Swap out).

مثلا در یک محیط چند برنامه ای که از الگوریتم زمانبندی RR استفاده می کند، وقتی هر فرایندی کوانتوم

زمانی خودش را تمام می کند، با فرایند دیگری مبادله می شود.

شکل زیر نحوه عملکرد سیستمی را نشان می دهد که بر اساس مبادله کار می کند. در ابتدا فقط فرایند A در حافظه است و سپس فرایندهای B و C، وارد حافظه شده اند. 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 می گردد. اگر انقیاد در زمان اجرا صورت گیرد، فرایند در محلی غیر از محل اول قرار می گیرد، زیر آدرسهای فیزیکی در زمان اجرا محاسبه می شوند.

- در پارتیشن بندی پویا، تعداد، موقعیت و اندازه پارتیشن ها متفاوت است. این انعطاف پذیری باعث بهبود بهره وری حافظه می شود.

- تخصیص و آزاد سازی حافظه در پارتیشن بندی پویا، پیچیده تر از پارتیشن بندی ایستا است.
- در پارتیشن بندی پویا، به خاطر مبادله، حفره های متعددی در حافظه به وجود می آید که باعث در رفتن حافظه می شود. به این مشکل،تکه تکه شدن خارجی می گویند. علت اینکه به آن خارجی می گویند این است که تکه تکه شدن از حفه خارج فضای تخصیص یافته به فرایندها ناشی می شود.

- برای مقابله با تکه تکه شدن خارجی از فشرده سازی (Compaction) می‌توان استفاده کرد. در این روش، حفره های کوچک با یکدیگر ادغام می شوند تا یک بلوک بزرگ از حافظه را ایجاد کنند. ساده ترین الگوریتم فشرده سازی این است که تمام حفره ها را به یک طرف برده و حفره بزرگی از حافظه آزاد تشکیل می شود. ولی این روش هزینه زیادی دارد.

وقتی که حافظه به صورت پویا تخصیص داده می شود، 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 می دهد 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 می شود. 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 (l) یا خالی (0) به ازاء هر 4KB حافظه اصلی است. لیست حفره ها به چه صورت است؟

پاسخ:

با نگاه به bitmap داده شده از چپ به راست، سه دسته صفر داریم :

11001011, 00000111

۱- دسته اول: دو صفر (8KB)

۲- دسته دوم: یک صفر (4KB)

۳- دسته سوم: پنج صفر (20KB)

بنابراین لیست حفره ها 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 می باشد.

۲- مدیریت حافظه با لیست های پیوندی

در این روش،یک لیست پیوندی از قطعه های آزاد یا تخصیص یافته حافظه تشکیل می شود. مزیت این روش این است که زمانی که فرایندی خاتمه می یابد، یا مبادله می شود، این لیست به سادگی update می شود. فرایندی که پایان یافته با همسایه های خود ترکیب می شود. فیلدهای هر گره در لیست پیوندی مثال قبل شامل فیلدهای زیر است:

۱- حفره (H) یا فرایند(P) بودن

۲- آدرس شروع

۳- طول

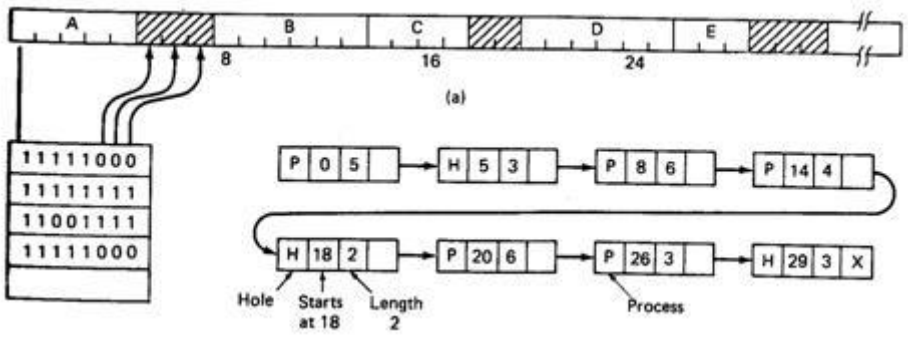
۴- آدرس گره بعدی

مثال

شکل زیر قسمتی از حافظه شامل 5 فرایند و 3 حفره را نشان می دهد. 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 شده اند. 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 می باشد.

در طرح لیسست پیوندی، فضاهای خالی با H و فضاهای پر با P مشخص شده اند.



الگوریتم های مکان یابی و تخصیص حافظه

وقتی فرایندها و حفره ها در یک لیست مرتب شده بر اساس آدرس قرار می گیرند، الگوریتم های مختلفی جهت تخصیص حافظه به یک فرایند وجود دارد. چند مورد از این الگوریتم ها عبارتند از:

۱- اولین برازش (First fit)

جستجو از ابتدای حافظه شروع شده و فرایند در اولین حفره ای قرار داده می شود که در آن جا می شود.

۲- برازش بعدی(Next fit)

مانند First fit است، با این تفاوت که جستجو از آخرین محل تخصیص شروع می شود.

۳- بهترین برازش (Best fit)

تمام لیست جستجو می شود و فرایند در کوچکترین حفره ای قرار داده می شود که در آن جا می شود.

۴- بدترین برازش (Worst fit)

تمام لیست جستجو می شود و فرایند در بزرگترین حفره ای قرار داده می شود که در آن جا می شود.

- الگوریتم First fit ، ساده ترین روش و معمولاً سریع ترین الگوریتم است.
- الگوریتم First fit ، بیشترین کارایی را دارد. یعنی بهره وری 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 می‌شود، حافظه اصلی شامل فضاهای خالی با اندازه‌های 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 و10 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 می‌شوند؟

پاسخ: روش First fit

تکه 12 از حفره 20 گرفته می‌شود و 8 کیلو باقی می ماند و تکه 10 از حفره 10 و تکه 6 از حفره 8 (باقیمانده از ۲۰) گرفته می‌شود:

	10	4	20	18	7	9	12
12	10	4	**8**	18	7	9	12
10	0	4	8	18	7	9	12
6	0	4	2	18	7	9	12

روش Next fit:

	10	4	20	18	7	9	12
12	10	4	**8**	18	7	9	12
10	10	4	8	**8**	7	9	12
6	10	4	8	**2**	7	9	12

توجه کنید که فرایند 6KB از همان فضای باقی مانده از حفره مرحله قبل استفاده کرد.

روش Best fit:

	10	4	20	18	7	9	12
12	10	4	20	18	7	9	0
10	0	4	20	18	7	9	0
6	0	4	20	18	1	9	0

روش Worst fit:

	10	4	20	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 می‌باشند.

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:

<table>
<tr><td></td><td colspan="6">1024</td></tr>
<tr><td></td><td colspan="5">512</td><td>512</td></tr>
<tr><td></td><td colspan="3">256</td><td colspan="2">256</td><td>512</td></tr>
<tr><td>Request 70</td><td>A</td><td colspan="2">128</td><td colspan="2">256</td><td>512</td></tr>
<tr><td>Request 35</td><td>A</td><td>B</td><td>64</td><td colspan="2">256</td><td>512</td></tr>
<tr><td>Request 80</td><td>A</td><td>B</td><td>64</td><td>C</td><td>128</td><td>512</td></tr>
<tr><td>Return A</td><td>128</td><td>B</td><td>64</td><td>C</td><td>128</td><td>512</td></tr>
<tr><td>Request 60</td><td>128</td><td>B</td><td>D</td><td>C</td><td>128</td><td>512</td></tr>
<tr><td>Return B</td><td>128</td><td>64</td><td>D</td><td>C</td><td>128</td><td>512</td></tr>
<tr><td>Return D</td><td colspan="3">256</td><td>C</td><td>128</td><td>512</td></tr>
<tr><td>Return C</td><td colspan="6">1024</td></tr>
</table>

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 کـوچکتر بـود، از تکنیـک Overlay استفاده می شد. در این تکنیک برنامه نویس، برنامه ها را به قسمت هایی به نام Overlay تقسیم مـی کنـد. 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 همزمـان در حافظـه وجـود داشـت، سیسـتم عامل Overlay ها را روی دیسک نگه می داشت و در مواقع لزوم، عملیات مبادله را انجـام مـی داد. تقسـیم برنامه به overlay ها و فراخوانی آنها، بر عهده برنامه نویس بود و سیستم عامل فقط عمل مبادلـه را انجـام می دهد.

صفحه بندی (Paging)

در این روش مدیریت حافظه، حافظه اصلی به بلوکهایی با اندازه های ثابت به نام قاب (frame) تقسیم می شود. حافظه منطقی نیز به بلوک هایی با اندازه های 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 قاب های آزاد به صفحه‌های فرایند A با سه صفحه را نشان می‌دهد. جدول صفحه را مشخص نمایید.

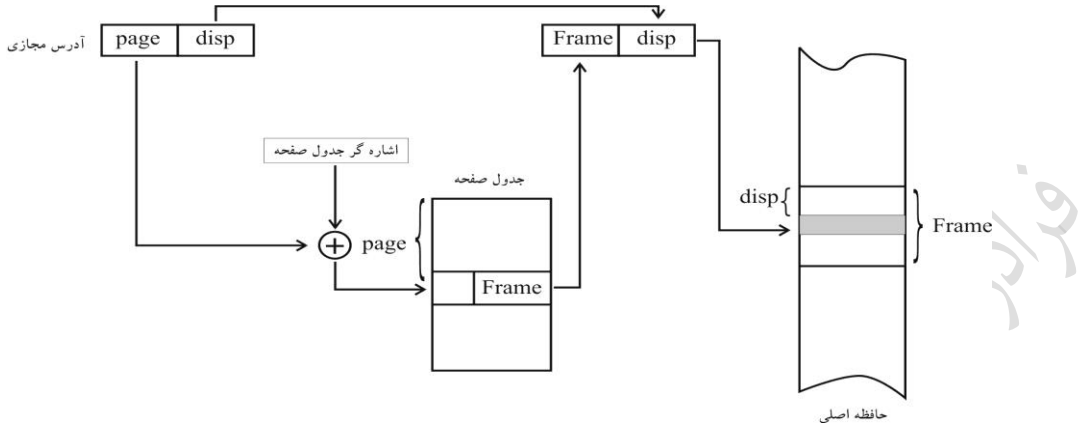
	⋮
5	A.0
6	
7	A.2
8	A.1
9	

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 می گویند.
- اندازه صفحه و اندازه قاب توسط سخت افزار تعریف می شود.
- 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:

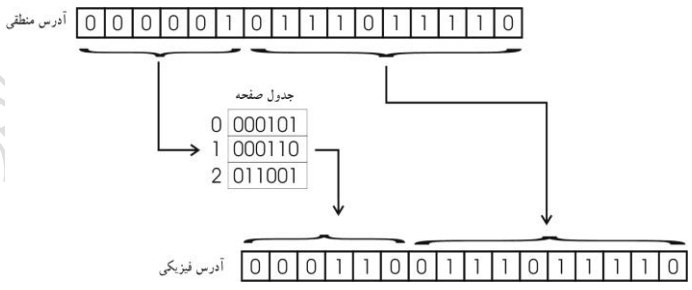


نحوه تولید آدرس فیزیکی با توجه به آدرس منطقی در شکال بالا: شماره صفحه موجود در آدرس مجازی(page) با آدرس شروع جدول صفحه، جمع شده و به آدرس بدست آمده، در جدول صفحه مراجعه می شود. در این آدرس شماره قاب (frame) متناظر با صفحه مورد نظر قرار دارد. این شماره قاب و همان آفست(disp) موجود در آدرس مجازی، آدرس فیزیکی را مشخص می کنند.

مثال

با توجه به آدرس نسبی 16 بیتی 0000010111011110 ، چند بیت برای شماره صفحه نیاز است؟(اندازه صفحه برابر یک کیلو بایت می باشد.)

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 می دهد:



تذکر: 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؟

فضای آدرس دهی منطقی = $4 \times 2 = 8 = 2^3$

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:

اندازه حافظه منطقی = $64 \times 512 = 2^6 \times 2^9 = 2^{15}$ بایت

اندازه حافظه فیزیکی برابر است با حاصل ضرب تعداد آدرسها در اندازه هر صفحه:

اندازه حافظه فیزیکی = $2^{10} \times 512 = 2^{19}$ بایت

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 کاملا 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 می‌سازد و کاربر را از محدودیتهای حافظه اصلی رها می‌کند. 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 می کند. این تفکیک موجب می شود در حالتی که حافظه فیزیکی کم است، حافظه مجازی بزرگی برای برنامه نویس فراهم شود.

حافظه مجازی، برنامه نویسی را ساده می کند، زیرا برنامه نویس نگران حافظه فیزیکی و کد جایگذاری نمی باشد.

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 برمی گرداند.

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 می توان 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 می شوند.

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 بر خلاف 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 می شوند. این روش تلفیقی از صفحه بندی و مبادله است.

آدرس هایی که توسط برنامه ها تولید می شوند(آدرس مجازی)، به واحد MMU که در درون پردازنده قرار دارد، منتقل می شوند. توسط این واحد، عمل ترجمه(نگاشت) آدرس مجازی به آدرس فیزیکی انجام می شود. 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 می کند.

اگر MMU به جدول صفحه مراجعه کند و بیت حضور را برای صفحه مورد نظر صفر ببیند، متوجه می شود که صفحه بر روی دیسک است. دراین حالت پردازنده در تله(trap) سیستم عامل می افتد. به این تله، خطای نقص صفحه (Page Fault) می گویند 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	این بیت ها نوع دسترسی مجاز را مشخص می کند. در حالت سه بیتی(rwx) ، یک

(Protection)	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 (Cashing 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 مدخل های 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؟

پاسخ: $\frac{512KB}{32KB}=16$

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 بایتی باشند، آدرسهای

ماشینی که بتواند این حافظه مجازی را مستقیماً آدرس دهی کند، باید چند بیتی باشند؟

پاسخ:

$$e = \frac{8\text{MB}}{16} = \frac{8 \times 2^{20}}{16} = 2^{19}$$

$$2^{19} \times (8kb) = 2^{19} \times (8 \times 2^{10}) = 2^{32}$$

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

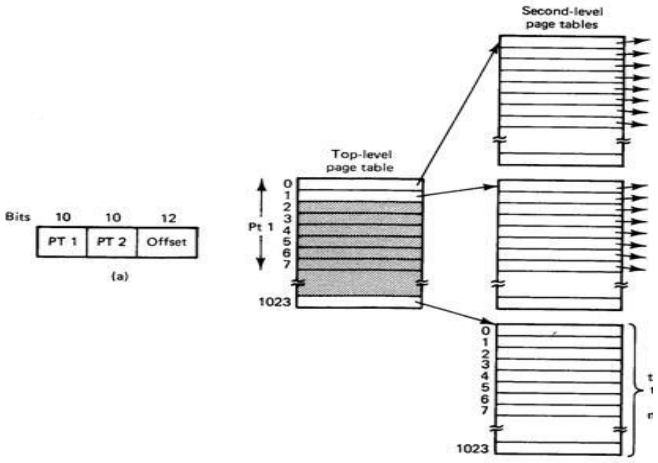
۱- جداول صفحه چند سطحی ۲- جدول صفحه وارونه(معکوس)

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 $\frac{2^6}{4} = 2^4$ است. بنابراین اندیس های PT_i برابر 4 بیت است. در نتیجه تعداد سطوح مورد نیاز برابر $\left\lceil \frac{12}{4} \right\rceil = 3$ 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، دقیقا 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 $(\frac{2^4}{2} = 2^3)$ 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 روش های 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 درایه ها جدول صفحه معکوس، مشخص می کند 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 می کند، ولی تبدیل آدرس مجازی به فیزیکی بسیار مشکل تر می باشد. برای سریع کردن جستجو، از یک جدول درهم (Hash Table) استفاده می شود. قسمت شماره صفحه از آدرس مجازی به کمک یک تابع درهم ساز به یک جدول درهم نگاشت می شود. جدول درهم دارای نشانگری به جدول صفحه معکوس است که شامل مدخلهای جدول صفحه می باشد. 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).

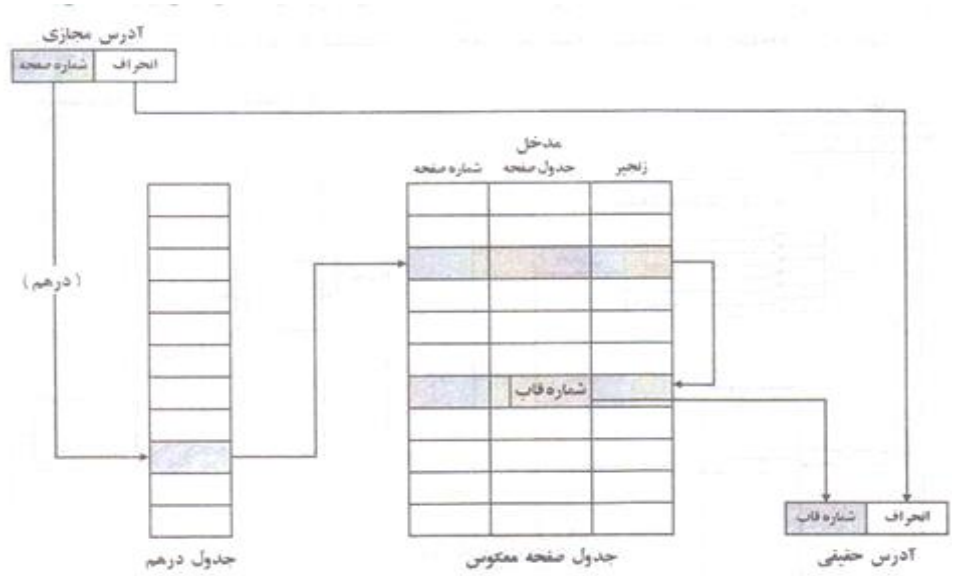
پس جدول صفحه معکوس:

۱- سبب کاهش اندازه حافظه فیزیکی جهت ذخیره سازی آن می شود.

۲- باید یک جدول صفحه خارجی نیز برای آن ذخیره شود.

۳- زمان سرویس نقص صفحه افزایش می یابد.(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 وP=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 چهـار حالـت 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.

الگوریتم های 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 ۳ قاب پر شده است، باید یکی از صفحه های 7 , 0 و1 ، 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		

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 ، اخیرا مورد استفاده قرار نگرفته و تغییر نکرده است. به همین علت بهترین انتخاب می باشد.

تذکر: 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

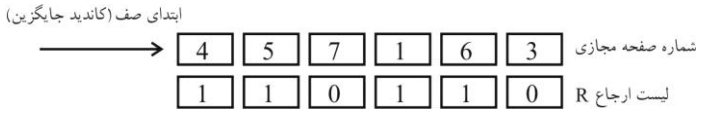
الگوریتم دومین شانس

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 و5 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 آن ها 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 قدیمی ترین صفحه یعنی صفحه 3 است. چون بیت R آن 1 است، بیت R آن 0 شده و عقربه به جلو حرکت می کند و بر روی صفحه 2 می رود(صفحه بعدی بر اساس زمان بارگذاری) . چون بیت R صفحه 2 ، صفر است، صفحه جدید با این صفحه جایگزین می شود.

الگوریتمی به نام WS-Clock هست که در آن وقتی عقربه به صفحه ای می رسد، که عضوی از مجموعه کاری (Working Set) فرایند باشد، انتخاب نمی شود و عقربه به جلو می رود. برای پیاده سازی این روش از فیلد سن (Age) جدول صفحه استفاده می شود.

(مجموعه صفحاتی که فرایند در حال حاضر از آنها استفاده می کند، مجموعه کاری نام دارد.)

الگوریتمLRU

در روش (LRU(Least Recently Used ، صفحه ای جایگزین می شود 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 ۳ قاب پر شده است، باید یکی از صفحه های 7 , 0 و1 ، را از قابها خارج کرد، که همان صفحه 7 است، چون نسبت به صفحه های 0 و1 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 ۴ قاب داریم، پس یک ماتریس 4×4 بیتی با عناصر صفر در نظر می گیریم. اولین درخواست به صفحه 0 است. بنابراین ابتدا همه بیت های سطر شماره صفر را 1 می کنیم. سپس همه بیت های ستون شماره صفر را 0 می کنیم(شکل a). درخواست بعدی شماره 1 است. در ماتریس شکل a ، ابتدا سطر یک را 1 کرده و سپس ستون یک را 0 می کنیم (شکل b) . و...

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

(a)

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

(b)

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

(c)

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

(d)

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

(e)

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

(f)

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

(g)

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

(h)

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

(i)

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

(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 می کند. اگر بخواهیم الگوریتم کمترین استفاده در گذشته اخیر(LRU) را با روش ماتریس دو بعدی تقریب بزنیم، 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 می شود، پس ابتدا تعداد قاب ها را محاسبه می کنیم:

$$\frac{256MB}{4KB}=\frac{2^8\times2^{20}}{2^2\times2^{10}}=\frac{2^{28}}{2^{12}}=2^{16}$$

بنابراین ماتریس مورد نیاز $2^{16}\times2^{16}$ بیتی است.

نحوه تبدیل $2^{16}\times2^{16}$ 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.

■

شبیه سازی LRU در نرم افزار (الگوریتم سالمندی)

در الگوریتم سالمندی (Aging)، از فیلد سن(age) در جدول صفحه استفاده می شود که مثلا 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 پریود زمانی، ابتدا این فیلد به اندازه یک بیت به 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 100دوره قبل، نمی توان از این موضوع آگاه شد. در اینصورت یک صفحه به صورت تصادفی انتخاب می شود.

الگوریتم های 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 می شود.

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 می شوند و با مراجعه به آنها نقص صفحه رخ می دهد. در این روش سه لیست صفحه داریم:

۱- درون حافظه ۲- درون بافر(بدون تغییر) ۳- درون بافر(تغییر یافته)

این لیست ها، به ترتیب ورود مرتب می باشند. هنگامی که صفحه ای با الگوریتم FIFO ، از ابتدای لیست اول برای جایگزینی انتخاب می شود، از لیست اول خارج شده و به انتهای لیست دوم یا سوم اضافه می شود. این صفحه واقعا از حافظه خارج نمی شود(در جدول صفحه به عنوان غایب علامت گذاری می شود)و به جای آن یک صفحه از ابتدای لیست دوم و یا سوم خارج شده و واقعا از حافظه حذف می شود.

یکی از نقطه ضعف های الگوریتم FIFO ، اخراج صفحات قدیمی پر استفاده است که در الگوریتم بافر کردن صفحه برطرف شده است. چون اگر صفحه ای با استفاده زیاد، به طور ظاهری اخراج شود، به زودی با نقص صفحه مواجه شده و به سرعت این صفحه از بافر بر گردانده می شود.

نکات طراحی سیستم های صفحه بندی

تعداد کل قاب صفحاتی که می توان در یک سیستم به مجموعه فرایندها داد، مقداری ثابت و متناسب با اندازه حافظه اصلی است. کاهش تعداد قاب ها باعث می شود که یک فرایند نتواند مجموعه صفحاتی که در طی یک زمان از آنها استفاده می کند را در حافظه اصلی بارگذاری کند. در نتیجه تعداد نقص صفحه ها زیاد می شود. در این وضعیت، زمان CPU به جای اجرای فرایندها، صرف مبادله صفحه ها می شود و کارایی سیستم کاهش می یابد. به این پدیده، کوبیدگی (Thrashing) می گویند.

از مشکل های دیگر، می توان 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 می شود. بنابراین به دنبال راه حل هایی هستیم تا از این نوع مشکلات به وجود نیاید.

پیش صفحه بندی (prepaging)

سیاست واکشی 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: $W(t,k)=\{1,2,5\}$

■

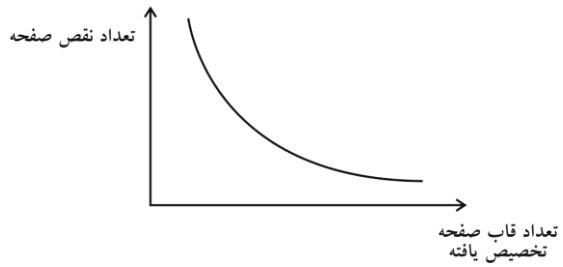
الگوریتم فرکانس نقص صفحه (PFF)

در روش (PFF(Page Fault Frequency ، کران بالا و پایین برای نرخ خطای صفحه تعیین می شود. اگر نرخ خطای صفحه از کران بالا، بیشتر شود، قاب دیگری به آن فرایند تخصیص داده می شود و اگر نرخ خطای صفحه از کران پایین کمتر شود، قابی از فرایند پس گرفته می شود.

بنابراین PFF سعی می کند 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 می گویند. تناقض بلیدی در الگوریتم های BO,LRU رخ نمی دهد. 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 می دهد و اگر اندازه حافظه را به چهار صفحه افزایش دهیم، تعداد ده نقص صفحه رخ می دهد. به این موضوع تناقض FIFO می گویند. 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 می دهد:

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 می‌کند، چنانچه اندزه 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: حلقه های داده شده، ماتریس را به صورت ستون به ستون صفر می کند. از آنجا که ماتریس داده شده دو صفحه را اشغال می کند $(\frac{4\times 4}{8}=2)$، و هر دو سطر آن یک صفحه است، در هر با خواندن صفحه (دو سطر)، فقط دو خانه از یک

ستون مقدار دهی می شوند و برای مقدار دهی به هر دو خانه، یک نقص صفحه رخ می دهد و تعداد کل نقص صفحه ها برابر است با: $\frac{4\times4}{2}=8$

قطعه بندی

در روش قطعه بندی برای مدیریت حافظه، برنامه و داده ها 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 می شود. هر سطر این جدول شامل آدرس شروع قطعه مورد نظر در حافظه اصلی و طول قطعه می باشد.

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 داده ها و کد می باشد.

۳- الگوی صفحه بندی نمی تواند 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.

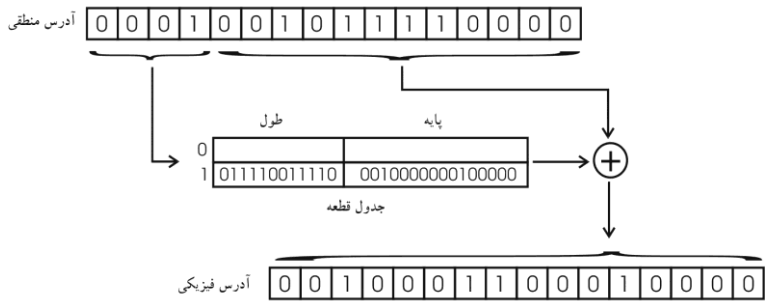
۶- قطعه بندی تکه تکه شدن داخلی را حذف می‌کند، اما دارای تکه تکه شدن **خارجی** است.

۷- تکه تکه شدن خارجی هم در بخش بندی پویا وهم 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، هرفرایند جدول قطعه خودش را دارد و هرگاه تمام قطعه‌های مربوط به آن فرایند به داخل حافظه اصلی بارشدند، جدول قطعه برای آن فرایند ایجاد و داخل حافظه بار می‌شود. اما در حافظه

مجازی با قطعه بندی ، لزومی ندارد تمام قطعه های یک فرایند درحافظه اصلی باشند و می‌توانند برحسب نیاز به داخل خوانده شوند. هر مدخل جدول قطعه شامل موارد زیر است:

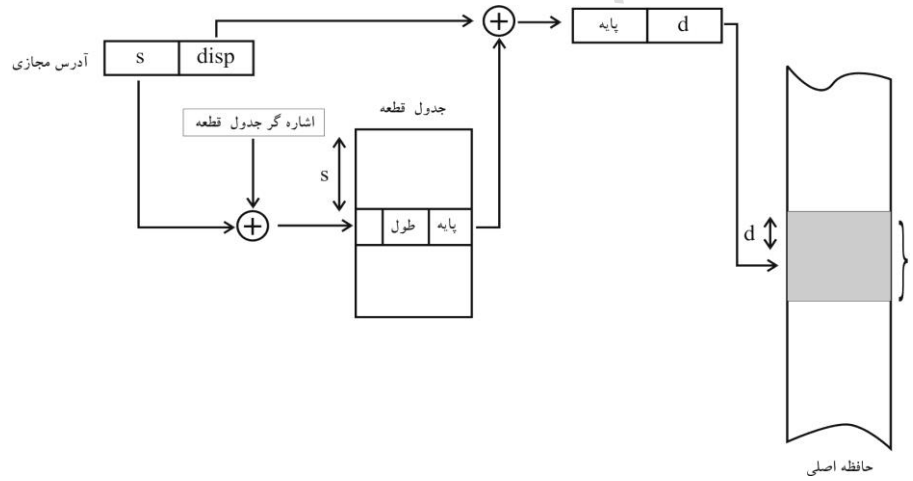
۱- طول قطعه ۲- پایه قطعه ۳- بیت حضور (P)

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 می دهد:



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:

شماره قطعه	شماره صفحه	آفست درون صفحه

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 بایتی 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 (همان101 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 [4678H , 01H] 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 بایتی است. بایت اول و دوم نشان دهنده آدرس پایه جدول صفحه(PTBA) و بایت سوم مسخص 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:

شماره قطعه	شماره صفحه	آفست

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:

شماره قاب	آفست	$\Rightarrow$	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	قطعه بندی ساده	صفحه بندی حافظه مجازی	قطعه بندی حافظه مجازی
حافظه اصلی به تکه های هم اندازه به نام قاب تقسیم می شود.	حافظه اصلی تقسیم نمی شود.	حافظه اصلی به تکه های هم اندازه به نام قاب تقسیم می شود.	حافظه اصلی تقسیم نمی شود.
برنامه توسط مترجم یا سیستم مدیریت حافظه تقسیم می شود.	قطعه های برنامه توسط برنامه ساز به مترجم اطلاع داده می شوند.یعنی تصمیم گیری با برنامه ساز است.	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.
بدون تکه تکه شدن خارجی	بدون تکه تکه شدن داخلی	بدون تکه تکه شدن خارجی درجه چندبرنامگی بالاتر فضای مجازی بزرگ برای فرایند	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 خارجی گسترش به کار گیری حافظه کاهش سربار نسبت به بخش بندی پویا	تکه تکه شدن داخلی درون قاب سربار پیچیدگی مدیریت حافظه	تکه تکه شدن خارجی سربار پیچیدگی مدیریت حافظه
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.	سیستم عامل باید فهرست حفره های آزاد در حافظه اصلی را نگهداری کند.
پردازنده از شماره صفحه و انحراف برای محاسبه آدرس مطلق استفاده می کند.	پرادزنده 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.	پردازنده از شماره قطعه 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.	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.	لزومی ندارد تمام صفحه های یک فرایند در قابهای حافظه اصلی باشند تا فرایند اجرا شود، صفحه ها می توانند بر حسب نیاز به داخل خوانده شوند.	لزومی ندارد تمام قطعه های یک فرایند در حافظه اصلی باشند تا فرایند اجرا شود، قطعه ها می توانند بر حسب نیاز به داخل خوانده شوند.
		خواندن یک صفحه به داخل	خواندن یک صفحه به داخل

		حافظـه اصـلی،ممکن اسـت نیازمند نوشتن یـک صـفحه بر روی دیسک باشد.	حافظه اصلی، ممکـن اسـت نیازمند نوشتن یک قطعه 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 کیلوعضو

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 پردازه A شروع و در انتهای این پردازه پایان می پذیرد. اندازه پردازه 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؟

procedure A; . call D; . call F; . end;	procedure B; . . . . . end;	Procedure C; . . . . . end;
procedure D; . call E; . call B; end;	procedure E; . . . end;	Procedure F; . call C; . end;

۱) 31 K ۲) 27 K ۳) 29 K ۴) 30 K

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 پردازه 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 $A \to D \to B$ 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 عنصر16 بیتی (شامل بیت نامعتبر/ معتبر (valid/invalid) است و اندازه صفحات1 کیلوبایتی است، به ترتیب اندازه فضای آدرس فیزیکی چقدر است و آدرس فیزیکی چند بیتی است؟

۱) $26 - 2^{26}$ ۲) $25 - 2^{25}$ ۳) $24 - 2^{24}$ ۴) $16 - 2^{16}$

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 قاب صفحه باقی می ماند. پس تعداد بیت های آدرس فیزیکی برابر 25=10+15 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 بایتی و صفحات 4 کیلو بایتی است. جدول صفحه چند مدخل (Table Entry) دارد؟

۱) 2^{8} ۲) 2^{12} ۳) 2^{20} 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 85% از ارجاعات به حافظه از طریق TLB انجام شود و هزینه هر ارجاع حافظه 250 نانو ثانیه و ارجاع به TLB با هزینه 5 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) 287.5 ثانیه ۲) 292.5 نانو ثانیه ۳) 291.75 ثانیه ۴) 505 نانو ثانیه

حل: جواب گزینه ۲ است.

همان حالت اول است:

$$T_{Translation}=T_{TLB}+(1-H)\times T_{Mem}=5+(0.15\times 250)=42.5\ \ ns$$

$$T_{Access}=T_{Translation}+T_{Mem}=42.5+250=292.5ns$$

(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 50ns و زمان خواندن از TLB برابر 20ns باشد و درصد کارایی سیستم بدون استفاده از TLB نسبت به استفاده سیستم از TLB برابر 80 درصد باشد، آن گاه نرخ برخورد TLB چقدر است؟

۱) 10 درصد ۲) 2 درصد ۳) 80 درصد ۴) 90 درصد

حل: گزینه ۳ جواب است.

اگر از TLB استفاده نشود:

$$T_{Access}=T_{Translation}+T_{mem}=2\times T_{Mem}=2\times 50ns=100ns$$

و اگر از TLB استفاده شود:

$$T_{Access}=[T_{TLB}+(1-H)\times T_{Mem}]+T_{Mem}=[20+(1-H)\times 50]+50$$

و چون درصد کارایی برابر 80% است، داریم:

$$\frac{70+(1-H)\times 50}{100}=\frac{80}{100}\quad\Rightarrow\quad H=0.8\Rightarrow H=80\%$$

(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 ۲و 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 نهان و با در نظر گرفتن احتمال وقوع نقص صفحه:

$$T_{Access} = T_{Translation} + T_{CM} + P \times T_{Disk}$$

$$T_{Translation} = T_{TLB} + (1-H) \times T_{CM}$$

$$T_{CM} = T_{Cache} + (1-H_{Cache}) \times T_{Penalty}$$

جریمه هر عدم اصابت در حافظه پنهان : $T_{Penalty}$

(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؟

۱) 75ns ۲) 45ns ۳) 73ns ۴) 43ns

پاسخ: جواب گزینه ۴ است.

$$T_{Translation} = T_{TLB} + (1-H) \times T_{CM} = 2 + (1-0.98) \times 20 = 2.4ns$$

$$T_{CM} = T_{Cache} + (1 - H_{cache}) \times T_{Penalty} = 10 + (1 - 0.9) \times 100 = 20\text{ns}$$
$$T_{Access} = T_{Translation} + T_{CM} + P \times T_{Disk} = 2.4 + 20 + (2 \times 10^{-6})(10 \times 10^{-3}) = 43\text{ns}$$

■

(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 جائی FIFO استفاده شود، تعداد خطاهای صفحه (Page Faults) برابر است با:

۱) 10 ۲) 12 ۳) 14 ۴) 13

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 (Reference) R 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 319 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 قاب ها خواسته شده است. بنابراین چون صفحه 1 در قاب 1 و صفحه 2 در قاب0 قرار دارد، گزینه ۲ جواب است.

تذکر: چون در زمان رخ دادن خطای صفحه، یعنی 319 ، همه صفحه ها موجود هستند، تصمیم گیری 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 می کند. اگر بخواهیم الگوریتم کمترین استفاده در گذشته اخیر(LRU) را با روش ماتریس دو بعدی تقریب بزنیم، 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 قاب ها را مشخص کرد:

$$\frac{128MB}{4KB}=\frac{128\times 2^{20}}{4\times 2^{10}}=2^{15}$$

بنابراین ماتریس مورد نیاز $2^{15}\times 2^{15}$ بیتی یا 128 مگا بایتی است.

نحوه تبدیل به بایت:

$$\frac{2^{15}\times 2^{15}}{8}=2^{27}=2^{7}\times 2^{20}=128MB$$

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 می باشد. در این صورت کدام یک از شرایط زیر را لزوما 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) تقریبا 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 الزما صفر) است. پس گزینه های ۱و۲ نادرست هستند. همچنین در صد عملیات مبادله صفحات بین حافظه اصلی و دیسک بسیار بالا حتی نزدیک به 100% (نه الزما 100%) می باشد. بنابراین گزینه ۴ که از عبارت "تقریبا" استفاده کرده، صحیح می باشد.

■

(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 نقص صفحه وجود داشته باشد. یعنی با افزایش تعداد قاب، تعداد نقص 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;
```

۱) 65 ۲) 8193 ۳) کمتر از 65 ۴) 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 $\left\lceil \frac{64}{3} \right\rceil$ یعنی 22 نقص صفحه برای کد رخ می دهد.

بنابراین در کل تعداد نقص صفحه ها برابر $22+64$ 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 برابر128 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:

اندازه صفحه $= 2^{13} = 8K$

اندازه حافظه مجازی هر فرایند $= 2^{(9+5+7+13)} = 2^{34} = 2^4 \times 2^{30} = 16G$

حداکثر اندازه هر قطعه $= 2^{(5+7+13)} = 2^{25} = 2^5 \times 2^{20} = 32M$

حداکثر تعداد جداول صفحه سطح یک در هر فرایند $= 2^9 = 512$

حداکثر تعداد صفحات در هر قطعه $= 2^{(5+7)} = 2^{12} = 2^2 \times 2^{10} = 4K$

حداکثر تعداد صفحات در هر فرایند $= 2^{(9+5+7)} = 2^{21} = 2 \times 2^{20} = 2M$

فصل ۷

مدیریک 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

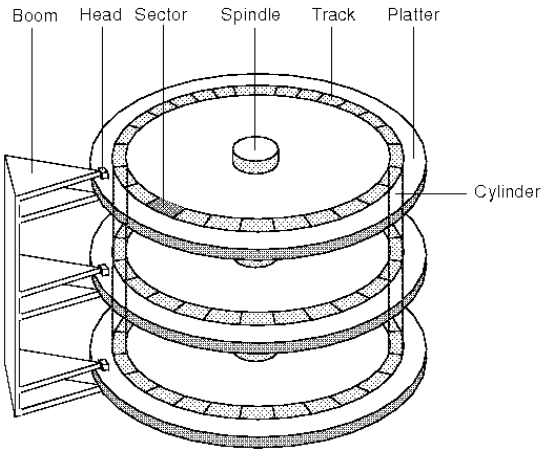
یکی از وظایف سیستم عامل، مدیریت دیسک می باشد. ابتدا دیسک مغناطیسی را تشریح می کنیم.

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 می‌دهند.

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 می‌دهند 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:

$$2r = \frac{60000}{rpm}$$

که rpm سرعت چرخش دیسک است و واحد آن دور در دقیقه است.

متوسط زمان درنگ دورانی از رابطه زیر محاسبه می شود:

$$r = \frac{30000}{rpm}$$

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 می کند. تذکر: نام دیگر روش SCAN ، روش LOOK می باشد.

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 ها ی 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 سیلندرهای به ترتیب ۱۰، ۲۲، ۲۰، ۲، ۴۰، ۶ و ۳۸ به Driver آن وارد شود و چنانچه حرکت هدI/O بین دو سیلندر مجاور ۶ میلی ثانیه طول بکشد، در صورت استفاده از الگوریتم SSTF برای خواندن سیلندرهای، کلseek time مورد نیاز چقدر خواهد بود؟

حل: در SSTF همواره به سمت نزدیکترین سیلندر حرکت می شود:

$$20 \rightarrow 22 \rightarrow 10 \rightarrow 6 \rightarrow 2 \rightarrow 38 \rightarrow 40$$

مجموعه فاصله‌های برابر است با:

$$2 + 12 + 4 + 4 + 36 + 2 = 60$$

و چون هر حرکت ۶ میلی ثانیه طول می‌کشد، پس در کل 6 × 60 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 فلش ها، و ضرب در 4 حاصل 712 می شود.

- عدالت فقط در روش FCFS رعایت می شود.
- روشهای FCFS ، Scan و C-Scan ، بدون قحطی هستند.
- کارایی(میانگین زمان جستجو) در FCFS پایین است.
- مراجعات در FCFS محلی می باشد.
- 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 درخواست های هر صف به روش Scan پاسخ داده می شود. در زمانی که صفی پردازش می شود، درخواست های جدید به صف های دیگر وارد می شوند.
- الگوریتم N-Setp_Scan با دو صف را الگوریتم F-Scan می گویند.
- یکی از تکنیکهای کاهش زمان درنگ دوران، استفاده از روش تداخل بلاکها (Interleaving) می باشد. در این تکنیک، بلاکها به صورت n در میان روی شیار چیده می‌شوند.

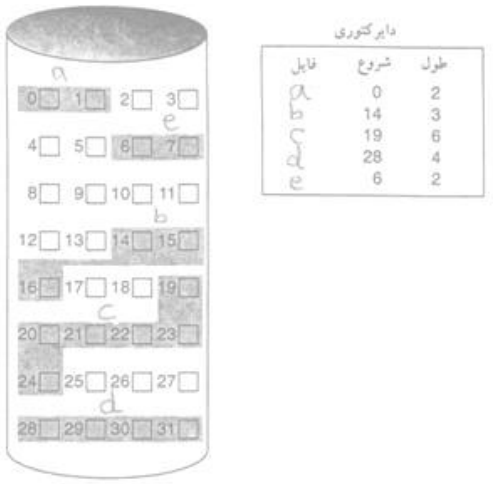
روش های تخصیص فضای دیسک به فایل

فضای دیسک به سه روش "پیوسته ، پیوندی و شاخصی" می تواند به فایل تخصیص داده شود. هدف روش های 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 می کند و کافی است که شماره بلاک اول روی دیسک و تعداد بلاک های فایل را ذخیره کرد.

شکل زیر این نوع تخصیص را نشان می دهد:



۲- تخصیص پیوندی

در این روش هر فایل، یک لیست پیوندی از بلاک های روی دیسک است. بلاک ها ممکن است در هر کجای دیسک پراکنده باشند. در فهرست راهنما برای هر فایل اشاره گری به اولین بلاک فایل قرار دارد.

۳- تخصیص شاخصی

در روش شاخصی، اشاره گرها به بلاکهای فایل روی دیسک در یک مکان که به آن بلاک شاخص می گویند، جمع آوری می شوند. هر فایل دارای بلاک شاخص خود است که یک ماتریس از آدرسهای بلاکها است. ورودی Iام در بلاک شاخص به بلاک I ام فایل اشاره می کند.

در فهرست راهنما تنها آدرس بلاک شاخص حفظ می شود. در این روش براحتی می توان دستیابی مستقیم را حمایت کرد. البته فضای بلاک شاخص تلف می شود و در بسیاری از مواقع به تمامی بلاک شاخص نیاز نمی باشد. علت پر طرفدار بودن روش شاخصی، رابطه نزدیک آن با مدیریت حافظه قطعه بندی– صفحه بندی شده است. بلاک شاخص می تواند یک جدول صفحه باشد و بلاک های 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 می کند. 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 پیوند ها

۳- عدم قابلیت اطمینان سیستم

با از بین رفتن تنها یکی از اشاره گرها، صدمات جدی به فایل و فضای روی دیسک وارد می گردد.

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 می سپارد و در صورت لزوم سراغ بلاکهای بعدی می رود.

۲- مستقیم

برای دستیابی مستقیم به بلاک i فایل، کافی است که به بلاک b+i دستیابی صورت گیرد. (با این فـرض که فایل از بلاک b دیسک آغاز شده باشد.)

مثال

فایلی دارای 5 بلوک از شماره 1 تا 5 می باشد. می خواهیم 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 می شود. بنابراین به 2 دسترسی دیسک نیاز است.

ب- **پیوندی**: چون هر بلوک فایل حاوی اشاره گری است که آدرس بلوک بعدی را مشخص می کند، از بلوک 1 تا 4 خوانده تا آدرس بلوک 5 را به دست آوریم. سپس این آدرس را جایگزین آدرس بلوک 3 می کنیم. 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 اندیس ها را خوانده و آدرس بلوک 4 را حذف کرده و بعد از به روز رسانی، بر روی دیسک باز نویسی می کنیم. 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 برابر0.5 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)=

$= (5+2) + (4 \times 0.5) = 9ms$

ب- اگر مستقیما از دیسک به حافظه کوچک منتقل می کردیم 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)

$4 \times (5 + 0.5) = 22ms$

کنکور ارشد

(مهندسی کامپوتر- 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؟

۱) 21.5 ۲) 68 ۳) 50 ۴) 55

پاسخ: جواب گزینه ۱ است.

زمان میانگین تاخیر چرخشی (r) :

$$2r = \frac{60000}{RPM} \Rightarrow 2r = \frac{60000}{12000} \Rightarrow r = 2.5m\sec$$

تعداد سکتورهای مورد نیاز برای ذخیره فایل:

$$\frac{1MB}{512} = 2048$$

زمان خواندن یک سکتور: (زمان خواندن یک شیار(2r) تقسیم بر تعداد سکتور در هر شیار)

$$\frac{5}{256} = 0.02m\sec$$

در نهایت زمان خواندن یک فایل با n سکتور به صورت تصادفی برابر است با:

$$2048\times(8+2.5+0.02) = 21.5\sec$$

(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.

زمان ورود درخواست	0	10	20	70	80	90
سیلندر درخواست شده	21	75	16	68	2	17

در زمان حرکت هد به سمت یک سیلندر، ورود درخواست جدید تاثیری بر حرکت ندارد. ترتیب اجرای درخواست ها برای الگوریتم SCAN(آسانسور) چیست؟

۱) 0 21 75 16 2 17 68
۲) 0 21 75 16 17 2 68
۳) 0 21 75 16 68 2 17
۴) 0 21 75 68 16 2 17

پاسخ: جواب گزینه ۴ است.

نحوه پاسخ به درخواست ها برابر است با:

$$0 \to 21 \to 75 \to 68 \to 16 \to 2 \to 17$$

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 سیلندر75و16 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 سیلندرهای 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 الگوریتم ها 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.

دسته‌بندی موضوعی آموزش‌های فرادرس، در ادامه آمده است:

 برنامه نویسی – کلیک (+)	 آموزش های دانشگاهی و تخصصی – کلیک (+)	 هوش مصنوعی و یادگیری ماشین – کلیک (+)	 مهندسی برق الکترونیک و روباتیک – کلیک (+)
 دروس دانشگاهی – کلیک (+)	 مباحث مشترک – کلیک (+)	 مهارت های دانشگاهی – کلیک (+)	 نرم افزارهای تخصصی – کلیک (+)
 مهندسی نرم افزار – کلیک (+)	 نرم افزارهای عمومی – کلیک (+)	 طراحی و توسعه وب – کلیک (+)	 آموزش های عمومی – کلیک (+)

منبع مطالعاتی تکمیلی مرتبط با این کتاب

آموزش سیستم‌های عامل

سیستم عامل یا سامانه عامل (Operating System) بدون شک مهمترین نرم‌افزار در کامپیوتر است. سیستم عامل اولین نرم‌افزاری است که پس از روشن کردن کامپیوتر مشاهده می‌شود و همچنین آخرین نرم‌افزاری خواهد بود که قبل از خاموش کردن کامپیوتر مشاهده می‌شود. سیستم عامل نرم‌افزاری است که مدیریت برنامه‌ها را به عهده گرفته و با کنترل، مدیریت و سازماندهی منابع سخت‌افزاری امکان استفاده بهینه و هدفمند آنها را فراهم کرده و بستری را برای اجرای نرم‌افزارهای کاربردی فراهم می‌کند.

آموزش سیستم عامل، توسط مهندس فرشید شیرافکن، یکی از بهترین مدرسین مسلط به این مباحث، ارائه شده است.

مدرس: مهندس فرشید شیر افکن

مدت زمان: ۱۱ ساعت

جهت مشاهده آموزش ویدئویی این آموزش – کلیک کنید **faradars.org /fvsft103**