# فصل اول: مفاهیم اولیه

**سیستم عامل چیست؟**

سیستم عامل نرم افزاری است که سه وظیفه زیر را برعهده دارد:

**1- مدیریت منابع:** سیستم‌عامل نرم افزاری است که مدیریت، کنترل و نظارت بر همه برنامه ها و منابع سخت افزاری و نرم افزاری را برعهده دارد.

**2- رابط بین کاربر و برنامه‌های کاربردی با سخت افزار:** سیستم عامل یک لایه نرم افزاری است که بر روی سخت افزار کامپیوتر قرار گرفته و تمامی ارتباطات میان کاربر و برنامه‌ها با سخت افزار از طریق سیستم عامل صورت می‌گیرد.

**3- ماشین مجازی:** سیستم عامل یک لایه نرم افزاری است که از لایه پایینی خود (سخت افزار) سرویس‌هایی را در یافت کرده و پس از بهینه‌سازی و فرآوری این سرویس‌ها، به لایه‌های بالایی خود یعنی برنامه‌ها و کاربران سرویس ارائه می‌دهد.

**مزایای بکارگیری سیستم‌عامل:**

بکارگیری سیتم‌عامل باعث مزایای زیر می‌گردد:

**1- سهولت کار کاربر:** با وجود سیستم‌عامل کاربران بجای کار با محیط سخت افزاری با یک محیط نرم‌افزاری و خوشایند برای کاربر کار می‌کنند.

**2- سهولت تولید نرم‌افزارها:** سیستم‌های عامل بسترها و امکاناتی فراهم می‌آورند که در نتیجه آن تولید نرم-افزارها ساده تر شده و نرم‌افزارهای قدرتمند تر تولید می‌شود.

**3- سهولت تولید سخت‌افزارهای جدید:** چون سیستم‌عامل واسط میان برنامه‌های کاربردی و سیستم‌عامل است، هنگام تولید یک سخت افزار جدید کافی است فقط سیستم‌عامل روش کار با سخت افزار جدید را بیاموزد و سایر برنامه ها بدون مشکل از طریق سیستم‌عامل با آن سخت افزار جدید ارتباط برقرار خواهند کرد.

**4- افزایش کارایی:** چون سیستم‌عامل استفاده از منابع سیستم را مدیریت می‌کند، باعث استفاده حداکثری از منابع سخت‌افزاری و نرم‌افزاری سیستم می‌شود.

**سلسله مراتب اجزاء در سیستم کامپیوتری:**

| |
|---|
| **کاربر** |
| **برنامه‌های کاربردی** |
| **برنامه‌های سیستم (مترجم‌ها، ویرایشگرها، پوسته فرمان)** |
| **سیستم‌عامل** |
| **زبان ماشین** |
| **ریز برنامه‌ها** |
| **دستگاههای فیزیکی** |

**دستگاه‌های فیزیکی:** این لایه شامل تمامی اجزاء سخت‌افزاری کامپیوتر مانند تراشه‌ها، مدارها و کلیه دستگاههای فیزیکی کامپیوتر است.

**ریزبرنامه‌ها:** ریزبرنامه‌ها مجموعه دستورالعمل‌هایی هستند که در نتیجه اجرای آنها سخت‌افزار کامپیوتر عملیات مورد نظر را در مدارهای مجتمع انجام میدهد. مثلا مقدار دو ثبات را باهم جمع می‌کند.

**زبان ماشین:** زبان ماشین مجموعه دستوراتی است که بکمک آن، برنامه نویسان باعث انجام عملیات مورد نظر خود در کامپیوتر می‌شوند. شایان ذکر است که در دستور زبان ماشین به تعدادی ریزدستور (یک ریز برنامه) در لایه پایینی تبدیل شده و اجرا می‌شود.

**سیستم‌عامل:** سیستم‌عامل همان لایه نرم‌افزاری است که میان سخت افزار و برنامه‌ها و کاربران قرار گرفته و باعث مخفی شدن پیچیدگی‌های سخت‌افزاری از دیدگاه آنان می‌شود.

**برنامه‌های سیستمی:** برنامه‌هایی هستند که معملا جنبه مولد دارند مانند کامپایلرها، ادیتورها و ...

**برنامه‌های کاربردی:** برنامه‌هایی خاص منظوره هستند که هریک برای انجام فعالیت خاصی تولید شده اند. مانند: یک نرم‌افزار حسابداری، گرافیکی و ...

**کاربر:** استفاده کننده نهایی از سیستم کامپیوتری است.

**تاریخچه سیستم‌عامل:**

**1- سیستم‌های ردیفی:** اولین کامپیوترهایی که در آنها برنامه ساز مستقیما با سخت افزار در ارتباط بود، یعنی در حقیقت سیستم‌عاملی در کارنبود. این ماشینها از طریق یک میز فرمان که شامل چراغ‌های نمایش، کلیدها و نوعی دستگاه‌های ورودی/خروجی مانند کارت خوان و چاپگر بودند اجرا می‌شدند. برنامه‌هایی که به زبان ماشین بودند و روی کارت پانچ شده بودند توسط دستگاه ورودی به درون حافظه اصلی بار شده وتوسط پردازنده اجرا می‌شدند. اگر برنامه خطا داشت، این خطاها توسط چراغها نشان داده می‌شد و برنامه ساز باید در این حالت ثباتها و حافظه اصلی را برای تعیین علت خطا بررسی می‌کرد و با برطرف کردن خطاها و تکمیل تدریجی برنامه، برنامه نهایی اجرا شده و خروجی برنامه روی چاپگر ظاهر می‌شد.

معایب این سیستم‌ها عبارتند از: هر کاربر برای استفاده از سیستم باید از قبل نوبت می‌گرفت، زمان زیادی برای پانچ برنامه بر روی کارت و بار کردن برنامه در سیستم صرف می‌شد، این ماشینها فاقد سیستم‌عامل بودند، سرعت پایین داشته و قابلیت محاوره با کاربر را نداشتند.

**2- سیستم‌های دسته ای ساده (Batch systems)**

سخت افزار سیستم‌های دسته‌ای متشکل از یک کارتخوان، حافظه اصلی، پردازنده و چاپگر بود. در سیستم‌های دسته‌ای برنامه‌ها روی کارت‌های پانچ شده ذخیره می‌شد (قرار می‌گرفت). سپس توسط اپراتور سیستم برنامه-هایی که دارای نیاز یکسان بودند مثلا به یک مترجم (زبان برنامه سازی) نیاز داشتند در یک گروه قرار گرفته و در نهایت این گروه‌ها در کارتخوان قرار می‌گرفتند. برای اجرای برنامه‌ها، پس از بار شدن سیستم‌عامل و مترجم مورد نیاز در حافظه اصلی، دنباله عملیات زیر تکرار و توسط سیستم‌عامل کنترل می‌شد:

- کارتخوان روشن شود.
- یک برنامه خوانده شده و به حافظه اصلی منتقل شود.
- کارتخوان خاموش شود.
- برنامه خوانده شده اجرا شود.
- خروجی برنامه چاپ شود.
- برگرد به مورد اول

در این سیستم‌ها مجموعه دستوراتی بنام زبان کنترل کار (Job Contril Language یا JCL) وجود داشت که توسط آن برنامه نویس سیستم‌عامل را در خصوص کارت شروع برنامه، پایان برنامه، کامپایلر مورد نیاز و ... آگاه می‌ساخت.

ویژگیهای این سیستم‌ها:

- ابزار محاوره در این سیستم‌ها وجود نداشت.
- امکان اجرای یک برنامه همزمان با انجام ورودی/خروجی همان برنامه یا برنامه دیگر وجود نداشت.
- بدلیل مورد قبل، همواره پردازنده یا کارتخوان یا چاپگر بیکار بودند و کارایی سیستم پایین بود.
- مهمترین وظیفه سیستم‌عامل در سیستم‌های دسته‌ای فقط انتقال اتوماتیک از یک کار به کار دیگر بود.

**تکنیک Spooling:** برای حل مشکل بیکاری و فراهم کردن امکان اجرای یک برنامه همزمان با انجام ورودی/خروجی برنامه دیگر، تکنیک مذکور ابداع گردید. این تکنیک بر دونوع است:

- Offline Spooling: در این تکنیک از نوار مغناطیسی استفاده می‌شد بدین صورت که پس از گروه بندی کارها بر اساس نیاز یکسان، ابتدا تعدادی کارها از دستگاه کارتخوان خوانده شده و روی یک نوار مغناطیسی نوشته می‌شد. سپس این نوار به دستگاه دیگری که متشکل از یک نوارخوان، پردازنده، حافظه اصلی و نوار نویس بود منتقل می‌شد. در این مرحله پردازنده برنامه‌ها را یکی یکی از نوار خوانده و اجرا می‌کرد. خروجی هر برنامه هم توسط نوار نویس روی یک نوار دیگر نوشته می‌شد. توجه شود که در این حین، کارتخوان متوقف نشده و برنامه‌های بعدی را روی نوار دیگری می‌نوشت. در نهایت هرگاه نوار خروجی پردازنده پر می‌شد، این نوار به دستگاه دیگری منتقل شده تا عمل چاپ خروجی‌ها صورت گیرد. شایان ذکر است که در حین چاپ خروجی‌ها، عملیات خواندن برنامه‌ها از کارتخوان و اجرای برنامه‌های دیگر توسط پردازنده ادامه داشت. بنابراین عملیات ورود/ اجرا/ خروج همزمان انجان شده و ضمن کاهش بیکاری، کارایی سیستم افزایش می‌یافت.
- Online Spooling: در این تکنیک از دیسک مغناطیسی استفاده می‌شد بدین صورت که پس از گروه بندی کارها بر اساس نیاز یکسان، ابتدا کارها از دستگاه کارتخوان خوانده شده و روی دیسک مغناطیسی نوشته می‌شد. سپس پردازنده برنامه‌ها را یکی یکی از دیسک مغناطیسی خوانده و اجرا می-کرد. خروجی هر برنامه هم توسط پردازنده روی همان دیسک مغناطیسی نوشته می‌شد. توجه شود که در این حین، کارتخوان متوقف نشده و برنامه‌های بعدی را روی دیسک مغناطیسی می‌نوشت. در نهایت هرگاه خروجی برنامه ای آماده می‌شد عمل چاپ آن خروجی صورت می‌گرفت. شایان ذکر است که در حین چاپ خروجی‌ها، عملیات خواندن برنامه‌ها از کارتخوان و اجرای برنامه‌های دیگر توسط پردازنده

ادامه داشت. بنابراین عملیات ورود/ اجرا/ خروج همزمان انجان شده و ضمن کاهش بیکاری، کارایی سیستم افزایش می‌یافت.

سیستم Spooling اجرای یک برنامه را با عملیات ورودی/خروجی برنامه دیگر بطور همزمان انجام داده و از این طریق ضمن کاهش بیکاری، کارایی سیستم را افزایش می‌دهد.

### 3- سیستم‌های عامل چندبرنامه‌ای دسته ای (Multiprograming Batch systems)

همزمان با پیشرفت سخت‌افزار و نرم‌افزارکامپیوتر، لازم بود تا برنامه‌ها در لابه‌لای دستورات اجرایی عملیات ورودی/خروجی انجام دهند. مثلا از دیسک یا هر ابزار ورودی دیگری مقداری را بخوانند یا خروجی میانی را روی چاپگر یا هر وسیله خروجی دیگری نمایش دهند. از طرفی همانگونه که میدانیم سرعت دستگاههای ورودی/خروجی بسیار کمتر از سرعت پردازنده است، بنابراین هنگامی که یک برنامه در لابه‌لای دستورات اجرایی عملیات ورودی/خروجی انجام می‌داد پردازنده تا اتمام عملیات ورودی/خروجی بیکار شده و کارایی سیستم پایین می‌آید. برای حل این مشکل سیستم‌های دسته‌ای به سیستم‌های چندبرنامه‌ای دسته‌ای تبدیل شدند. در این سیستم‌ها از یک حافظه اصلی بزرگتر استفاده می‌شد و بجای یک برنامه، چندین برنامه از انبارکار (دیسک مغناطیسی) به حافظه اصلی آورده می‌شد. حال هرگاه که یک برنامه در لابه‌لای دستورات اجرایی عملیات ورودی/خروجی انجام می‌داد پردازنده موقتا اجرای آن برنامه را رها کرده و برنامه دیگری را از حافظه اصلی اجرا می‌کرد. چنانچه آن برنامه هم نیاز به عملیات ورودی/خروجی داشته باشد، این برنامه نیز رها شده و برنامه دیگری اجرا می‌شد. بدیهی است که در زمانی بعدتر اجرای برنامه‌هایی که موقتا متوقف شده بود ادامه می‌یافت. بنابراین اگر به تعداد کافی برنامه در حافظه اصلی موجود باشد می‌توان پردازنده را در تمام اوقات مشغول نگه-داشت. ویژگی‌های این سیستم‌ها عبارتند از:

- کاهش بیکاری پردازنده و افزایش کارایی.
- امکان همزمانی اجرای یک برنامه با عملیات ورودی/خروجی برنامه دیگر.
- اولین سیستم‌عاملی است که درآن بجای کاربر سیستم‌عامل در مورد ترتیب اجرای برنامه‌ها تصمیم می-گیرد.
- در این سیستم‌ها برای ترتیب‌دهی به جرای برنامه‌ها "زمان‌بندی" انجام شده و برای قرار دادن چند برنامه در حافظه نیاز به "مدیریت حافظه" است.
- این سیستم‌ها از سیستم‌های تک‌برنامگی پیچیده‌تر هستند.
- محاوره بین کاربر و کامپیوتر وجود نداشت.

- برای انجام عملیات ورودی/خروجی بدون درگیرکردن پردازنده از تکنیک DMA استفاده می‌شود. در تکنیک DMA، قطعه‌ای بنام کنترل کننده DMA در سخت‌افزار کامپیوتر تعبیه شده که میتواند داده‌ها را بصورت مستقیم و بدون دخالت پردازنده میان حافظه اصلی و دستگاههای ورودی/خروجی انتقال دهد. در این حالت، پردازنده فرمان انجام ورودی خروجی را به کنترل کننده DMA صادر کرده و خود به اجرای برنامه دیگری می‌پردازد. از این پس عملیات ورودی/خروجی توسط کنترل کننده DMA انجام می‌شود.

**4- سیستم‌های عامل اشتراک زمانی یا محاوره‌ای (Timesharing)**

در سیستم‌های اشتراک‌زمانی یک کامپیوتر مرکزی بنام mainframe متشکل از پردازنده، حافظه اصلی و دیسک سخت وجود دارد که قرار است به تعدادی کاربر سرویس دهد. هر کاربر برای کار با این کامپیوتر یک ترمینال در اختیار دارد که شامل یک صفحه کلید و یک صفحه نمایش (و شاید هم یک موشواره) است. پس از اینکه کاربر به سیستم login می‌کند برنامه مورد نظر خود برای اجرا را درخواست میکند. با در خواست کاربر برنامه وی در حافظه اصلی قرار می‌گیرد. در چنین شرایطی علاوه بر سیستم‌عامل چندین برنامه در حافظه اصلی قرار دارد که هریک متعلق به یکی از کاربران است. حال سیستم‌عامل زمان پردازنده را به بازه‌های مساوی بنام برش‌زمانی یا کوانتوم تقسیم کرده و در بازه‌زمانی بخشی از برنامه یکی از کاربران اجرا می‌شود. با پایان یافتن برش‌زمانی پردازنده برنامه در حال اجرا را موقتا رها کرده و برنامه کاربر دیگری را اجرا می‌کند. بعبارت دیگر پردازنده دائما میان برنامه‌های کاربران مختلف سوئیچ می‌کند.عمل تخصیص برش‌های زمانی آنقدر سریع صورت می‌گیرد که کاربر متوجه توقف/ادامه اجرای برنامه خود نمی‌شود و امکان محاوره کاربر با سیستم بوجود می‌آید. محاوره یعنی اینکه پس صدور دستوری از سوی کاربر، بلافاصله و با تاخیری ناچیزی، دستور کاربر توسط سیستم اجرا شده و پاسخ (خروجی) به کاربر برگردانده شود. ویژگیهای این سیستم‌ها عبارتند از:

- این سیستم‌ها تعمیم یافته سیستم‌های چندبرنامگی هستند و هدف از تولید آنها پاسخگویی همزمان به چندین کاربر است.
- ارتباط محاوره‌ای با کاربر وجو دارد.
- یک برنامه تا پایان برش زمانی خود اجرا می‌شود در حالیکه در سیستم‌های چند برنامه ای یک برنامه تا زمانی اجرا می‌شود که نیاز به عملیات ورودی/خروجی داشته باشد.
- بیکاری منابع از طریق اشتراک منابع کاهش یافته و کارایی افزایش می‌یابد.

- سوئیچ کردن بین برنامه‌ها باید تا حد ممکن سریع بوده و بنابراین باید برش‌زمانی تا حدممکن کوتاه باشد.
- زمان پاسخ کاربر باید تا حد ممکن کوتاه باشد.
- این سیستم‌ها پیچیدتر از سیستم‌های چندبرنامه‌ای بوده و علاوه بر مدیریت حافظه و زمان‌بندی نیازمند سیستم مدیریت فایل (فایل سیستم) نیز هستند.

### 5- کامپیوترهای شخصی (Personal Computer)

به لطف توسعه سخت‌افزار، در کامپیوترهای شخصی، ابزار ورودی به صفحه کلید و موشواره و ابزار خروجی به صفحه نمایش و چاپگرهای کوچک تبدیل شده است. همچنین پردازنده مرکزی این کامپیوترها بنام ریزپردازنده شناخته می‌شود. اولین سیستم‌عامل نوشته شده برای این کامپیوترها، سیستم‌عامل DOS بود. سیستم DOS یک سیستم‌عامل تک کاربره و تک برنامه‌ای بود. پس از آن سیستم‌عامل OS/2 به بازار آمد که چندان مورد توجه قرار نگرفت. امروزه در کامپیوترهای شخصی عموما از سیستم عامل‌های windows و Linux که سیستم‌عامل‌های چندکاربره و چندبرنامگی با قابلیت‌های شبکه ای هستند استفاده می‌شود. اصولا کامپیوترهای شخصی دارای سخت‌افزار و نرم‌افزار ارزان قیمت بوده و سیستم‌های خوشایند برای کاربر هستند.

### 5- سیستم‌های موازی (Parallel Processing)

این سیستم به لحاظ سخت‌افزاری متشکل از چندین پردازنده است که بصورت مشترک از یک ماژول حافظه، ذخیره ساز جانبی و پالس ساعت یکسان استفاده می‌کنند. در چنین سیستمی علاوه بر سیستم‌عامل چندین برنامه در حافظه اصلی قرار گرفته و سپس هر برنامه توسط یک پردازنده اجرا می‌شود. بنابراین واقعا در هر لحظه چندین برنامه در حال اجرا است. ویژگی‌های این سیستم‌ها عبارتند از:

- سرعت بسیار زیاد.
- توان عملیاتی بالا: تعداد برنامه‌های کامل اجرا شده در بازه زمانی بسیار زیاد است.
- کارایی بالا: چون منابع سیستم بصورت اشتراکی میان چندین پردازنده استفاده می‌شود، بیکاری منابع در سیستم کاهش یافته و کارایی افزایش می‌یابد.
- قابلیت اعتماد یا اطمینان پذیری یا تحمل پذیری در مقابل خرابی: چون در این سیستم چند پردازنده وجود دارد، با ازکار افتادن پردازنده‌ای توان سیستم کاهش می‌یابد ولی سیستم کاملا از کار نمی افتد.

این سیستم‌ها از نظر نرم‌افزاری بر دو نوع هستند:

- متقارن: در این سیستم هر پردازنده هم اجرای سیستم‌عامل هم اجرای برنامه‌ها را برعهده دارد. این سیستم ها از نظر توازن بار عملکرد بهتری دارند چرا که اجرای سیستم‌عامل بعنوان یک نرم‌افزار سنگین بر عهده همه پردازنده‌ها است. همچنین تحمل پذیری خطا در این سیستم‌ها بالاتر است چون سیستم-عامل توسط همه پردازنده‌ها اجرا شده و با از کار افتادن پردازنده خاصی سیستم از کار نمی‌افتد.
- نامتقارن: در این سیستم یک پردازنده اجرای سیستم عامل و سایر پردازنده‌ها اجرای برنامه‌ها را برعهده دارند. این سیستم ها از نظر توازن بار عملکرد مطلوبی ندارند چرا که اجرای سیستم‌عامل بعنوان یک نرم‌افزار سنگین بر عهده یکی از پردازنده‌ها است. همچنین تحمل پذیری خطا در این سیستم‌ها پایین‌تر است چون سیستم‌عامل فقط توسط یکی از پردازنده‌ها اجرا شده و با از کار افتادن آن پردازنده خاص سیستم از کار می‌افتد. لیکن تولید سیستم‌های عامل نامتقارن آسان‌تر است.

### 6- سیستم‌های توزیع شده (Distributed Systems)

این سیستم‌ها بلحاظ سخت‌افزاری از یک مجموعه کامپیوتر مستقل از هم تشکیل شده که بوسیله یک شبکه با سرعت بالا بهم متصل شده‌اند. بواسطه وجود سیستم‌عامل توزیع شده، این کامپیوترهای مستقل بصورت یک سیستم واحد عمل می‌کنند یعنی بگونه‌ای که در ظاهر تمامی منابع در اختیار همه کاربران است و کاربران میتوانند بدون اینکه خودشان متوجه شوند از منابع کامپیوترهای یکدیگر استفاده نمایند. مثلا برنامه کاربر A به کامپیوتر کاربر B منتقل شده و پس از اجرا روی آن کامپیوتر، نتیجه به کامپیوتر کاربر A منتقل گردد. یا اینکه فایل‌های یک کاربر روی کامپیوتر کاربر دیگری ذخیره گردد. نکته بسیار مهم در این سیستم وجود "شفافیت" ( Trancparency) است یعنی کاربران نباید از محل اجرای برنامه، ذخیره سازی داده و بطور کلی محل فیزیکی منابع آگاه گردند و باید همه این امور توسط سیستم‌عامل توزیع شده و بصورت خودکار و با کارایی بالا صورت گیرد.

### 7- سیستم‌عامل‌های بلادرنگ (Realtime)

سیستم‌هایی هستند که در آنها هر برنامه برای اجرا شدن یک مهلت زمانی بنام ضرب الاجل (Deadline) دارد. سیستم عامل باید هر برنامه را تا قبل از منقضی شدن ضرب‌الاجل مربوطه‌اش اجرا نماید چرا که اجرای برنامه بعد از ضرب الاجل کم ارزش یا بی‌ارزش خواهد شد. این سیستم‌ها بر دو گونه هستند:

- سیستم بلادرنگ سخت: در این سیستم اجرای برنامه بعد از ضرب الاجل بی‌ارزش خواهد بود و باید سیستم عامل هر برنامه را تا قبل از منقضی شدن ضرب‌الاجل مربوطه‌اش اجرا نماید. همانند سیستم-های کنترل صنعتی، موشک‌ها، ماهواره‌ها و ... بنابراین سیستم عامل باید رعایت ضرب‌الاجل ها را گارانتی نماید.
- سیستم بلادرنگ نرم: در این سیستم اجرای برنامه بعد از ضرب الاجل کم‌ارزش خواهد بود و سیستم عامل سعی میکند هر برنامه را تا قبل از منقضی شدن ضرب‌الاجل مربوطه‌اش اجرا نماید. همانند سیستم‌های چندرسانه‌ای، زمان‌بندی پروژه و ... بنابراین سیستم عامل بهترین تلاش خود را انجام می-دهد.

# فصل دوم: مدیریت فرآیند(Process Management)

**1)فرآیند،پردازه Process**: به هر برنامه در حال اجرا فرآیند گویند. در حالت کلی به هر برنامه ای که از حافظه جانبی به حافظه اصلی آمده باشد و اجرایش آغاز شده باشد فرآیند گفته میشود، اگرچه ممکن است در حال حاضر پردازنده در حال اجرای دستوراتش نباشد. بنابراین تعابیر زیر در خصوص یک فرآیند صحیح است:

- برنامهء در حال اجرا را فرآیند گویند.
- همهء برنامه های درون حافظه ی اصلی.
- یک سری ساختمان داده ها و دستورالعمل ها.
- یک فعالیت در سیستم کامپیوتری.

**2)تفاوت برنامه و فرآیند:** در حالت کلی دو تفاوت عمده میان یک برنامه و فرآیند است که عبارتند از تفاوت در محل قرارگیری و تفاوت در ماهیت.

- از نظر محل قرار گیری: برنامه در حافظهء جانبی هستند و فرآیندها عموما در حافظه اصلی هستند.
- از نظر ماهیت : برنامه ماهیتی غیر فعال دارد درحالیکه فرآیند ماهیتی فعال دارد.

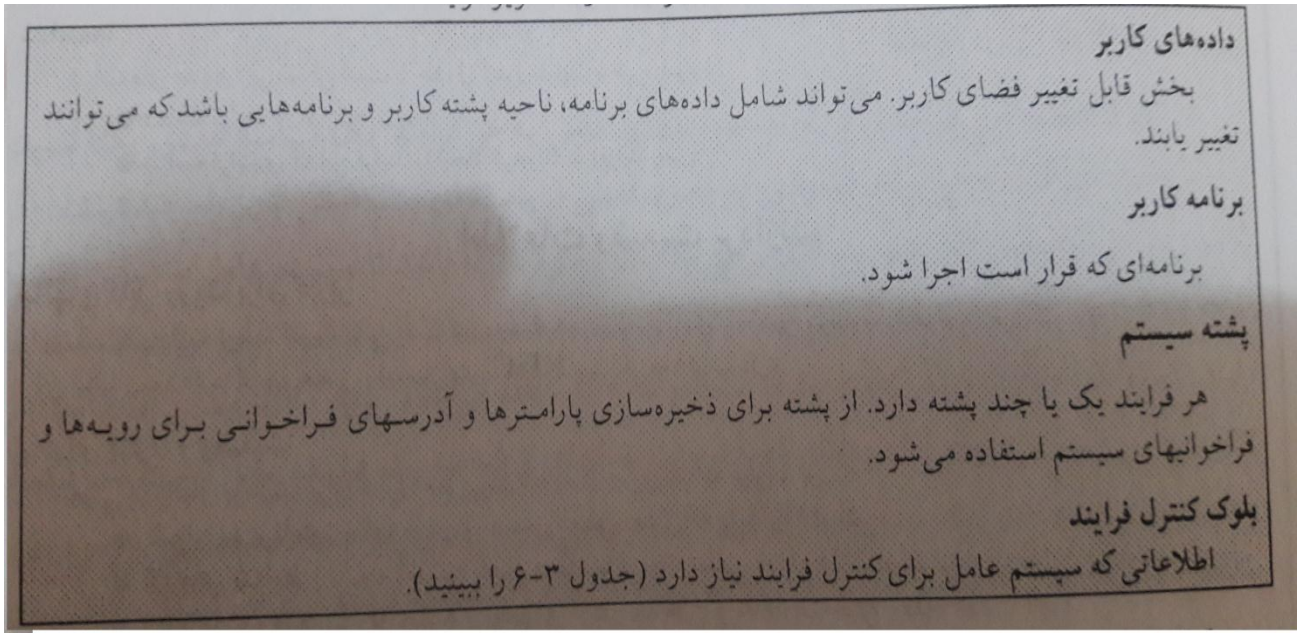
**3)دلایل ایجاد فرآیند:**

- ورود کار دسته ای جدید: سیستم عامل با جریانی از کارهای دسته ای (معمولا روی دیسک) مواجه است. وقتی برای گرفتن یک کار جدید آماده است، کار بعدی را تبدیل به فرآیند می‌کند.
- برقراری ارتباط محاوره ای: درخواست کاربر برای اجرای برنامه ها، همانند سیستم‌های اشتراک زمانی، با عث ایجاد فرآیند می شود.
- تولید یک فرآیند توسط فرآیند دیگر
- ارایهء سرویس ها توسط سیستم عامل: سیستم عامل می‌تواند فرآیندی را برای ارائه خدمتی از طرف برنامه کاربر ایجاد نماید، بدون اینکه کاربر ناچار به انتظار کشیدن باشد. همانند کپی کردن یک فایل، چاپ کردن یک فایل یا ... .

**4)دلایل خاتمه فرآیند:** دلایل خاتمه فرآیند در جدول ذیل آورده شده است.

| | |
|---|---|
| **پایان طبیعی** | این فرایند، یک خدمت سیستم عامل را برای بیان تکمیل اجرایش فراخوانی می‌کند. |
| **سقف زمانی** | این فرایند برای سقف زمانی منظور شده، اجرا شده است. انواع مختلفی از سقف زمانی می‌تواند مطرح باشد. از جمله کل زمان سپری شده (زمانی که ساعت دیواری نشان می‌دهد)؛ زمانی که صرف اجرا شده است و در مورد فرایندهای محاوره‌ای، مقدار زمانی که از آخرین ورودی کاربر گذشته است. |
| **نبود حافظه** | این فرایند به حافظه‌ای بیش از آنچه که سیستم می‌تواند فراهم کند، نیاز دارد. |
| **تجاوز از حدود** | این فرایند می‌خواهد به محلهایی از حافظه مراجعه کند، که مجاز نیست. |
| **خطای حفاظت** | این فرایند برای دسترسی به پرونده یا منبعی تلاش می‌کند، که مجاز به استفاده نیست و یا به نحو مناسبی استفاده نمی‌کند، مثل نوشتن در پرونده فقط خواندنی. |
| **خطای محاسباتی** | این فرایند، یک محاسبه غیرمجاز، مثل تقسیم بر صفر را انجام می‌دهد؛ یا می‌خواهد عددی بزرگتر از ظرفیت سخت افزاری را ذخیره نماید. |
| **گذشت زمان** | این فرایند بیش از حداکثر زمانی که تعیین شده است، برای بروز حادثه مشخصی منتظر مانده است. |
| **خطای ورودی/خروجی** | خطایی در ورودی/خروجی اتفاق افتاده است. مثل پیدا نکردن یک پرونده، عدم توفیق در خواندن یا نوشتن در حداکثر تعداد دفعاتی که برای تلاش مجدد مشخص شده است (مثل موقع مواجه شدن با ناحیه صدمه دیده‌ای از نوار) یا عمل غیر معتبر (مثل خواندن از روی چاپگر). |
| **دستورالعمل نامعتبر** | این فرایند سعی می‌کند دستورالعملی را اجرا کند که وجود ندارد (اغلب به خاطر انشعاب به یک ناحیه داده‌ای و تلاش برای اجرای یک داده، این اتفاق می‌افتد). |
| **دستورالعمل ممتاز** | این فرایند برای اجرای دستورالعملی تلاش می‌کند، که مخصوص سیستم عامل است. |
| **استفاده نامناسب از داده** | داده‌ای از نوع نامناست و یا بدون مقدار اولیه است. |
| **دخالت سیستم عامل یا متصدی** | به دلایلی (مثل وجود بن بست) سیستم عامل، یا متصدی، این فرایند را پایان داده‌اند. |
| **پایان یافتن پدر** | ممکن است سیستم عامل طوری طراحی شده باشد که با پایان یافتن یک فرایند، تمام فرایندهایی که فرزند آن هستند، پایان داده شوند. |
| **درخواست پدر** | معمولاً یک فرایند حق پایان دادن به هر یک از فرایندهای فرزند خود را در اختیار دارد. |

**5)تصویر فرآیند:** اجزاء تصویر فرآیند در جدول زیر آورده شده است. محل تصویر فرآیند عموما در حافظه اصلی است اما ممکن است بدلایلی موقتا به حافظه جانبی منتقل گردد.



**داده‌های کاربر**

بخش قابل تغییر فضای کاربر. می‌تواند شامل داده‌های برنامه، ناحیه پشته کاربر و برنامه‌هایی باشد که می‌توانند تغییر یابند.

**برنامه کاربر**

برنامه‌ای که قرار است اجرا شود.

**پشته سیستم**

هر فرایند یک یا چند پشته دارد. از پشته برای ذخیره‌سازی پارامترها و آدرسهای فراخوانی برای رویه‌ها و فراخوانیهای سیستم استفاده می‌شود.

**بلوک کنترل فرایند**

اطلاعاتی که سیستم عامل برای کنترل فرایند نیاز دارد (جدول ۳-۶ را ببینید).

**6)انواع فرآیند:** جهت تعریف انواع فرآیند به دو تعریف ذیل احتیاج است.

از آنجا که یک فرآیند در حین اجرا متناوبا عملیات پردازشی و ورودی خروجی انجام میدهد، بنابراین بخشی از زمان اجرای خود را صرف انجام عملیت پردازشی و بخشی دیگر را صرف انجام عملیات ورودی خروجی مینماید. بنابراین:

- زمان انفجار محاسباتی یا CPU Burst: مقدار زمانی که یک فرآیند برای انجام یک بار عملیات پردازشی درپردازنده صرف می نماید را یک CPU Burst گویند.
- زمان انفجار ورودی/خروجی یا I/O Burst: مقدار زمانی که یک فرآیند برای انجام یک بار عملیات ورودی/خروجی صرف می نماید را یک I/O Burst گویند.

بر اساس تعاریف گفته شده دونوع فرآیند تعریف می شود.

- فرآیند با تنگنای پردازشی(CPU Limited/CPU Bounded): به فرآیندی گفته می شود که عمده زمان اجرای خود را صرف انجام عملیات پردازشی درپردازنده می‌نماید.

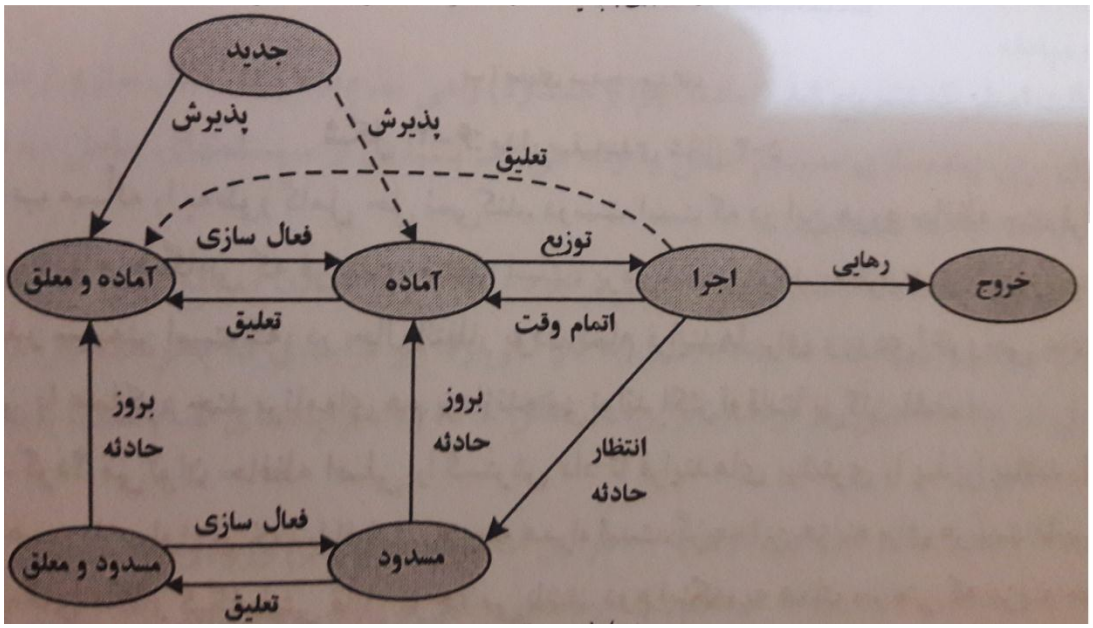
- فرآیند با تنگنای ورودی/خروجی(I/O Limited / I/O Bounded): به فرآیندی گفته می شود که عمده زمان اجرای خود را صرف انجام عملیات ورودی/خروجی با ابزارهای I/O می‌نماید.

**7)مراحل اجرای فرآیند**

1) ایجاد فرآیند: برای اینکه یک فرآیند ایجاد گردد باید مراحل زیر به ترتیب انجام گیرد:

- مرحله اول- پذیرش کار جدید: در این مرحله یکی از چهار دلیل ذکر شده برای ایجاد فرآیند باعث می‌شود تا یک کارجدید توسط سیستم عامل پذیرش گشته و تبدیل به فرآیند گردد.
- مرحله دوم- ایجاد و تخصیص بلوک کنترلی فرآیند: در این مرحله یک ساختمان داده بنام PCB (متعاقبا شرح داده خواهد شد) برای فرآیند ساخته می‌شود. در این ساختمان داده اطلاعات ضروری فرآیند از دید سیستم عامل ذخیره می‌گردد. هر فرآیند دارای ساختمان داده منحصر به خود بوده و در محیط سیستم عامل فرآیند توسط PCB خود شناخته می‌شود. PCB تمام فرآیندها در یک ساختار از نوع لیست پیوندی در حافظه اصلی نگهداری می‌شود.
- مرحله سوم- قرار گرفتن در صف آماده: پس از اینکه برای کار پذیرش شده PCB ساخته شود، این کار تبدیل به فرآیند می‌شود. حال باید این فرآیند (آدرس PCB این فرآیند) در صف فرآیندهای آماده قرار گیرد تا طبق الگوریتم زمان بندی پردازنده اجرا گردد. از این پس طبق چرخه حالات فرآیند، فرآیند به سمت خاتمه ادامه مسیر میدهد.

**8 )چرخه حالات فرآیند:** همانطور که پیش‌تر گفته شد، برای اینکه یک برنامه بتواند پس از تبدیل شدن به فرآیند اجرا شده و به سوی خاتمه خود پیش برود باید در یک چرخه حالات گردش کند. بدیهی است که هر حالت این چرخه متناظر با وضعیتی از اجرای فرآیند است.




- **حالت پذیرش یا New یا Hold** : در این حالت اطلاعات شامل اسم و آدرس کارهایی قرار دارد که بتازگی توسط سیستم عامل پذیرش شده اند ولی هنوز تبدیل به فرآیند نشده اند. بعبارت دیگر کارهای موجود در این حالت درحال تبدیل شدن به فرآیند بوده و هنوز در حافظه جانبی هستند. همانگونه که در فصل بعدی توضیح داده خواهد شد، زمان بند بلند مدت درخصوص انتخاب کار جهت تبدیل شدن به فرآیند تصمیم گیری می‌کند. شایان ذکر است که زمان‌بند بلند مدت عموما در سیستم‌های دسته ای وجود داشته است.
- **حالت آماده یا Ready**: پس از اینکه یک کار به فرآیند تبدیل شد اطلاعات شامل نام و آدرس آن فرآیند در این حالت قرار می‌گیرد. بعبارت دیگر نام فرآیندهایی در این آرایه است که در حافظه اصلی بوده و تمامی منابع لازم جهت اجرا به غیر از پردازنده را در اختیار داشته و آماده(منتظر) در اختیار گرفتن پردازه هستند تا دستوراتشان اجرا گردد. در حالت کلی سه رخداد باعث می شود تا نام فرآیند در حالت آماده قرار گیرد: 1- پس از اینکه کار پذیرش شده و تبدیل به فرآیند شد. 2- هنگامی که فرآیند دیگری توسط سیستم عامل (زمان بند) از حالت آماده برای اجرا اننتخاب شود و باید جایگزین فرآیند درحال اجرا گردد. در این حالت فرآیند در حال اجرا قطع شده و به حالت آماده برمی-گردد. 3- هنگامی که فرآیندی به دلیلی مسدود شده باشد و دلیل مسدود شدن برطرف شود؛ مثلا جهت انجام عمل ورودی/خروجی مسدود شده باشد و حال عمل ورودی خروجی خود را انجام داده باشد.

- **حالت اجرا یا Run :** در این حالت فرآیند تمامی منابع مورد نیاز جهت اجرا، حتی پردازنده، را در اختیار دارد و در حال اجراست. بعبارت دیگر تنها اسم و آدرس یک فرآیند در این حالت قرار دارد و آن فرآیندی است که در این لحظه دستوراتش توسط پردازنده اجرا می شود و خود فرآیند نیز در حافظه اصلی است.
- **حالت مسدود یا Block:** هنگامی که فرآیندی منتظر رخداد حادثه‌ای باشد نامش در این حالت قرار می‌گیرد. در حالت کلی نام فرآیندهایی در این حالت قرار می‌گیرد که در حال عمل I/O باشند، منتظر شروع عمل I/O باشند و یا موقّتا بدلیلی غیرقابل اجرا (حتی اگر پردازنده در اختیار آنها قرار گیرد) هستند. شایان ذکر است که این فرآیندها نیز در حافظه اصلی هستند.
- **حالت مسدود معلق یا Suspend wait/block**: چنانچه سیستم عامل نیاز به آزاد سازی حافظه اصلی داشته باشد مقداری از فرآیند های موجود در حالت Wait را بصورت موقت از حافظه اصلی به حافظه جانبی منتقل میکند بنابراین اسم و آدرس این فرآیند ها از حالت wait به حالت suspend wait منتقل می شود. پس در حالت suspend wait اسم و آدرس فرآیند هایی قرار دارد که در حال عمل I/O باشند، منتظر شروع عمل I/O باشند و یا موقّتا بدلیلی غیرقابل اجرا باشند. بعلاوه قسمتی از آنها به حافظه جانبی منتقل شده است .
- **حالت آماده معلق یا Suspend Ready:** چنانچه حافظه مورد نیاز سیستم عامل با انتقال فرآیند های حالت wait به حالت Suspend wait آزاد نشود، این بار مقداری از فرآیند های موجود در حالت Ready از حافظه اصلی به حافظه جانبی منتقل می شود و اسم و آدرس آنها از حالت Ready به حالت Suspend Ready منتقل می گردد .

**دلایل دیگری نیز برای تعلیق فرآیند وجود دارد که در جدول ذیل آمده اند.**

| | |
|---|---|
| مبادله | برای آوردن فرایندی که آماده اجراست، سیستم عامل نیاز به آزاد کردن حافظه کافی دارد. |
| دلایل دیگر سیستم عامل | ممکن است سیستم عامل یک فرایند زمینه یا سودمند یا فرایندی که مظنون به ایجاد مشکل هست را معلق نماید. |
| درخواست کاربر محاوره‌ای | ممکن است کاربر بخواهد به منظور اشکال زدایی یا استفاده از منابع، اجرای برنامه‌ای را معلق نماید. |
| ترتیب زمانی | ممکن است فرایندی به‌طور دوره‌ای اجرا شود (مثل یک فرایند حسابداری یا نظارتی) و هنگامی که در انتظار اجرای بعدی است، معلق گردد. |
| درخواست فرایند پدر | ممکن است فرایندی، اجرای فرایند دیگری را که خودش ایجاد کرده است، به حال تعلیق در آورد. این کار می‌تواند برای بررسی و تغییر فرایند معلق شده یا برای هماهنگی فعالیت فرزندان صورت گیرد. |

**9) تعریف مرتبط با چرخه ی حالات فرآیند :**

1) Sleep: به انتقال فرآیند از حالت Run به حالت Wait ، Sleepگویند .
2) Wakeup : به انتقال فرآیند از حالت Wait به حالت Ready ،wake up گویند .
3) Dispatch : به انتقال فرآیند از حالتReady به حالت Run ، Dispatch گویند .
4) Run time out : به انتقال فرآیند از حالتRun به حالت Ready ، Run timeout گویند .
5) تعویض متن Context switch : به ترک کردن پردازنده توسط فرآیند جاری و اجرا شدن فرآیند دیگر توسط پردازنده تعویض متن گویند .
6) اعزام کننده Dispatcher : اعزام کننده وظیفه دارد برای عمل تعویض متن برای فرآیندی که قطع می شود اطلاعات ضروری را ذخیره کرده و برای فرآیندی که قرار است اجرا شود اطلاعات ضروری را بار کند ، این عمل که نیاز به کمی زمان دارد و باعث بیکار شدن پردازنده در حین تعویض متن می شود پس هر چه زمان تعویض متن بیشتر شود کارایی پردازنده کمتر می شود .
7) بلاک کنترل فرآیند process control black(PCB) : ساختمان داده پیچیده ای است که اطلاعات ضروری برای هر فرآیند در آن ذخیره می شود . هر فرآیند ، PCB منحصر به خود را دارد . هر اطلاعاتی که سیستم عامل برای مدیریت فرآیند به آن نیاز دارد در PCB آن فرآیند ذخیره میشود. محتویات PCB در ادامه آمده است.

8) زمان انتظار (Waiting time) : به مجموع زمان هایی که یک فرآیند در وضعیت آماده یا Ready به سر می برد زمان انتظار گویند .

10)زمان بازگشت یا زمان کامل شدن (turn-around time) : از لحظه ی ورود فرآیند به سیستم تا لحظه ی خاتمه یافتن فرآیند زمان بازگشت یا زمان کامل شدن گویند . این پارامتر زمانی بیشتر در سیستم‌های دسته ای اهمیت دارد.

11)زمان پاسخ ( Response time): به فاصله ی زمانی ورود فرآیند به سیستم تا تولید اولین خروجی برای کاربر زمان پاسخ گویند . این پارامتر زمانی بیشتر در سیستم‌های اشتراک زمانی و محاوره‌ای اهمیت دارد.

## شناسایی فرایند

### شناسه‌ها

شناسه‌های عددی زیر ممکن است با جدول کنترل فرایند ذخیره شود :

- شناسه فرایند
- شناسه فرایندی که این فرایند را ایجاد کرده است (فرایند پدر)
- شناسه کاربر

## اطلاعات وضعیت پردازنده

### ثباتهای قابل رؤیت برای کاربر

ثبات قابل رؤیت توسط کاربر، ثباتی است که به‌وسیله دستورالعملهای زبان ماشین بتواند مورد مراجعه قرار گیرد. معمولاً ۸ تا ۳۲ عدد از این ثباتها وجود دارد. گرچه بعضی پیاده‌سازیهای RISC، بیش از ۱۰۰ ثبات دارند.

### ثباتهای کنترل و وضعیت

بعضی از ثباتهای پردازنده برای کنترل عمل پردازنده به کار می‌روند، که عبارتند از:

- **شمارنده برنامه** : حاوی آدرس دستورالعمل بعدی که باید واکش شود.
- **کدهای شرایط** : نتیجه آخرین عمل محاسباتی یا منطقی (مثل علامت، صفر، رقم نقلی، مساوی، سرریز)
- **اطلاعات وضعیت** : از جمله پرچمهای ازکارانداختن و به کارانداختن وقفه، حالت اجرا

### اشاره‌گرهای پشته

به هر فرایند یک یا چند پشته نسبت داده می‌شود. پشته برای ذخیره‌سازی پارامترها و آدرسهای فراخوانیهای رویه و سیستم به‌کار می‌رود. اشاره‌گر پشته به بالای آن اشاره می‌کند.

## اطلاعات کنترل فرایند

### اطلاعات زمانبندی و حالت

اطلاعاتی که سیستم عامل برای انجام زمانبندی نیاز دارد. اقلام اطلاعاتی متداول عبارتند از :

- **وضعیت فرایند** : معرف آمادگی فرایند برای زمانبندی و اجراست (مثل در حال اجرا، آماده، منتظر، متوقف).
- **اولویت** : ممکن است از بعضی اقلام برای بیان اولویت زمانبندی فرایند، استفاده شود. در بعضی سیستمها، چند مقدار لازم است (مثل مقدار جاری، مقدار پیش فرض، بالاترین مقدار مجاز).
- **اطلاعات مربوط به زمانبندی** : این اطلاع به الگوریتم زمانبندی مورد استفاده بستگی دارد. مثل مدت زمانی که این فرایند منتظر بوده است و یا مقدار زمانی که آخرین بار اجرا شده است.
- **حادثه** : شناسه حادثه‌ای که تا بروز نکند این فرایند باید در انتظار بماند.

### ساختمان داده‌ها

ممکن است یک فرایند در صف، حلقه یا ساختمان داده‌های دیگری به فرایندهای دیگر پیوند خورده باشد. برای مثال، تمام فرایندهایی که از یک سطح اولویت هستند، ممکن است در یک صف به یکدیگر پیوند خورده باشند. یا ممکن است رابطه پدر - فرزندی بین فرایندها بخواهد نمایش داده شود. برای حمایت از این ساختمان داده‌ها، ممکن است بلوک کنترل فرایند اشاره‌گرهایی به فرایندهای دیگر داشته باشد.

### ارتباط بین فرایندها

برای ارتباط بین دو فرایند مستقل، ممکن است پرچمها، علائم و پیامهای مختلفی مطرح باشد. ممکن است تمام یا بخشی از این اطلاعات در بلوک کنترل فرایند نگهداری شود.

### امتیازات فرایند

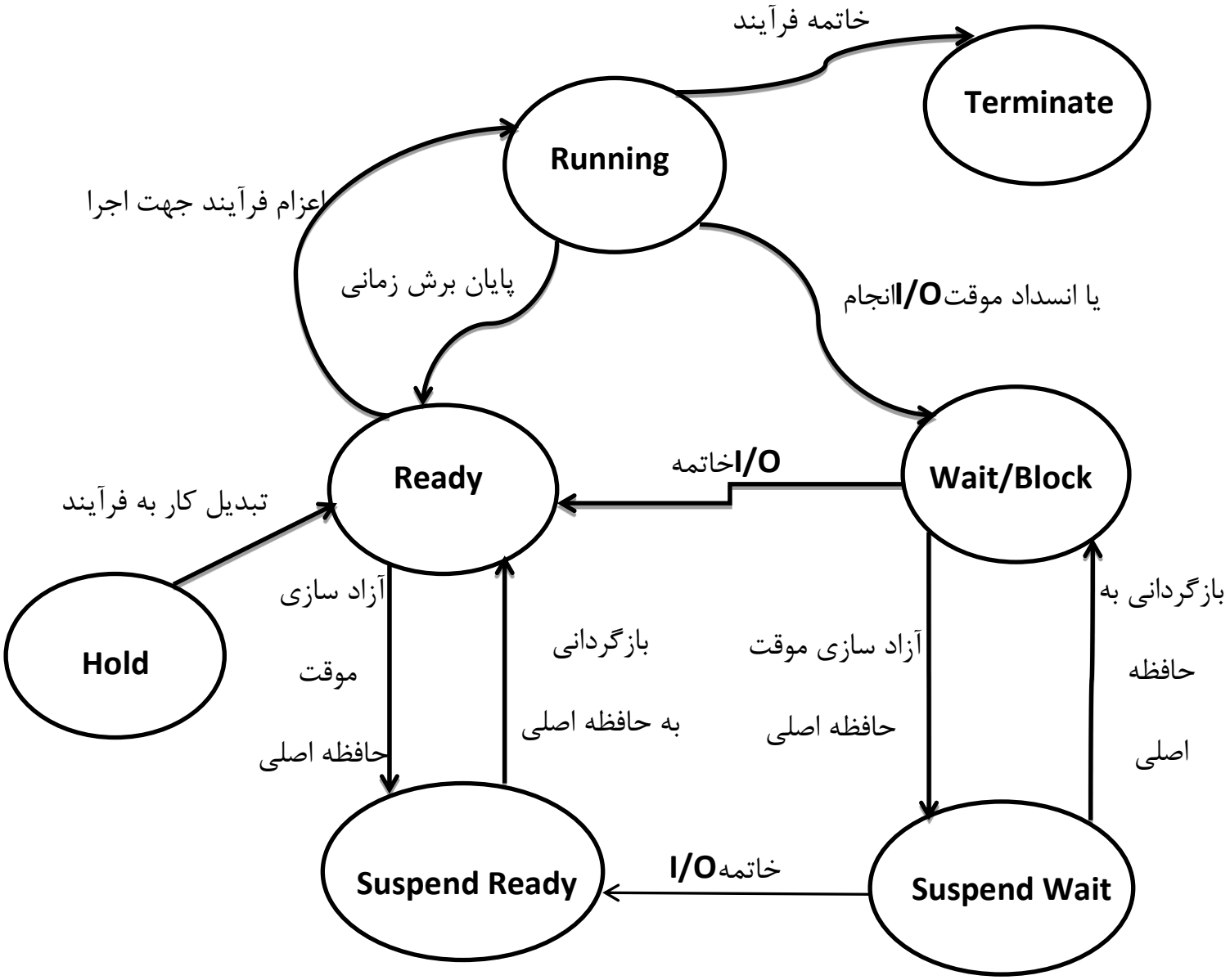
ممکن است به یک فرایند، بر حسب حافظه‌ای که می‌تواند دستیابی کند، انواع دستورالعملهایی که می‌تواند اجرا نماید، امتیازاتی داده شود. به‌علاوه ممکن است این امتیازات در استفاده از برنامه‌های سودمند و خدمات سیستم نیز اعمال گردد.

### مدیریت حافظه

این بخش می‌تواند شامل اشاره‌گرهایی به جداول قطعه و یا جداول صفحه حافظه مجازی اختصاص یافته به این فرایند باشد.

### مالکیت و استفاده از منابع

منابعی که به‌وسیله این فرایند کنترل می‌شود (مثل پرونده‌ها) می‌تواند در اینجا نشان داده شود. ممکن است تاریخچه استفاده از پردازنده یا دیگر منابع نیز نگهداری شود؛ ممکن است چنین اطلاعاتی برای زمانبندی لازم باشد.


خاتمه فرآیند
Terminate
Running
اعزام فرآیند جهت اجرا
پایان برش زمانی
یا انسداد موقتI/Oانجام
Ready
I/Oخاتمه
Wait/Block
تبدیل کار به فرآیند
Hold
آزاد سازی
موقت
حافظه اصلی
بازگردانی
به حافظه اصلی
آزاد سازی موقت
حافظه اصلی
بازگردانی به
حافظه
اصلی
Suspend Ready
I/Oخاتمه
Suspend Wait

# فصل سوم: زمانبندی

**1) زمان بندی (scheduling)**

با توجه به اینکه در سیستم‌هایی که خاصیت چندبرنامگی دارند در هر لحظه ممکن است چندین فرآیند متقاضی استفاده از پردازنده و یا منبعی باشند، باید سیستم عامل در خصوص اینکه کدام فرآیند از آن منبع یا پردازنده استفاده کند تصمیم گیری کند؛ به این تصمیم گیری زمان‌بندی گفته می‌شود. به عبارت دیگر به مدیریت فرآیند در چرخه حالات فرآیند که طی آن منابع و پردازنده در اختیار فرآیند قرار میگیرند و از او گرفته می شوند زمانبندی گویند . اهداف عمده ی زمانبندی کاهش زمان پاسخ کاربر و افزایش کارایی سیستم می باشد .

**2) اهداف زمان بندی :** در حالت کلی میتوان اهداف زمان بندی را بصورت ذیل بیان نمود:

1)کاهش زمان پاسخ کاربر: فاصله زمانی میان درخواست اجرای برنامه از سوی کاربر و تولید خروجی توسط سیستم کوتاه گردد.

2)افزایش کارایی پردازنده: در صورت وجود فرآیند جهت اجرا، تا حدامکان پردازنده بیکار(آزاد) نباشد.

3)توازن در استفاده از منابع: در صورت وجود کار برای منابع سیستم کامپیوتری، این منابع مشغول باشند.

4)رسیدن به مهلت های زمانی: در صورتی که برای اجرای فرآیندها مهلت زمانی تعریف شده باشد، بتوان این فرآیندها را در مهلت زمانی تعیین شده اجرا نمود (همانند سیستم‌های بلادرنگ)

**3) معیارهای زمانبندی** : الگوریتم‌های زمان بندی باید دارای معیارهای زیر باشند تا الگوریتم‌های مناسبی به شمار آیند.

## معیارها از دیدگاه کاربر و مربوط به کارآمدی

**زمان پاسخ** برای یک فرایند محاوره‌ای زمان پاسخ فاصله زمانی ارائه یک تقاضا تا شروع دریافت پاسخ آن می‌باشد. اغلب، یک فرایند در حالی شروع به تولید خروجی می‌کند که پردازش آن تقاضا ادامه دارد. پس، از دید کاربر این معیار بهتر از معیار زمان کل است. نظام زمانبندی باید سعی در دستیابی به زمان پاسخ کم و حداکثرسازی تعداد کاربران محاوره‌ای، با توجه به زمان پاسخ قابل قبول، داشته باشد.

**زمان کل** فاصله زمانی بین پذیرفتن یک فرایند تا تکمیل آن است. این زمان شامل زمان واقعی اجرا و زمان صرف شده جهت انتظار برای منابع (از جمله پردازنده) می‌باشد. زمان کل معیار مناسبی برای کارهای دسته‌ای است.

**آخرین مهلت** هنگامی که آخرین مهلت تکمیل فرایند بتواند مشخص شود، نظام زمانبندی باید در جهت حداکثرسازی درصد ارضای آخرین مهلتها، به اهداف دیگر کمتر اهمیت دهد.

## معیارهای دیگر از دیدگاه کاربر

**قابلیت پیش‌بینی** یک کار داده شده باید در زمان تقریباً یکسان و با بهای تقریباً یکسان و مستقل از بار روی سیستم انجام گیرد. تغییرات شدید زمان پاسخ و زمان کل می‌تواند باعث آزار کاربر شود. ممکن است از تغییرات شدید بار کاری سیستم به‌عنوان علامت ضرورت تنظیم سیستم تلقی شود.

## معیارها از دیدگاه سیستم و مربوط به کارآمدی

**توان عملیاتی** سیاست زمانبندی باید سعی در حداکثرسازی تعداد فرایندهای کامل شده در واحد زمان داشته باشد. این معیار مبین مقدار کار انجام شده است و به روشنی به متوسط طول فرایندها بستگی دارد، ولی توسط سیاست زمانبندی نیز تحت تأثیر قرار می‌گیرد، که به نوبه خود می‌تواند در بهره‌وری مؤثر باشد.

**استفاده ازپردازنده** مبین درصد زمانی است که پردازنده مشغول می‌باشد و برای سیستمهای اشتراکی گران‌قیمت، یک معیار مهم است. در سیستمهای تک‌کاربره و همچنین برخی سیستمهای دیگر، مثل سیستمهای بلادرنگ، این معیار اهمیت کمتری دارد.

## معیارهای دیگر از دیدگاه سیستم

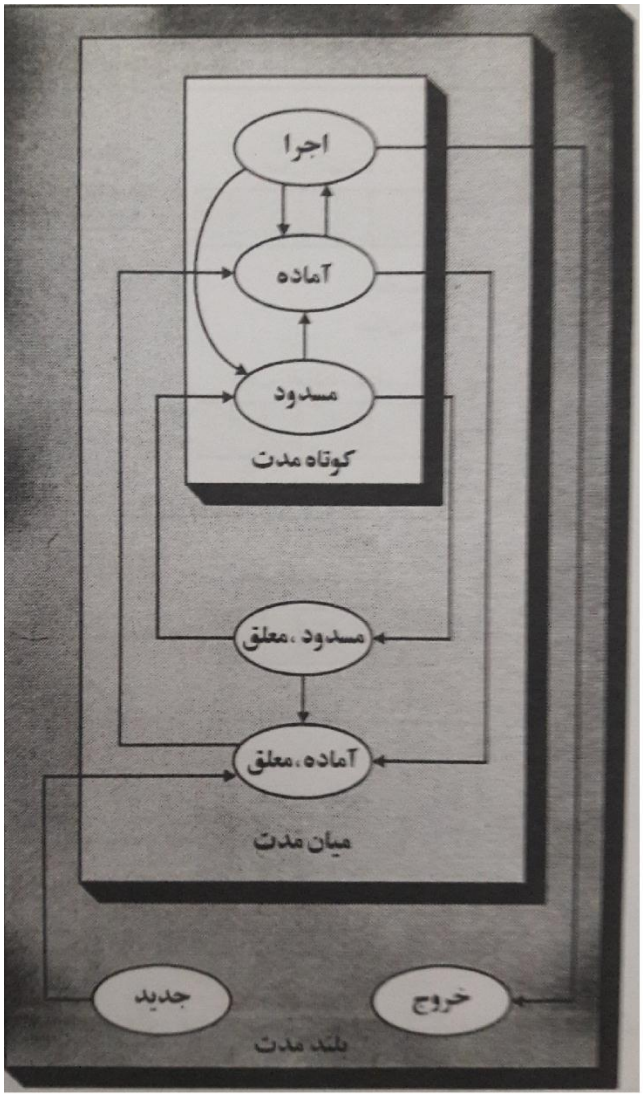
**عدالت** در غیاب رهنمودهای کاربر یا در غیاب رهنمودهای ارائه شده توسط سیستم، با فرایندها باید یکسان برخورد شود و هیچ فرایندی نباید از گرسنگی رنج برد.

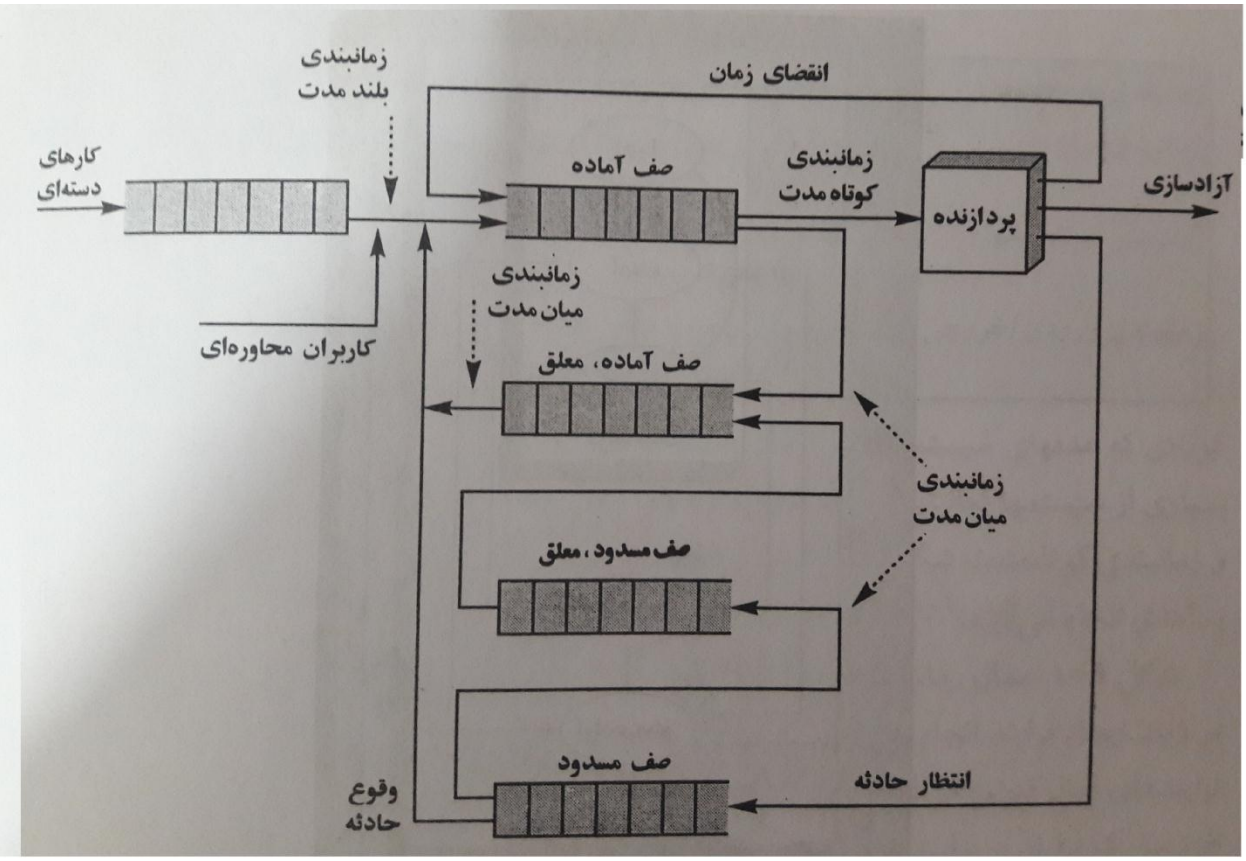
**اعمال اولویتها** زمانی که به فرایندها اولویت داده شود، سیاست زمانبندی باید فرایندهای با اولویت بالاتر را مقدم بدارد.

**توازن در منابع** سیاست زمانبندی باید منابع سیستم را مشغول نگه‌دارد. فرایندهایی که از منابع بسیار مشغول، کمتر استفاده می‌کنند، باید مقدم شمرده شوند. این معیار شامل زمانبندهای میان‌مدت و بلندمدت نیز می‌شود.

**4) سطوح زمانبندی :** زمان بندی در یک سیستم‌عامل دارای سطوح ذیل بوده که در ادامه شرح داده خواهد شد.

| | |
|---|---|
| **زمانبندی بلندمدت** | تصمیم‌گیری در مورد افزودن به مجموعه فرایندها برای اجرا |
| **زمانبندی میان‌مدت** | تصمیم‌گیری در مورد افزودن به تعداد فرایندهایی که بخشی یا تمام آنها در حافظه اصلی است. |
| **زمانبندی کوتاه‌مدت** | تصمیم‌گیری در مورد اینکه کدام یک از فرایندهای موجود در حافظه اصلی، برای اجرا توسط پردازنده انتخاب شود. |
| **زمانبندی ورودی/خروجی** | تصمیم‌گیری در مورد اینکه کدام درخواست ورودی/خروجی فرایندها به وسیله یک دستگاه ورودی/خروجی موجود انجام گیرد. |






**الف- زمان بند ورودی/خروجی (I/O Scheduler):** در یک سیستم چندبرنامه ای ممکن است در هر لحظه چندین فرآیند متقاضی استفاده از یک دستگاه ورودی/خروجی باشند (مثلا بخواهند عمل چاپ انجام دهند یا فایلی را از دیسک بخوانند) و برای استفاده از آن درخواست کرده باشند. بنابراین سیستم عامل برای استفاده از هر دستگاه ورودی/خروجی یک صف در نظر گرفته و درخواست‌های استفاده از هر دستگاه را در صف مربوطه‌اش قرار می‌دهد و یا بعبارت دیگر فرآیندهای متقاضی استفاده از یک دستگاه در صف مربوط به آن دستگاه مسدود می‌شوند. وظیفه زمان‌بند ورودی/خروجی این است که در مورد اینکه درخواست کدام فرآیند بوسیله دستگاه مربوطه انجام گیرد، تصمیم‌گیری نماید.

**ب- زمان بند بلند مدت یا زمان بند کار (Long term scheduler یا Job scheduler) :** این زمان‌بند در سیستم‌های دسته ای وظیفه دارد از میان برنامه های درخواست شده برای اجرا تعدادی را انتخاب کرده

و آنها را به حافظه اصلی منتقل کند و تبدیل به فرآیند نماید. نام دیگر این زمان‌بند، زمان بند *سطح بالا* است. **این کار بر اساس سه الگوریم زیر صورت میگیرد :**

1) **FIFO**: این الگوریتم کاری که زودتر برای اجرا در خواست شده است را زودتر انتخاب میکند ( زودتر وارد شده ، زودتر خارج شده ) این الگوریتم عدالت دارد اما اگر کارهای سر صف طولانی باشند زمان انتظار کارهای انتهای صف زیاد می شود .

2) **SJF (Shortest Jof First)** : این الگوریتم ابتدا برای کارهای درخواست شده زمان اجرا را تخمین می زد و سپس کار با کوتاهترین اجرا را انتخاب می کند . این الگوریتم عدالت ندارد و باعث گرسنگی برای کارهای طولانی می شود همچنین به دلیل نیاز به تخمین زمان اجرا قابل پیاده سازی عملی نیست . **MIXED (ترکیبی) :** این الگوریتم سعی در برقراری توازن در سیستم داشته و ترکیبی از کارهای با تنگنای پردازشی و ورودی و خروجی را انتخاب میکند و با کاهش بیکاری سیستم باعث افزایش کارایی میشود. شایان ذکر است که وجود راهی برای تعیین وجود کارهای با تنگنای پردازشی و ورودی و خروجی وجود ندارد و این الگوریتم از نظر عملی قابل پیاده‌سازی نمی‌باشد.

**نکته :** کنترل درجه ی چند برنامه ای بر عهده ی زمان بند بلند مدت است .

**نکته:** در سیستم های اشتراک زمانی و محاوره‌ای و یا سیستم‌های شخصی (Personal Computer) این زمان بند با انتخاب کار از سوی کاربر بلافاصله آن را به فرآیند تبدیل می‌کند و به نحوی انتخاب کار جهت اجرا برعهده کاربر گذاشته شده است.

**نکته:** زمان بند بلند مدت بین انتقال از حالت پذیرش به آماده عمل می‌کند.

**ج)زمانبند کوتاه مدت یا زمان بند پردازنده (Short term scheduler) :**

این زمان بند وظیفه دارد از میان فرآیند هایی که نام آنها در وضعیت Ready است یکی را انتخاب کند تا پردازنده آن را اجرا کند. بعبارت دیگر این زمان‌بند وظیفه دارد با انتخاب فرآیندها از صف آماده و تحویل آنها به پردازنده (در صورت وجود فرآیند) از بیکاری پردازنده تا حد امکان جلوگیری نماید. ، اما در چه زمانی

زمان بند کوتاه مدت عمل خود را انجام می دهد ؟ هنگامی که پردازنده بیکار شود زمانبند کوتاه مدت عمل میکند . اما بیکار شدن پردازنده که باعث می شود زمانبند کوتاه مدت عمل کند در چهار مورد زیر رخ میدهد :

1) فرآیند در حال اجرا به هر دلیلی خاتمه یابد و از سیستم خارج شود، یعنی ازحالت Run به حالت terminate انتقال یابد، پس پردازنده بیکار شده و باید زمان‌بند فرآیند بعدی از صف آماده را برای اجرا انتخاب نماید.
2) فرآیند در حال اجرا عمل I/O دارد یعنی فرآیند در حال اجرا از حالت Run به حالت Wait منتقل شده و در این حالت بلوکه می‌شود، پس پردازنده بیکار شده و باید زمان‌بند فرآیند بعدی از صف آماده را برای اجرا انتخاب نماید.
3) فرآیند در حال اجرا اتمام برش زمانی دارد، یعنی با وقوع وقفه ساعت سیستم عامل متوجه می‌شود که برش زمانی اختصاص یافته به فرآیند در حال اجرا پایان یافته است و باید این فرآیند از حالت Run به حالت Ready منتقل شده و فرآیند بعدی در سر صف Ready برای اجرا انتخاب گردد .
4) فرآیند با اولویت بالاتر وارد صف آماده شود؛ یعنی فرآیند جدیدی تولید شده است که اولویت آن از فرآیند در حال اجرا بیشتر است. البته اگر فرآیندی که بدلیل نیاز به وقوع حادثه ای مانند انجام عمل ورودی/خروجی در حالت Wait بوده باشد و با پایان عمل ورودی/خروجی اش به حالت Ready منتقل شود(یعنی از Wait به Ready منتقل شود) و اولویتش از فرآیند در حال اجرا بیشتر باشد باز هم زمان‌بند باید فرآیند در حال اجرا را قطع کرده و فرآیند ورودی به صف آماده را اجرا نماید.

واضح است که در موارد اول و دوم فرآیندی که پردازنده را ترک میکند با میل خودش است، یعنی فرآیند در حال اجرا زمانی از پردازنده خارج می‌شود که خودش تصمیم گرفته باشد و سیستم عامل آن را به زور از پردازنده خارج نکرده است (پردازنده را از او پس نگرفته است). اما در موارد 3 و 4 فرآیندی که اجرایش

متوقف شده به زور پردازنده را رها کرده است، یعنی خودش هنوز تمایل به اجرا داشته و بدلیل اتمام برش زمانی یا ورود فرآیند با اولویت بالاتر پردازنده را رها کرده است.

**بنابراین، در حالت کلی دو خانواده از الگوریتم های زمان بند کوتاه مدت وجود دارد :**

1) **الگوریتم های انحصاری یا غیر قابل پس گرفتنی (Non-Preemptive)** : در این الگوریتم ها وقتی زمان بند پردازنده را در اختیار فرآیندی قرار میدهد اجازه میدهد این فرآیند اجرا شود تا زمانی که فرآیند به دلخواه خود پردازنده را رها کند یعنی فرآیند در حال اجرا عمل ورودی / خروجی داشته باشد یا خاتمه یابد یعنی فقط در حالت‌های 1 و 2 از چهار حالت گفته شده عمل می‌کند . الگوریتم های این خانواده عبارت اند از :

A) **الگوریتم FIFO یا سرویس به ترتیب اجرا FCFS (First Come First Service)**: این الگورتیم فرآیندهای جدید را در انتهای صف آماده قرار داده و فرآیندها را به همان ترتیب که وارد صف آماده شده اند از صف آماده انتخاب می کند و پردازنده را در اختیار آنها قرار می دهد و دارای این ویژگی ها است :

1) نیاز به دانستن زمان اجرا برای انتخاب فرآیند بعدی ندارد .
2) قابل پیاده سازی است و پیاده سازی آن آسان است.
3) عدالت دارد یعنی فرآیندی که زودتر وارد صف شده است زودتر اجرا می‌شود.
4) گرسنگی و قحطی زدگی ندارد یعنی امکان ندارد که فرآیندی زودتر وارد شده باشد و بدلیل اجرای فرآیندهای دیرتر وارد شده، اجرایش بصورت نامحدود به تعویق بیافتد.
5) در این الگوریتم چون اجرا به ترتیب ورود است، اگر فرآیند های سر صف طولانی باشند (زمان اجرای آنها زیاد باشد) آنگاه زمان انتظار فرآیند های ته صف زیاد شده و بنابراین میانگین زمان انتظار فرآیندها زیاد می‌شود.

6) این الگوریتم بیشتر برای فرآیندهای با تنگنای پردازشی مناسب است، چرا که فرآیندهای با تنگنای ورودی/خروجی مقدار زمان کمی را صرف انجام عملیات پردازشی کرده و سپس برای انجام عملیات ورودی/خروجی مسدود شده و به حالت wait منتقل شده و پس از انجام عمل ورودی/خروجی به انتهای صف Ready منتقل می‌شوند بنابراین زمان انتظار آنها زیاد خواهد شد.

7) چون این الگوریتم انحصاری است نمی‌توان از آن در سیستم‌های محاوره‌ای و اشتراک زمانی استفاده کرد و برای سیستم‌های دسته‌ای مناسب است.

B)**الگوریتم SPN,SPT یا اول کوتاهترین فرآیند (Shortest Process Next):** این الگوریتم برای فرآیند های موجود در وضعیت آماده زمان اجرا را تخمین می زند، سپس فرآیند با کمترین اجرا انتخاب می شود، اگر زمان اجرای دو فرآیند با هم برابر باشد آنگاه فرآیندی انتخاب میشود که زمان ورود کوچکتری دارد. این الگوریتم دارای این ویژگی ها است :

1- نیاز به دانستن زمان اجرا برای انتخاب فرآیند بعدی دارد.

2- چون نیاز به دانستن زمان اجرا برای انتخاب فرآیند دارد، قابل پیاده سازی عملی نیست.

3-اولویتی است یعنی اولویت را به فرآیندهای دارای زمان اجرای کمتر میدهد.

4- بدلیل اینکه اولویت را به فرآیندهای کوتاه‌تر میدهد، غیر عادلانه است.

5- باعث قحطی زدگی فرآیندهای طولانی می شود، یعنی ممکن است اجرای یک فرآیند طولانی بدلیل ورود مداوم فرآیندهای کوتاهتر تا مدت زمان زیادی به تعویق بیافتد.

6-با بکارگیری این الگوریتم میانگین زمان انتظار و زمان پاسخ فرآیندها کمترین مقدار پس از SRTF خواهد شد.

7- این الگوریتم نیز بدلیل انحصاری بودن برای سیستم‌های اشتراک زمانی مناسب نیست.

C)**الگوریتم اول بالاترین نسبت پاسخHRRN (یا Highest Response Raito Next)**: این الگوریتم برای فرآیندهای وضعیت Ready ابتدا زمان اجرا را تخمین می‌زند و پس از تخمین زمان اجرا نسبت پاسخ را محاسبه کرده و فرآیند با نسبت پاسخ بالاتر جهت اجرا انتخاب می شود.

$$\text{نسبت پاسخ } RR = \frac{W+S}{S} = \frac{\text{زمان اجرا + زمان انتظار در صف تاکنون}}{\text{زمان اجرا}} = 1 + \frac{W}{S}$$

با توجه به فرمول فوق فرآیندهای کوتاهتر که زمان اجرای کمتری دارند (S کوچکتر) دارای نسبت پاسخ بالاتری بوده و نسبتا زودتر اجرا می‌شوند (چون S در مخرج است، با کوچکتر شدن آن مقدار RR افزایش پیدا می کند). از طرف دیگر وقتی فرآیندهای بزرگ در صف انتظار می‌کشند (در حین اجرای فرآیندهای کوتاهتر) زمان انتظار (w) آنها افزایش یافته و بهمین ترتیب نسبت پاسخ آنها نیز افزایش می‌یابد (چون w در صورت است با افزایش w ، RR هم افزایش پیدا می کند). بنابراین فرآیندهای طولانی هم اجرا شده و دچار قحطی زدگی نمی‌شوند. بعبارت دیگر هدف از طراحی این الگوریتم ایجاد ترکیبی از دو الگوریتم SPN و FCFS بوده است؛ چرا که الگوریتم SPN به نفع فرآیندهای کوتاهتر عمل می‌کند و الگوریتم FCFS بیشتر به نفع فرآیندهای طولانی است.

ویژگیهای الگوریتم عبارتند از:

1- نیاز به دانستن زمان اجرا برای انتخاب فرآیند بعدی دارد.

2- چون نیاز به دانستن زمان اجرا برای انتخاب فرآیند دارد، قابل پیاده سازی عملی نیست.

3- غیراولویتی و عادلانه است، یعنی بصورت عادلانه به فرآیندهای کوتاه و طولانی اولویت اجرا میدهد.

4- قحطی زدگی و گرسنگی ندارد.

5- سربار آن زیاد است، چون برای هر بار انتخاب فرآیند بعدی جهت اجرا باید ابتدا زمان انتظار کلیه فرآیندهای موجود در صف آماده محاسبه شده و سپس نسبت پاسخ محاسبه گردد سربار این الگوریتم زیاد است.

6- این الگوریتم نیز بدلیل انحصاری بودن برای سیستم‌های اشتراک زمانی مناسب نیست.

**تکنیک سالمندی برای تخمین زمان اجرا:** همانطور که پیشتر گفته شد الگوریتم‌های SPN و HRRN بدلیل نیاز به دانستن زمان اجرا قابل پیاده‌سازی نیستند، لیکن در شرایطی در سیستم‌های محاوره‌ای و اشتراک زمانی سیستم عامل می‌تواند زمان اجرای فرآیند در مراحل قبلی را نگهداری کرده و بوسیله فرمول زیر زمان اجرا برای مرحله بعدی را تخمین بزند:

$$S_{n+1} = \alpha T_n + (\alpha - 1)S_n$$

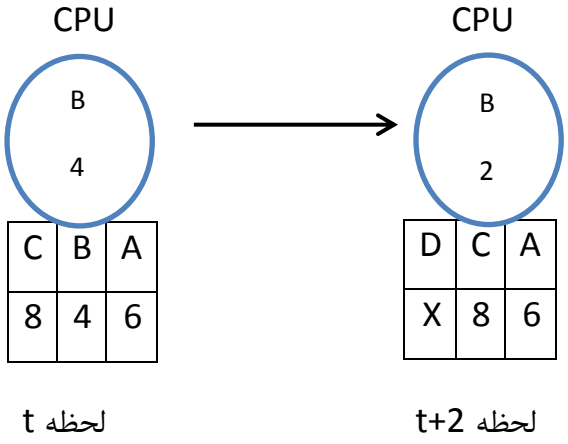
در فرمول فوق $S_{n+1}$ زمان اجرای تخمینی برای مرحله بعدی (n+1)، $S_n$ زمان اجرای تخمینی برای مرحله فعلی (n)، $T_n$ زمان اجرای واقعی مرحله فعلی و $0 < \alpha < 1$ نسبت وزنی است که با تغییر آن میزان تاثیر زمان واقعی یا زمان تخمینی مرحله فعلی در زمان تخمینی مرحله بعدی تغییر می‌کند.

**2) الگوریتم های غیر انحصاری یا قابل پس گرفتنی یا Preemtive** :در این الگوریتم ها هر گاه فرآیند در حال اجرا به دلخواه خود پردازنده را ترک نماید الگوریتم زمان‌بندی فعال شده و فرآیند بعدی جهت اجرا را انتخاب می‌کند. علاوه بر این، الگوریتم زمان بندی می تواند در هر لحظه در حین اجرای فرآیند که سیستم‌عامل بخواهد پردازنده را به زور از اختیار فرآیند در حال اجرا خارج کرده و در اختیار فرآیند دیگری قرار دهد. در چنین شرایطی فرآیندی که از پردازنده خارج شده است (نیمه کاره است) به صف آماده برگردانده می‌شود تا در آینده مجددا توسط زمان‌بند برای اجرا انتخاب شود. بعبارت دیگر این الگوریتم در هر چهار حالت بیکار شدن پردازنده فعال می‌شود. بناراین مسئله ی اولویت ها و برش زمانی در این خانواده مطرح می شود و شامل الگوریتم های زیر می باشد:

A) **الگوریتم کوتاهترین زمان باقیمانده( SRTF یا Shortest Remainung Time First)**: هنگامی که پردازنده آزاد باشد (مشغول اجرای فرآیندی نباشد) این الگوریتم برای انتخاب فرآیند از حالت آماده، فرآیندی را انتخاب می کند که زمان اجرایش کوتاهتر باشد. علاوه بر این، اگر در حین اجرای یک فرآیند، فرآیندی وارد حالت آماده شود که زمان تخمینی اجرایش از باقیمانده زمان اجرای فرآیند در حال اجرا کوتاهتر باشد، فرآیند

در حال اجرا قطع شده به حالت آماده برمی گردد و فرآیند دوم اجرا می شود. در غیر این صورت فرآیند در حال اجرا قطع نشده و ادامه می‌یابد. این الگوریتم همه ویژگی های الگوریتم SPN را دارد ولی غیر انحصاری است. در واقع همان الگوریتم SPN است با این تفاوت که SPN انحصاری ولی SPTF غیر انحصاری است .

مثال: در لحظه t سه فرآیند A,B,C با زمانهای اجرای به ترتیب 8، 4، 6 ثانیه در صف آماده وجود دارد و پردازنده بیکار است. فرآیند کوتاهتر یعنی B برای اجرا انتخاب می‌شود. حال اگر 2 ثانیه بعد از اجرای B، فرآیند D با زمان اجرای X ثانیه وارد صف آماده شود آنگاه 2 ثانیه باقیمانده زمان اجرای B است. در نتیجه اگر $X<2$ باشد،آنگاه B قطع شده و D اجرا میشود، اگر $X\geq2$ باشد، B ادامه میدهد .




B)**الگوریتم گردش نوبت یا Round Robin :** در این روش حالت (صف) آماده بصورت FiFo است. زمان توسط سیستم عامل به قسمت های مساوی بنام برش زمانی یا کوانتوم تقسیم می شود. فرآیند سر صف آماده برای اجرا انتخاب شده و به اندازه ی یک برش زمانی پردازنده در اختیارش قرار می‌گیرد. اگر تا پایان برش زمانی فرآیند به دلخواه خودش پردازنده را ترک نکند (خاتمه نیافته و یا برای انجام عمل ورودی/خروجی به حالت wait نرود)، اجرای آن فرآیند قطع شده و پردازنده به زور از فرآیند گرفته شده و فرآیند به انتهای صف آماده می رود. سپس فرآیند بعدی سر صف برای اجرا انتخاب شده و پردازنده در اختیارش قرار می‌گیرد. ویژگی های آن عبارتند از :

1- این الگوریتم غیرانحصاری بوده وبرای استفاده در سیستم‌های اشتراک زمانی مناسب است.

2- نیاز به دانستن زمان اجرای فرآیندها ندارد و قابل پیاده‌سازی عملی است.

3- عادلانه و غیر اولویتی است و باعث قحطی زدگی فرآیندها نمی‌شود.

4- معمولا زمان پاسخ فرآیندها در این الگوریتم کاهش می‌یابد.

5- مقدار برش زمانی باید با دقت انتخاب گردد.

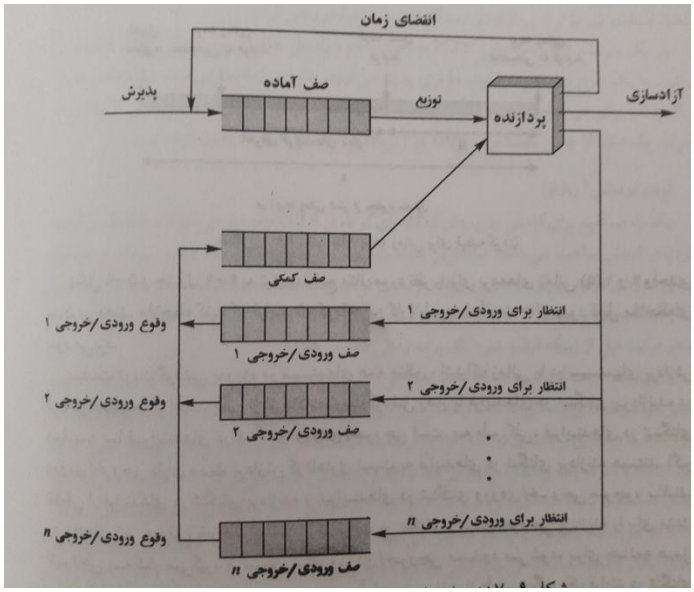
**برای بکارگیری الگوریتم گردش نوبت باید به نکات زیر توجه نمود:**

1- اگر مقدار برش زمانی از میانگین زمان اجرای فرآیندها کمتر باشد، کارایی پردازنده از رابطه $Ra_{cpu} = \frac{\mathrm{Q}}{Q+CS}$ محاسبه می‌شود که در این رابطه، Q: مقدار زمان برش زمانی و CS : مقدار زمان تعویض متن است. اگر مقدار برش زمانی از میانگین زمان اجرای فرآیندها بیشتر باشد، کارایی پردازنده از رابطه $Ra_{cpu} = \frac{\mathrm{S}}{S+CS}$ محاسبه می‌شود که در این رابطه، S: میانگین زمان اجرای فرآیندها و CS : مقدار زمان تعویض متن است.

2- اگر مقدار برش زمانی(کوانتوم) اکوچک باشد باعث افزایش تعداد تعویض متن، افزایش بیکاری پردازنده، کاهش کارایی پردازنده و افزایش زمان پاسخ کاربر میشود. اگر مقدار برش زمانی(کوانتوم) برزگ باشد هر فرآیند آنقدر اجرا شده تا به دلخواه خود پردازنده را ترک نماید بنابراین باعث تبدیل الگورتیم گردش نوبت به الگوریتم FIFO می‌شود، همچنین باعث کاهش بیکاری پردازنده و افزایش نسبی کارایی آن خواهد شد. حد پایین (حداقل) مقدار برش زمانی کمی بیش از زمان تعویض متن و مقدار ایده آل آن کمی بیش از کمترین زمان اجرای کوتاهترین فرآیند است.

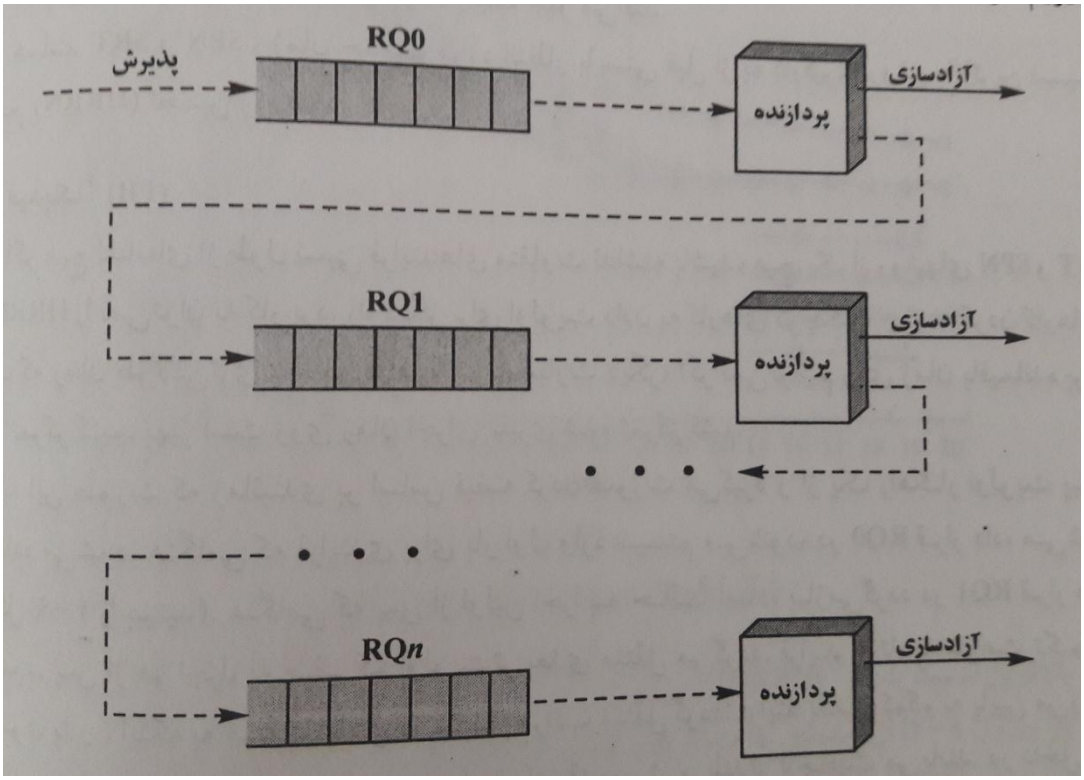
3- اگر دوفرآیند بصورت همزمان، یکی از پردازنده پس از اتمام برش زمانی و دیگری از حالت جدید، بخواهند وارد صف آماده شوند آنگاه فرآیند تازه وارد در صف آماده جلوتر قرار می‌گیرد.

**زمان بندی گردش نوبت مجازی (Virtual Round Robin یا VRR):** در الگوریتم زمان‌بندی گردش نوبت معمولی هنگامی که فرآیندهای با تنگنای ورودی/خروجی در میان فرآیندهای با تنگنای پردازشی (بویژه پشت سرآنها) در صف آماده قرار می‌گیرند، پس از تحمل انتظار در صف آماده وارد پردازنده می‌شوند اما بدلیل اینکه نیاز کمی به عملیات پردازشی دارند سریعا به عمل ورودی/خروجی احتیاج پیدا کرده و پردازنده را ترک کرده و به حالت wait می‌روند. این درحالی است که فرآیندهای با تنگنای پردازشی بصورت کامل از برش زمانی خود استفاده می‌کنند. بنابراین فرآیندهای با تنگنای ورودی/خروجی به میزان انتظاری که متحمل می‌شوند از پردازنده استفاده نمی‌کنند و بنحوی به ضرر فرآیندهای با تنگنای ورودی/خروجی خواهد شد. برای حل این مشکل روش گردش نوبت مجازی پیشنهاد شده است. در این روش یک صف آماده کمکی در نظر گرفته می‌شود و هنگامی که فرآیندی قبل از اتمام برش زمانی خود بدلیل انجام عمل ورودی/خروجی مجبور به ترک پردازنده شود، پس از پایان عمل ورودی/خروجی اش از حالت wait به این صف منتقل می‌شود. اولویت این صف آماده کمکی از صف آماده بیشتر است یعنی تا زمانیکه در صف آماده کمکی فرآیندی وجود دارد از صف آماده اصلی فرآیندی جهت اجرا انتخاب نمی‌شود. وقتی فرآیندی از صف آماده کمکی انتخاب می‌شود و به اجرا می‌رود به اندازه کوانتوم زمانی معمول منهای زمان اجرای آخرین بار به او زمان داده می‌شود.



**C) الگوریتم صف باز خورد چند سطحی یا Multilevel Feedback Queue یا MLFQ**

چون تعیین مقدار مناسب و دقیق کوانتوم در الگوریتم گردش کار دشواری است الگوریتم صف بازخورد چندسطحی ابداع شده است. در این الگوریتم چندین صف با کوانتوم های متغیر و افزایشی داریم و همه صف‌ها به صورت FIFO مدیریت می شوند. فرآیند هایی که از وضعیت جدید وارد می شوند در بالاترین صف قرار می گیرند. به هر فرآیند که از هر صف انتخاب می شود به اندازه ی کوانتوم همان صف برش زمانی اجرا داده می شود و اگر با پایان این کوانتوم فرآیند به دلخواه خودش پردازنده را ترک نکند به زور پردازنده از اختیارش خارج شده و فرآینده به صف پایینی منتقل می شود. شرایط زیر حاکم است :




1- آخرین صف برش زمانی ندارد و بصورت FCFS عمل می‌کند. یعنی فرآیندی که از این صف انتخاب شده و اجرا شده، قطع نمی‌شود

2- تا زمانی که در صف بالاتر فرآیندی وجود داشته باشد از صف های پایینی انتخاب صورت نمی گیرد.

3- اگر فرآیندی از صف های پایینی در حال اجرا باشد و فرآیند جدیدی وارد صف اول شود، فرآیند در حال اجرا قطع شده به ابتدای صف خود باز می گردد و فرآیند تازه وارد اجرا می شود . در این حالت دفعه بعدی که فرآیند قطع شده اجرا شود، به اندازه برش زمانی آن صف منهای زمان اجرای آخرین بار به او زمان داده می‌شود.

4- اگر فرآیندی برای انجام عمل ورودی/خروجی قبل از اتمام برش زمانی اش پردازنده را ترک کرده و به حالت wait برود، پس از پایان عمل ورودی/خروجی به بالاترین صف منتقل می‌شود.

**این الگوریتم دارای ویژگی‌های زیر است:**

1-غیر انحصاری است.

2-نیاز به دانستن زمان اجرای فرآیندها ندارد و قابل پیاده سازی عملی است.

3- اولویتی و غیر عادلانه بوده و به نفع فرآیندهای با تنگنای ورودی/خروجی و باعث قحطی زدگی برای فرآیندهای طولانی است: در این روش فرآیندهای با تنگنای ورودی/خروجی به سرعت اجرا می‌شوند در حالیکه فرآیندهای با تنگنای پردازشی (طولانی) کم‌کم به صف‌های پایینی منتقل می‌شوند. بنابراین فرآیندهای جدیدتر و کوتاه‌تر بر فرآیندهای طولانی اولویت پیدا می‌کنند. با تکرار این شرایط فرآیندهای طولانی دچار قحطی زدگی می‌شوند.

4- با توجه به اینکه فرآیندهای طولانی به صف‌های پایینی منتقل می‌شوند، و در صف‌های پایینی برش زمانی طولانی تر دریافت می‌کنند تعداد تعویض متن ها بطور قابل توجهی کاهش یافته و این مساله باعث افزایش کارایی پردازنده می‌گردد.

**تکنیک سالخوردگی برای جلوگیری از قحطی زدگی فرآیندهای طولانی** :برای جلوگیری از قحطی زدگی طولانی برای فرآیندهای طولانی از تکنیکی بنام سالخوردگی استفاده می شود که در آن اگر فرآیندی برای مدت زمان طولانی بدون آنکه اجرا شود در صفی معطل باشد سیستم عامل آن را به صف بالاتر منتقل می کند.

**D) الگوریتم صف چند سطحی (Multilevel Queue یا MLQ)**

در این الگوریتم چندین صف آماده وجود دارد که صف ها از بالا به پایین را دارای اولویت هستند و اولویت ها از بالا به پایین کمتر می شود، همچنین برای هر صف تعریف می‌شود که چه فرآیندهایی باید در این صف قرار گیرند. این الگوریتم در سیستم‌هایی قابل اجراست که بتوان در آنها فرایندها را به چندین کلاس مختلف با اولویت‌های متفاوت دسته‌بندی نمود. بنابراین هر فرآیند در زمان انتقال از حالت جدید به حالت آماده، در صفی قرار می‌گیرد که برای آن نوع فرآیند تعریف شده است. در هنگام انتخاب فرآیند بعدی جهت اجرا، تا زمانیکه در صف بالاتر فرآیندی وجود دارد، از صف پایینی فرآیندی برای اجرا انتخاب نمی‌شود بعبارت دیگر زمانی فرآیندی از یک صف انتخاب می‌شود که همه صف‌های بالایی آن صف خالی شده باشند. بعلاوه، اگر فرآیندی از صفی در حال اجرا باشد و فرآیند جدیدی وارد صفی بالاتر از صف مربوط به فرآیند در حال اجرا شود، فرآیند در حال اجرا قطع شده به ابتدای صف خود باز می گردد و فرآیند تازه وارد اجرا می شود . برای انتخاب یک فرآیند از یک صف (زمان‌بندی فرآیندهای درون صف) میتوان از الگوریتم گردش نوبت (Round Robin) یا FCFS استفاده نمود. این الگوریتم دارای ویژگی‌های زیر است:

1- غیر انحصاری است.

2- اولویتی و غیرعادلانه بوده و برای فرآیندهایی که در صف‌های پایینی قرار می‌گیرند امکان قحطی زدگی است.

**E) الگوریتم زمان بندی اولویتی Priority :** در الگوریتم‌های زمان بندی اولویتی برای هر فرآیند یک عدد اولویت تعریف می‌شود که براساس یک یا چند پارامتر مشخص و از پیش تعیین شده اولویت فرآیند مشخص می‌شود(عدد اولویت مقدار می‌گیرد). حال، برای انتخاب فرآیند بعدی جهت اجرا فرآیندی انتخاب می‌شود که اولویت بالاتری داشته باشد. اگر اولویت دو فرآیند مساوی باشد، فرآیندی انتخاب می‌شود که زمان ورود کوچکتری داشته باشد یعنی زودتر وارد شده باشد. این الگوریتم‌ها ممکن است انحصاری یا غیر انحصاری باشند ولی در حالت کلی همه ی الگوریتم‌های زمانبندی اولویتی حتما قحطی زدگی دارند و غیر عادلانه هستند.

جدول ۹-۳ : مشخصات سیاستهای زمانبندی مختلف

| | FCFS | RR | SPN | SRT | HRRN | FB |
|---|---|---|---|---|---|---|
| تابع انتخاب | max[w] | ثابت | min[s] | min[s-e] | $\max(\frac{w+s}{s})$ | (به متن مراجعه شود) |
| 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 | تأکید نشده است | اگر برهه زمانی خیلی کوچک باشد کم می‌شود | زیاد | زیاد | زیاد | تأکید نشده است |
| زمان پاسخ | می‌تواند زیادباشد، به‌خصوص اگر واریانس زمانهای اجرا خیلی بزرگ باشد | برای فرایندهای کوتاه، زمان پاسخ خوبی ارائه می‌کند | برای فرایندهای کوتاه زمان پاسخ خوبی ارائه می‌کند | زمان پاسخ خوبی ارائه می‌کند | زمان پاسخ خوبی ارائه می‌کند | تأکید نشده است |
| سربار | حداقل | کم | می‌تواند زیاد باشد | می‌تواند زیاد باشد | می‌تواند زیاد باشد | می‌تواند زیاد باشد |
| تأثیر بر روی فرایندها | به‌فرایندهای کوتاه صدمه‌می‌زند، به فرایندهای درتنگنای ورودی/خروجی صدمه می‌زند | عملکرد عادلانه | به فرایندهای طولانی صدمه می‌زند | به فرایندهای طولانی صدمه می‌زند | توازن مناسب | می‌تواند به نفع فرایندهای در تنگنای ورودی/خروجی 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

**ج - زمان بند میان مدت یا مبادله گر** : قسمتی از سیستم عامل است که وظیفه دارد هنگام نیاز به آزادسازی حافظه اصلی قسمتس از فرآیندهای موجود در حافظه اصلی را موقتا" از حافظه ای خارج کند که به این کار مبادله به خارج گفتند می شود. و با مهیا بودن حافظه اصلی مجددا" فرآیند ها را به حافظه اصلی بازگرداند که به این کار مبادله به داخل گفته می شود.

# فصل چهارم: مدیریت حافظه (Memory Management)

1) **مدیر حافظه(Memory Manager)**:قسمتی از سیستم عامل است که از فضاهای پر و خالی اصلی اطلاع دارد. او این اطلاعات را درون جدولهای مدیریت حافظه نگهداری می‌نماید. برای اجرا شدن یک برنامه، آن را از حافظه جانبی به حافظه کپی میکند (برنامه تبدیل به فرآیند می‌شود). در این حالت می‌گویند به فرآیند حافظه تخصیص داده شده است. در پایان اجرای فرآیند نیز حافظه تخصیص داده شده به فرآیند را آزاد میکند.

2) **آدرس:** محل قرارگیری هر آیتم در حافظه، توسط آدرس آن آیتم مشخص میشود. این آیتم می‌تواند یک متغیر، ثابت، دستور یا ... باشد. در حالت کلی دو نوع آدرس وجود دارد:

**الف- آدرس منطقی یا مجازی:** آدرس منطقی محل یک آیتم(متغیر) در برنامه را مشخص میکند. این آدرس‌ها هنگام تولید فایل اجرایی برنامه توسط مترجم (Compiler) تولید می‌شوند.

**ب- آدرس فیزیکی یا واقعی:** هنگامی که قرار است یک برنامه اجرا شود، از حافظه جانبی به حافظه اصلی منتقل می‌شود. سپس برای هر آیتم(متغیر) یک محل در حافظه اصلی در نظر گرفته می‌شود. بدیهی است که این محل در حافظه اصلی دارای آدرس است. به آدرسی که محل یک آیتم (متغیر) در حافظه ی اصلی هتگام اجرا را مشخص میکند، آدرس فیزیکی گفته می‌شود.

**ج- نگاشت یا ترجمه آدرس:** هنگامی که پردازنده (CPU) یک دستور فرآیند (برنامه در حال اجرا) را واکشی کرده تا آن را اجرا نماید، با آدرس‌های منطقی مواجه خواهد شد (همانطور که گفته شده آدرس آیتم‌ها درون برنامه، آدرس منطقی است)، در حالیکه هنگام اجرای فرآیند باید عملیات مورد نظر بر روی آیتم‌های متناظر در حافظه اصلی صورت گیرد. پس پردازنده نیاز به آدرس فیزیکی آیتم‌ها دارد و باید آدرس‌های منطقی به آدرس فیزیکی تبدیل شود. به تبدیل آدرس منطقی به آدرس فیزیکی نگاشت آدرس یا تبدیل آدرس می گویند. بمنظور افزایش سرعت اجرا، این عمل معمولا بصورت سخت افزاری صورت می‌گیرد.

3) **نیازهای مدیریت حافظه:** یک سیستم مدیریت حافظه باید نیازهای ذیل را برآورده سازد.

**الف- جابجایی:** همانطور که میدانیم تغییر محل فرآیند در حافظه اصلی باعث تغییر آدرس فیزیکی المان (متغیر)های درون فرآیند می‌شود در حالیکه آدرس‌های منطقی تغییری نخواهد کرد. از طرف دیگر با توجه به مطالب فصل گذشته گاهی اوقات سیستم‌عامل ناچار است محل فرآیند در حافظه اصلی را تغییر دهد

(مثلا فرآیندی که معلق شده و از حافظه اصلی خارج شده است، پس از بازگردانی به حافظه اصلی در محل دیگری از حافظه اصلی قرار داده شود). جابجایی یعنی اینکه آدرس‌های منطقی مستقل از آدرس فیزیکی باشند و سیستم‌عامل بتواند آزادانه یک فرآیند را به هر نقطه از حافظه اصلی تغییر مکان دهد، بدون اینکه در اجرای فرآیند مشکلی ایجاد گردد.

**ب- حفاظت:** ازآنجا که در سیستم‌های چندبرنامه‌ای همواره چندین فرآیند در حافظه اصلی وجود دارد و بعلاوه همواره سیستم‌عامل نیز در حافظه اصلی است، باید هر فرآیند فقط به محدوده خودش در حافظه اصلی دسترسی داشته باشد و نتواند به محدوده سایر فرآیندها در حافظه اصلی دسترسی پیدا کند. بویژه باید محدوده سیستم‌عامل در حافظه اصلی در مقابل کلیه فرآیندهای دیگر محافظت گردد.

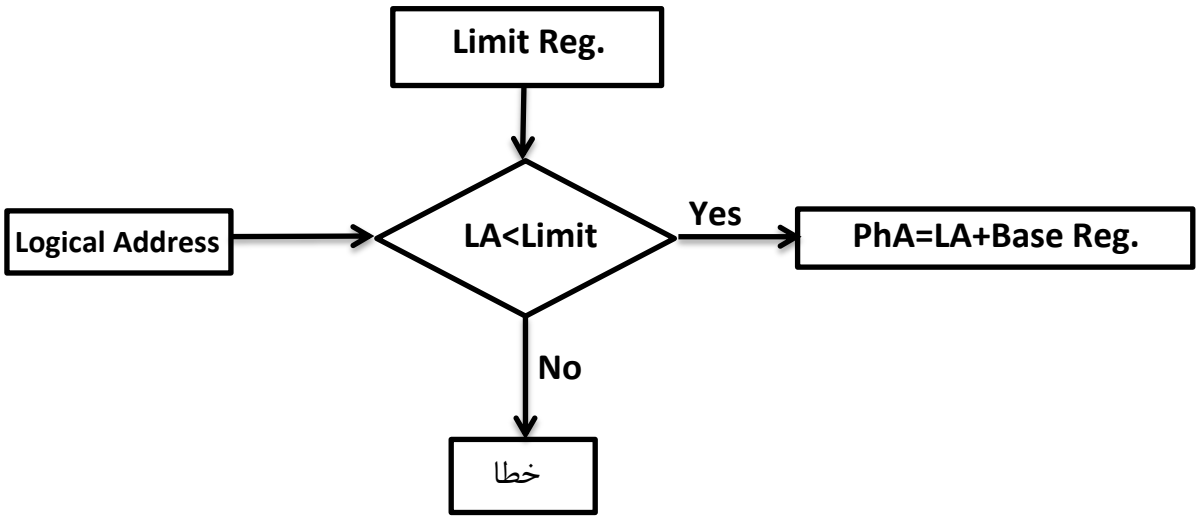
**ب- اشتراک:** در شرایطی که چندین فرآیند نیاز به دسترسی به آیتم (داده، متغیر) یکسانی داشته باشند و یا بخواهند دستورات یکسانی را اجرا نمایند، بهتر است که که این داده یا دستورات بین فرآیندها به اشتراک گذاشته شود. بنابراین سیستم مدیریت حافظه باید مکانیزم‌هایی برای به اشتراک گذاری آیتم‌ها بین فرآیندها داشته باشد و از این طریق نحوه دسترسی به داده‌ها یا دستورات مشترک را کنترل نماید.

**4) تکنیک های مدیریت حافظه:** هر تکنیک مدیریت حافظه شامل دو بخش است. بخش اول روشی است که طبق آن به فرآیند در حافظه اصلی، حافظه تخصیص داده می‌شود که به آن *روش تخصیص* گفته می‌شود. بخش دوم روشی است که طبق آن آدرس منطقی به آدرس فیزیکی تبدیل می‌شود که به آن *روش نگاشت* گفته می‌شود. بدیهی است که روش نگاشت وابستگی مستقیم به روش تخصیص دارد.

**4-1- تکنیک های مدیریت حافظه بصورت هم جوار یا پیوسته:** در تکنیک های این خانواده،فضایی که به یک فرآیند در حافظه اصلی داده میشود بصورت پشت سر هم و یکپارچه است و نمی توان فرآیند را در حافظه توزیع کرد. خانواده ی تکنیک های هم جوار روش های تخصیص متفاوتی دارد ولی در همه ی آنها از یک روش برای تبدیل آدرس منطقی به فیزیکی استفاده می شود.

**4-1-1-روش تبدیل آدرس در تکنیک های هم جوار:** در این روش در پردازنده یک ثبات پایه( Base Register) و یک ثبات حد(Limit Register) در نظر گرفته میشود که به ترتیب شروع و طول فرآیند در حافظه ی اصلی را نگهداری میکنند. برای تبدیل آدرس، ابتدا آدرس منطقی با ثبات Limit مقایسه شده تا از دسترسی غیرمجاز جلوگیری شود. اگر آدرس منطقی از ثبات Limit کوچکتر باشد دسترسی مجاز بوده و آدرس

فیزیکی از حاصلجمع آدرس منطقی با مقدار ثبات Base تولید میشود. (LA= Logical Address آدرس منطقی، PhA=Physical Address آدرس فیزیکی)



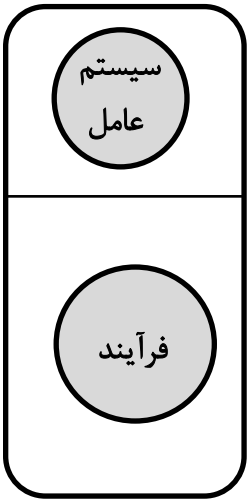

**4-1-2-روش های تخصیص در تکنیک هم جوار:**

**الف)تخصیص تک قسمتی Single Partition memory management:**در این روش تخصیص، یک قسمت از حافظه برای سیستم عامل در نظر گرفته شده و قسمت دیگر تنها در اختیار یک فرآیند قرار میگیرند. بعبارت دیگر علاوه بر سیستم‌عامل فقط یک فرآیند (برنامه) در حافظه اصلی قرار می‌گیرد. و سه مشکل دارد:

-استفاده ی غیربهینه از حافظه: اگر یک فرآیند نتواند کل حافظه اصلی را اشغال نماید، این باقیمانده فضا بدون استفاده باقی می‌ماند.

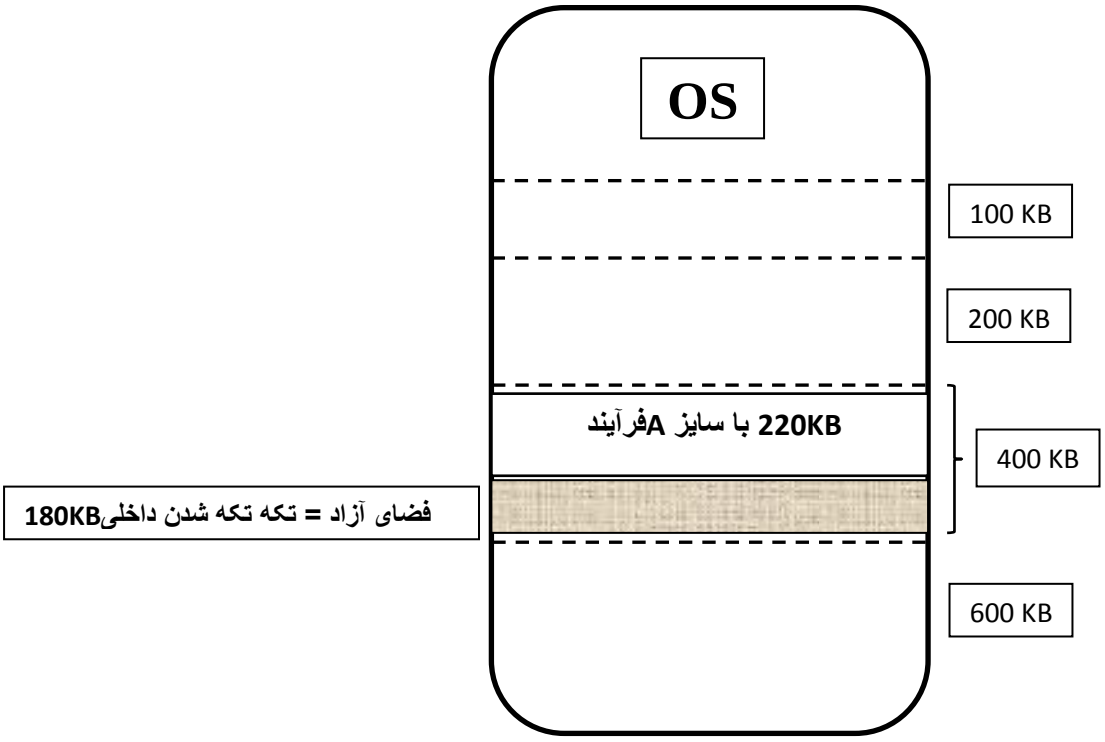
-چند برنامگی غیرممکن است: چون فقط یک فرآیند در حافظه اصلی قرار می‌گیرد، چندبرنامگی در این حالت وجود ندارد.

-کارآیی پایین پردازنده و وسایل ورودی خروجی: چون تنها یک فرآیند در سیستم است (چندبرنامگی نداریم)، در اکثر اوقات پردازنده یا وسایل ورودی/خروجی بیکار هستند، پس کارایی پایین است.




حافظه اصلی (Ram)

**ب)تخصیص چندقسمتی-پارتیشن ایستا Static Partition memory management:**در این روش یک قسمت از حافظه برای سیستم عامل در نظر گرفته شده و بقیه حافظه به صورت منطقی به بخش‌هایی با اندازه های ثابت ولی متفاوت شکسته می‌شود؛ به هر بخش یک پارتیشن گفته می‌شود. سپس برای اجرای هر فرآیند، پارتیشنی به آن فرآیند اختصاص داده می‌شود که اندازه‌اش بزرگتر یا مساوی اندازه فرآیند باشد. نکته مهم این است که به هر فرآیند تنها یک پارتیشن داده می‌شود و در هر پارتیشن تنها یک فرآیند قرار می‌گیرد.



این روش دارای 3 مشکل زیر است:

- چون در هر پارتیشن فقط یک فرآیند قرار می‌گیرد، حداکثر تعداد فرآیندهای موجود درحافظه اصلی مساوی تعداد پارتیشن هاست.

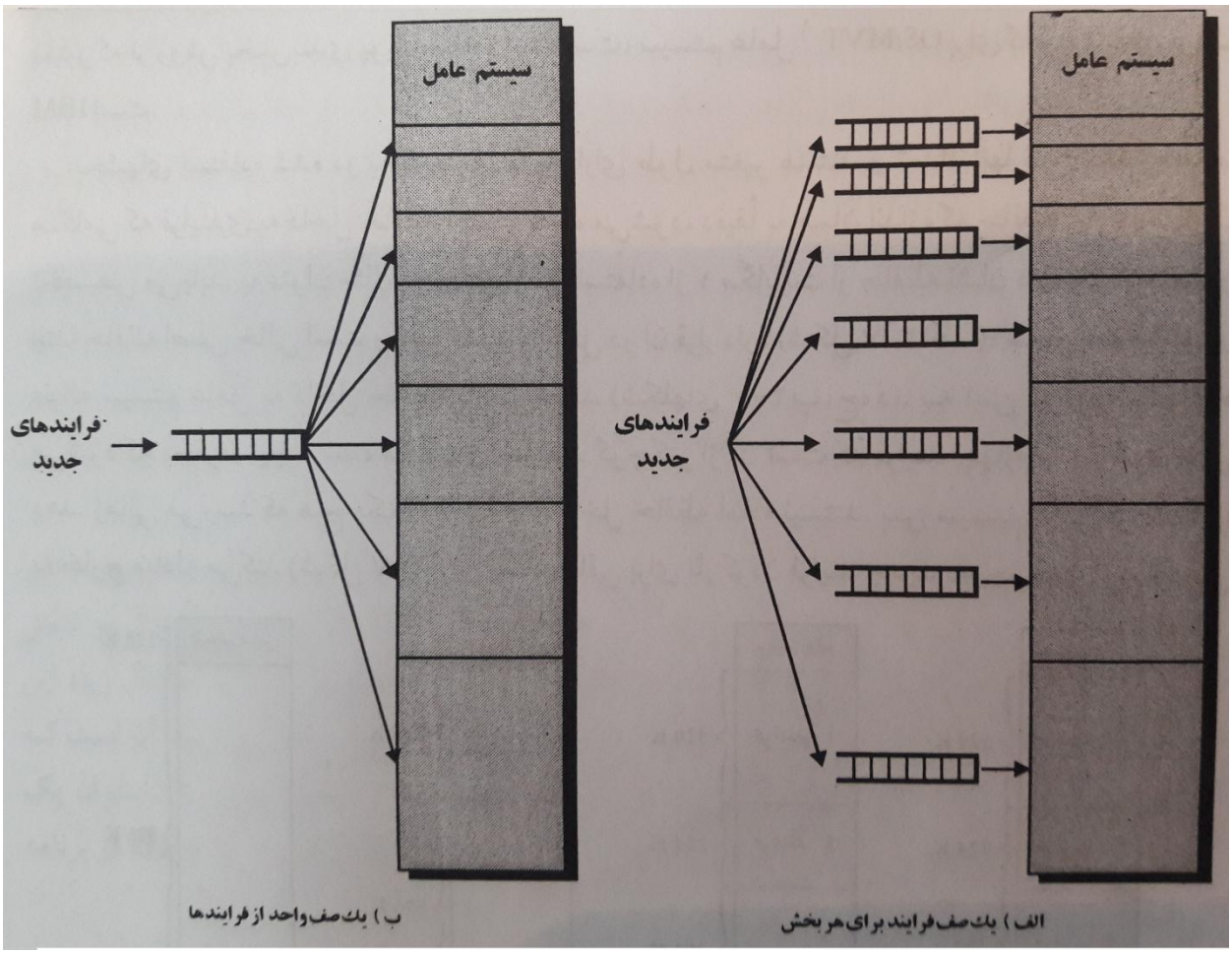
- چون به هر فرآیند تنها یک پارتیشن اختصاص داده می‌شود، بزرگتر از بزرگترین پارتیشن برنامه‌ای (فرآیندی) قابل اجرا نیست.

- **تکه تکه شدن داخلی حافظه یا Internal Fragmentation**:اگر پارتیشن اختصاص داده شده به فرآیند بزرگتر از اندازه ی فرآیند باشد، مقداری از فضای پارتیشن اختصاص داده شده خالی می ماند که به این اتفاق تکه تکه شدن داخلی می گویند. بنابراین ما مقداری حافظه آزاد ولی درعمل غیرقابل استفاده داریم، پس تکه‌تکه شدن داخلی باعث اتلاف حافظه می‌شود.

**تقسیم بندی بندی بخش‌ها به دو صورت است:**

**1- بخش‌های با اندازه مساوی:** در این روش حافظه به بخش‌های با اندازه مساوی شکسته می‌شود یا اندازه همه پارتیشن‌ها با هم مساوی است. ویژگی‌های این روش عبارتند از: سرعت در انتخاب پارتیشن، سربار کم، اگر فرآیندی در یک پارتیشن قابل قرارگیری نباشد آنگاه در هیچ پارتیشن دیگری نیز قابل قرار گیری نیست. تکه-تکه شدن داخلی افزایش می‌یابد.

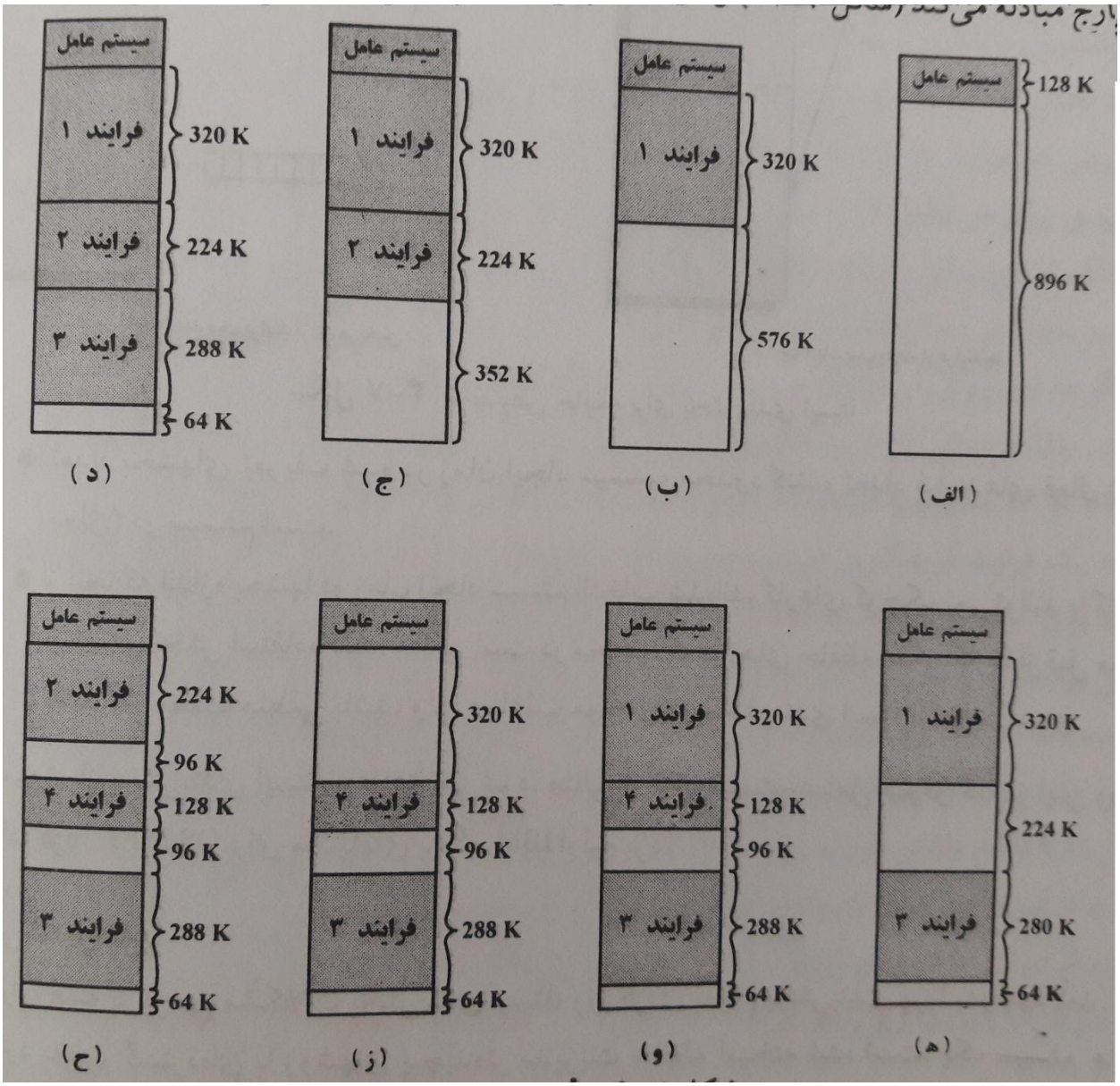
**2- بخش‌های با اندازه متفاوت:** در این روش حافظه به پارتیشن‌های با اندازه‌های متفاوت شکسته می‌شود. بنابراین مقدار تکه‌تکه شدن داخلی تا حدی کاهش می‌یابد. در این حالت برای قرار دادن فرآیندها در حافظه اصلی دو راه وجود دارد. **در راه اول** برای هر پارتیشن یک صف در نظر گرفته می‌شود و فرآیندها براساس اندازه-شان در صف کوچکترین پارتیشن مناسب (بزرگتر یا مساوی اندازه فرآیند) برای خود قرار داده می‌شوند تا هر گاه پارتیشن مربوطه آزاد گردد، فرآیند سر صف در پارتیشن قرار گیرد. مزیت این روش کاهش اتلاف بدلیل تکه‌تکه شدن داخلی است اما مشکل آن این است که ممکن است پارتیشن مناسب دیگری برای فرآیندی آزاد باشد ولی فرآیند منتظر ورود به کوچکترین پارتیشن مناسب بوده و اجرا نشود. **در راه دوم** برای همه پارتیشن‌ها یک صف واحد در نظر گرفته می‌شود و همه فرآیندها در این صف قرار می‌گیرند. به محض اینکه پارتیشنی آزاد شود اولین فرآیند سرصف که آن پارتیشن برایش مناسب باشد از صف برداشته شده و در آن پارتیشن قرار می-گیرد. اگر چندین پارتیشن بصورت همزمان آزاد باشند کوچکترین پارتیشن مناسب انتخاب می‌شود.




الف ) یک صف فرایند برای هر بخش

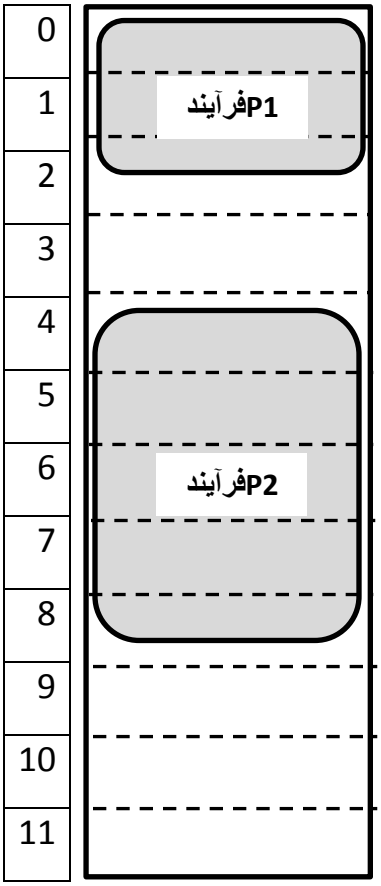
ب ) یک صف واحد از فرایندها

**ج)تخصیص چندقسمتی- پارتیشن پویا Dynamic Partition memory management:**در این روش یک قسمت از حافظه برای سیستم عامل در نظر گرفته شده و بقیه ی حافظه آزاد میباشد. هنگام اجرای یک فرآیند یک پارتیشن دقیقا به اندازه همان فرآیند ساخته شده و فرآیند در این پارتیشن قرار میگیرد. برای ساختن یک پارتیشن دقیقا یا متناسب با اندازه فرآیند، دو روش زیر وجود دارد:




**1- مدیریت حافظه با نگاشت بیتی:** در این روش حافظه اصلی به تعدادی پارتیشن هم‌اندازه تقسیم می‌شود که معمولا اندازه این پارتیشنها کوچک است. سپس سیستم عامل یک بردار بیتی در نظر گرفته که هر خانه آن نشان دهنده پر یا خالی بودن پارتیشن متناظر با این خانه است. مثلا اگر این بیت برابر 1 باشد آن پارتیشن پر و اگر صفر باشد آن پارتیشن آزاد است. حال هرگاه فرآیندی میخواهد در حافظه اصلی قرار گیرد سیستم عامل

تعدادی از این پارتیشنهای آزاد پشت سر هم را انتخاب کرده و فرآیند را در آن قرار میدهد. بعنوان مثال در شکل زیر اندازه هر پارتیشن برابر 100KB ، اندازه فرآیند P1=250KB و اندازه فرآیند P2=450KB است.



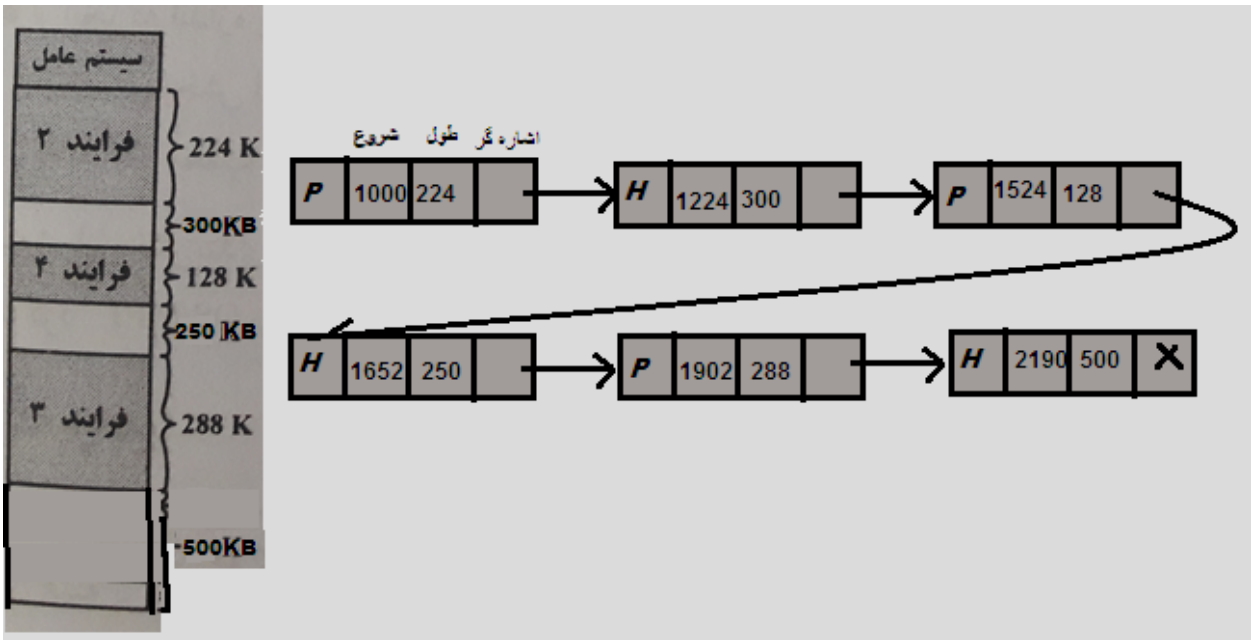

| شماره پارتیشن | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| وضعیت | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |

ماتریس بیتی متناظر با وضعیت بلاک‌های حافظه اصلی

هنگامی که سیستم‌عامل بخواهد فرآیند جدیدی در حافظه اصلی قرار دهد باید تعدادی پارتیشن آزاد پشت سر هم پیداکند به نحوی که آن فرآیند را بتوان در آن پارتیشن‌های پشت سرهم قرار داد. مثلا برای قرار دادن یک فرآیند 340 کیلوبایتی در شکل بالا نیاز به 4 پارتیشن آزاد پشت سرهم است. پس باید در ماتریس بیتی به همان تعداد بیت صفر پشت سرهم پیدا کند. این روش سه مشکل دارد: اول اینکه جستجوی ماتریس بیتی برای یافتن فضای مناسب برای فرآیند زمان را تلف میکند، دوما، هنگامیکه اندازه فرآیند مضرب صحیحی از اندازه پارتیشن نباشد مقداری از فضای آخرین پارتیشن اختصاص داده شده به فرایند خالی می‌ماند و این باعث تکه‌تکه شدن داخلی حافظه و در نتیجه اتلاف حافظه خواهد شد. سوما، برای ذخیره سازی ماتریس بیتی مقداری حافظه تلف می‌شود که برای حافظه‌های اصلی بزرگ فضای لازم برای ذخیره سازی ماتریس بیتی قابل توجه خواهد بود.

**نکته مهم اینکه:** هرچه اندازه پارتیشن‌ها بزرگتر باشد میزان تکه‌تکه شدن خارجی افزایش یافته و اندازه ماتریس بیتی کاهش می‌یابد، هرچه اندازه پارتیشن‌ها کوچکتر باشد میزان تکه‌تکه شدن خارجی کاهش یافته و اندازه ماتریس بیتی افزایش یافته و زمان جستجو در آن بیشتر می‌شود.

**2- مدیریت حافظه با لیست پیوندی:** در این روش کل فضای حافظه شامل قسمتهای تخصیص یافته و آزاد بصورت یک لیست پیوندی نمایش داده می‌شود. هر گره این لیست نشان دهنده یک قطعه از حافظه است که میتواند آزاد یا اشغال باشد. به قطعه‌های آزاد حفره گفته می‌شود. اولین فیلد در هر گره لیست نشان دهنده این است که این گره متناظر با یک فرآیند یا متناظر با یک حفره است، اگر مقدار H (Hole) نشان دهنده حفره و مقدار P (Process) نشان دهنده فرآیند است.




مطابق با شکل فوق حفره‌های خالی بدون هیچگونه نظمی در نقاط مختلف حافظه پراکنده شده اند. بنابراین هنگامی که سیستم عامل میخواهد فرآیند جدیدی را در حافظه اصلی قرار دهد باید در این لیست جستجو نموده تا حفره مناسب برای فرآیند را پیدا کند. برای انتخاب حفره خالی برای قرار دادن فرآیند از یکی از این 4 الگوریتم استفاده می شود: (فرض می‌کنیم بخواهیم فرآیندی با اندازه 220KB را در حافظه قرار دهیم)

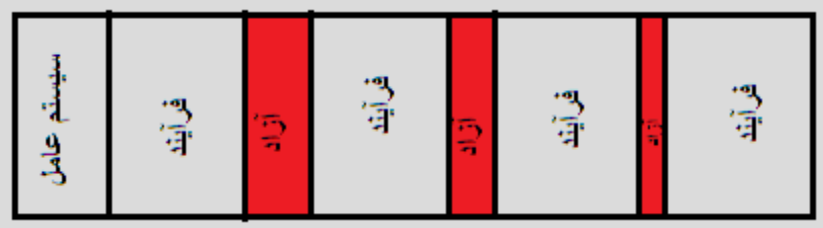
**1)First Fit:** مدیر حافظه لیست را از ابتدا بررسی کرده و فرآیند را در اولین حفره خالی مناسب قرار می‌دهد. در مثال ما، فرآیند در حفره 300KB قرار می‌گیرد. در این روش تراکم فرآیندها در ابتدای حافظه بیشتر است.

**Best Fit(2**: مدیر حافظه کل لیست را بررسی کرده و فرآیند را در کوچکترین حفره مناسب قرار میدهد. در مثال ما، فرآیند در حفره 250KB قرار می‌گیرد.

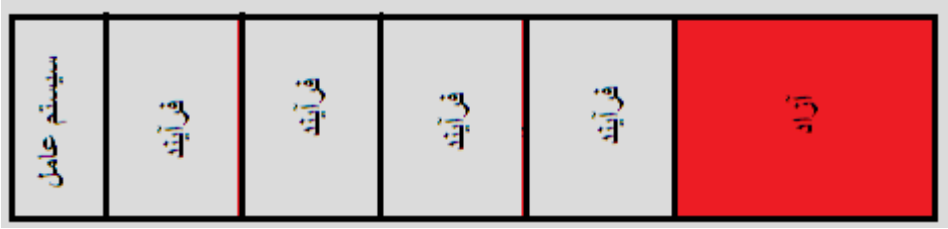
**Worst Fit(3**: مدیر حافظه کل لیست را بررسی کرده و فرآیند را در بزرگترین حفره مناسب قرار میدهد. در مثال ما، فرآیند در حفره 500KB قرار می‌گیرد.

**Next Fit(4**: این الگوریتم، همانند First Fit است ولی جستجوی لیست برای یافتن حفره‌ی خالی را از جایی آغاز میکند که دفعه ی قبل حفره‌ی خالی یافته بود.

**تکه تکه شدن خارجی:** در روش پارتیشن بندی پویا در اثر ورود و خروج متناوب فرایندها باریکه هایی از حافظه آزاد در حافظه اصلی ایجاد می‌شود. به این رخداد تکه تکه شدن خارجی میگویند و چون این فضاها غالبا کوچک هستند و قابل تخصیص به فرآیندهای دیگری نیستند باعث اتلاف حافظه می‌شوند.




**فشردگی یا compaction** : برای برطرف کردن تکه‌تکه شدن خارجی میتوان فرایند ها را در حافظه جابه جا کرده (شیفت داده ) و فضاهای خالی را در کنار هم قرار داد و فضاهای خالی بزرگتر ایجاد کرد که به این کار فشردگی گویند که عملی زمان بر است.




ویژگی‌های الگوریتم‌های ذکر شده:

1- الگوریتم Best Fit دارای این معایب است: - اتلاف زمان جهت جستجوی کل لیست و یافتن کوچکترین حفره، - تشدید تکه تکه شدن خارجی و انجام فشردگی به دفعات زیاد.

2- الگوریتم worst Fit دارای این معایب است: - اتلاف زمان جهت جستجوی کل لیست و یافتن بزرگترین حفره، - شکستن سریع حفره‌های بزرگ و مشکل در پذیرش فرآیندهای بزرگ

3- الگوریتم First Fit دارای این عیب است: هر بار جستجو را از ابتدای لیست آغاز کرده و زمان را تلف می‌کند.

4- الگوریتم Next Fit دارای این ویژگی‌ها است: - عدم جستجو برای کوچکترین و بزرگترین - عدم جستجو از ابتدای لیست.

**4-2- تکنیک های مدیریت حافظه بصورت غیر هم جوار یا پیوسته:** در تکنیک های این خانواده میتوان فرآیند را در جاهای مختلف حافظه توزیع نمود و عبارتند از :

**4-2-1 - مدیریت حافظه به روش صفحه بندی (Page memory management):** در این روش حافظه اصلی توسط سیستم عامل به قسمت هایی هم اندازه (اندازه ها ثابت و یکسان) به نام بلاک تقسیم می شود. سپس برنامه ای که قرار است اجرا شود به قسمت هایی هم اندازه با بلاک ها به نام صفحه تقسیم میشود.در نهایت میتوان هرصفحه فرآیندرا در هر بلاک آزاد قرار داد.

**الف- روش تبدیل ادرس منطقی به فیزیکی در صفحه بندی :**

برای تبدیل آدرس، آدرس منطقی به دو قسمت شماره صفحه و فاصله از شروع صفحه تبدیل می شود.سپس شماره صفحه به جدول صفحه اندیس می شود تا مشخص شود این صفحه در کدام بلاک قرار گرفته است.در نهایت با کنار هم قرار دادن شماره بلاک و فاصله از ابتدای صفحه آدرس فیزیکی بدست می آید.(فاصله از ابتدای صفحه با فاصله از ابتدای بلاک یکسان است.)

جدول صفحه :هربرنامه جدول صفحه ی اختصاصی خود را دارد. جدول صفحه جدولی است که مشخص می کند هر صفحه برنامه در کدام بلاک قرار گرفته است.در این جدول اندیس شماره صفحه و محتوای داخلی هر اندیس شماره بلاک است.

**ب- معایب صفحه بندی:**

**1- اتلاف حافظه:**

**a) بدلیل تکه تکه شدن داخلی :** چون همیشه اندازه ی برنامه ها مضرب صحیح از اندازه ی صفحه نمی باشد مقداری از فضای آخرین صفحه ی اختصاص داده شده به برنامه بدون استفاده باقی می ماند که این فضا به طور متوسط نصف یک صفحه یا $\frac{P}{2}$ می باشد.

**b) بدلیل حجم جدول صفحه :** هر برنامه مقداری از حافظه را بابت فضای جدول صفحه ی خود اتلاف می کند که اگر اندازه ی برنامه ها به طور متوسط s بایت باشد و اندازه هرخانه جدول صفحه e بایت باشد حجم جدول صفحه برابر است با : $\frac{S}{P} * e$ که $\frac{S}{P}$ تعداد خانه های جدول صفحه = تعداد صفحه‌های فرآیند است.

**c) کل اتلاف یک برنامه :**در نهایت هر برنامه به اندازه $\frac{S}{P} * e + \frac{P}{2}$ از حافظه را تلف می کند.

**2- کاهش سرعت کار با حافظه:** چون برای دسترسی به یک متغیر باید ابتدا به جدول صفحه مراجعه شود تا آدرس منطقی به آدرس فیزیکی تبدیل شود و سپس به خود متغیر مراجعه شود سپس برای دسترسی به هر متغیر دو مرتبه دسترسی به حافظه انجام شود سپس پس از مراجعات به حافظه دو برابر شده و سرعت حداقل نصف می گردد.

**ج) تعیین اندازه صفحه به روش در روش صفحه بندی:**

*- اگر اندازه صفحه یا P کوچک باشد : 1.کاهش اتلاف تکه تکه شدن داخلی 2.افزایش اتلاف حجم جدول صفحه.

*- اگر اندازه صفحه یا P بزرگ باشد : 1.افزایش اتلاف بدلیل تکه تکه شدن داخلی 2. کاهش اتلاف به دلیل حجم جدول صفحه .

*- اندازه ایده ال صفحه $P = \sqrt{2se}$ $f'(P) = 0$ $f(P) = \frac{S}{P} * e + \frac{P}{2}$

**4-2-2- مدیریت حافظه به روش قطعه بندی (Segmentation) :** در این روش برنامه توسط کامپایلر یا برنامه نویس به قسمت هایی با اندازه های متفاوت و محتویات متفاوت شکسته می شود که هر قسمت یک segmentمیگویند. سیستم عامل یک قسمت از حافظه اصلی را برای خود بر میدارد و جهت اجرای برنامه برای هر قطعه یک قسمت مجزا می سازد که به آن پارتیشن گفته میشود و هر سگمنت را در یک پارتیشن قرار می‌دهد ولی الزامی نیست که این قسمت‌ها پشت سر هم باشند.

**تبدیل ادرس منطقی به فیزیکی در قطعه بندی:** برای تبدیل ابتدا ادرس منطقی به فیزیکی، ابتدا آدرس منطقی به دو بخش شماره قطعه و فاصله از شروع قطعه تقسیم می‌شود. سپس شماره قطعه به جدول قطعه اندیس می-شود تا محل شروع قطعه و طول قطعه مشخص گردد. در مرحله‌ی بعد فاصله از شروع قطعه با طول قطعه مقایسه می‌شود تا از دسترسی غیر مجاز جلوگیری شود در نهایت حاصل جمع فاصله از شروع قطعه و آدرس شروع قطعه ، آدرس فیزیکی را تولید می‌کند.

**جدول قطعه:** هر برنامه جدول قطعه خود را دارد و جدولی است که برای هر برنامه شروع و طول قطعه‌های هر برنامه را نشان می‌دهد.

**معایب قطعه بندی:**

**الف) اتلاف حافظه:**

**1- تکه تکه شدن خارجی :**در روش قطعه بندی باریکه‌های فضای آزاد حافظه که میان قسمت‌ها ایجاد می‌شود و به آن تکه تکه شدن خارجی می‌گوییم باعث اتلاف حافظه می‌شود .2- حجم جدول قطعه:هر برنامه مقداری از حافظه را بابت فضای مورد نیاز برای جدول قطعه خود اشغال یا تلف می‌کند.

**ب) کاهش سرعت کار با حافظه:** برای دسترسی به هر متغیر باید یکبار برای تبدیل آدرس منطقی به فیزیکی به حافظه اصلی مراجعه کرد و بار دیگر پس از بدست آمدن آدرس فیزیکی به خود متغیر مراجعه کرد، در نتیجه برای هر دسترسی به متغیر مراجعات به حافظه دو برابر شده و سرعت حداقل نصف می‌گردد.

**4-2-3- مقایسه صفحه بندی و قطعه بندی:** 1- هردو روش تخصیص غیر هم جوار هستند. 2- هردو روش باعث اتلاف حافظه می‌شوند. 3- هردو سرعت کار با حافظه را کاهش می‌دهند. 4- در صفحه‌بندی اندازه قسمتهای حافظه یکسان است ولی در قطعه بندی یکسان نیست. 5- صفحه بندی تکه تکه شدن داخلی دارد و قطعه بندی تکه تکه شدن خارجی دارد. 6- صفحه بندی از دید برنامه نویس شفاف است ولی قطعه بندی شفاف نیست و برنامه نویس از اندازه قطعات آگاه است. 7- در صفحه بندی تبدیل آدرس با جانشینی است ولی در قطعه بندی با مقایسه و جمع.

## فصل پنجم: حافظه مجازی (Virtual memory)

**1- حافظه مجازی:** با توجه به اینکه در برخی موارد جهت اجرای یک برنامه، الزامی به انتقال کل برنامه به حافظه اصلی نیست. مثلا: قسمت‌هایی از یک برنامه که برای مدیریت و برخورد با خطاها نوشته میشوند و چون خطاها به‌ندرت اتفاق می‌افتند لذا این قسمت‌ها هم به‌ندرت اجرا می‌شوند؛ برنامه‌های بزرگ دارای منوهای متعدد که برای هر منو قسمتی از برنامه اجرا می‌شود. لذا ***حافظه مجازی*** یعنی به کارگیری حافظه پشتیبان ( هارد) در کنار حافظه اصلی به این منظور که برای اجرای یک برنامه، تمامی برنامه وارد حافظه اصلی نشود و فقط قسمت-هایی وارد شود که جهت اجرا لازم است.

**توجه مهم:** بمنظور تسریع در عملیات تبدیل آدرس منطقی به آدرس فیزیکی در سیستم‌هایی که از صفحه-بندی استفاده می‌کنند، عمل تبدیل آدرس بصورت سخت افزاری و درون پردازنده توسط واحدی بنام MMU (Memoey Management Unit) صورت می‌گیرد. این واحد وظیفه دارد آدرس منطقی (مجازی) تولید شده توسط برنامه را به آدرس فیزیکی (واقعی) تبدیل نماید.

شایان ذکر است که قبل از ایجاد تکنیک حافظه مجازی، تکنیک دیگری بنام برهم‌نهی یا Overlay ایجاد شده بود. در این تکنیک برنامه اجرایی توسط خود برنامه نویس به بخشهایی بنام overlay تقسیم بندی می‌شد که این بخشها از عدد صفر شماره گذاری می‌شدند (overlay 0, ovarlay 1,..., ovarlay n). سپس برای اجرای برنامه، اولین overlay یعنی overlay 0 وارد حافظه اصلی می‌شد و اجرای آن آغاز می‌شد. در انتهای هر overlay دستوراتی توسط برنامه نویس گذاشته شده بود که ovarlay 1 را به درون حافظه اصلی فراخوانی می‌کرد و همین عملیات تکرار می‌شد (هر overlay، overlay بعدی خود را فرخوانی می‌کرد). مشکل اصلی

این تکنیک عدم شفافیت برای برنامه نویس بود به این معنی که برنامه نویس را مستقیما در گیر تقسیم بندی برنامه به overlay ها و جایگزینی آنها با یکدیگر در حافظه اصلی می‌نمود.

**2- مزایای حافظه مجازی:**

1-امکان اجرای برنامه‌های بزرگتر از حافظه اصلی: چون لازم نیست کل برنامه جهت اجرا وارد حافظه اصلی شود و فقط در هر لحظه قسمت‌هایی از برنامه در حافظه اصلی است، میتوان برنامه‌هایی را اجرا کرد که سایزی بزرگتر از حافظه اصلی دارند.

2- افزایش تعداد فرآیندهای درون حافظه اصلی به صورت هم‌زمان: چون فقط بخشی از هربرنامه وارد حافظه اصلی می-شود، بنابراین می‌توان تعداد بیشتری برنامه را بصورت همزمان وارد حافظه اصلی نمود. این مساله بنوبه خود باعث افزایش درجه چندبرنامگی و کاهش بیکاری منابع سیستم و افزایش کارایی خواهد شد.

3- عدم انجام ورودی خروجی بیهوده: چون قسمت‌هایی از برنامه که برای اجرا لازم نیست وارد حافظه اصلی نمی‌شود، عملیات ورودی/خروجی بیهوده نیز صورت نمی‌گیرد.

**3- محلیت ارجاع:** اصل محلیت ارجاع تضمین کننده درستی و کاربردی بودن تکنیک حافظه مجازی است. بر اساس این اصل یک برنامه در حال اجرا در فاصله‌های زمانی کوتاه به داده‌ها و دستورات محدود مراجعه دارد یا ارجاعات یک برنامه به داده‌ها و دستوراتش در یک بازه کوتاه زمانی به داده‌های محدودی انجام خواهد شد.

**4- روش‌های پیاده سازی حافظه مجازی**

1- حافظه مجازی به روش صفحه بندی یا صفحه بندی درخواستی (**Demanded page memory managenet**)

2- حافظه مجازی به روش قطعه بندی یا قطعه بندی درخواستی (**Demanded segmentation**)

3- ترکیب قطعه‌بندی با صفحه‌بندی درخواستی ( قطعات صفحه‌بندی شده)

**5- حافظه مجازی به روش صفحه‌بندی یا صفحه‌بندی درخواستی:**

در این روش همانند صفحه‌بندی معمولی حافظه اصلی به تعدادی بلاک و برنامه‌ی جهت اجرا به تعدادی صفحه هم اندازه با بلاک تقسیم می‌شود، ولی همه‌ی صفحه‌های برنامه وارد حافظه اصلی نمی‌شود و فقط آن صفحه-هایی وارد می‌شود که جهت اجرا لازم هستند.

در اینجا بررسی و پاسخ به دو سئوال ضروری است. اولا اینکه سیستم عامل چگونه متوجه می‌شود که کدام صفحه از برنامه برای اجرا لازم است و باید وارد حافظه اصلی شود؟ پاسخ: وقتی آدرس منطقی دریافتی از برنامه تجزیه شده و شماره صفحه ، offset آن آدرس محاسبه می‌شود، شماره صفحه مشخص می‌کند که باید کدام صفحه وارد حافظه اصلی بشود. دوما، با توجه به اینکه همه صفحه‌های فرآیند در حافظه اصلی قرار ندارند چگونه میتوان فهمید که آیا صفحه مورد نیاز برای اجرای برنامه در این لحظه در حافظه اصلی قرار دارد یا خیر؟ پاسخ: در اینجا باید سیستم عامل اجزاء دیگری به جدول صفحه فرآیند اضافه نماید که در ذیل توضیح داده شده اند.

**الف) بیت اعتبار/عدم اعتبار یا حاضر/غایب (Valid bit یا اختصارا V):** در جدول صفحه برای هر صفحه یک بیت حاضر/غایب در نظر گرفته می‌شود، که اگر v برابر "یک" باشد صفحه در حافظه اصلی است و اگر v برابر "صفر" باشد صفحه در حافظه اصلی نیست.

**ب) بیت مراجعه (Reference bit یا اختصارا R):** هنگامی که صفحه‌ای وارد حافظه اصلی می‌شود، مقدار این بیت برایش برابر "صفر" است. چنانچه مراجعه‌ی مجددی به صفحه در حافظه اصلی صورت گیرد مقدار این بیت برابر "یک" خواهد شد.

**ج) بیت تغییر (Modify bit یا اختصارا M):** هنگامی که صفحه‌ای وارد حافظه اصلی می‌شود، مقدار این بیت برایش برابر "صفر" است. با اولین مراجعه‌ی به صفحه در حافظه اصلی که باعث تغییر در محتوای صفحه گردد (عمل نوشتن در صفحه) مقدار این بیت برابر "یک" خواهد شد.

**تذکر مهم:** هنگامی که صفحه‌ای از حافظه اصلی خارج می‌شود، مقدار این سه بیت برایش برابر "صفر" خواهد شد. ساختار جدول صفحه در شکل زیر آمده است.

| Page number p_no | Block number b_no | Valid/Invalid (V) | Reference (R) | Modify (M) |
|---|---|---|---|---|
| 0 | | | | |
| 1 | | | | |
| 2 | | | | |
| . | | | | |
| . | | | | |
| . | | | | |

البته ممکن است فیلدهای دیگری نیز بر حسب نیاز به جدول اضافه شود که متعاقبا بحث خواهد شد.

**6- تبدیل آدرس مجازی به فیزیکی در روش صفحه‌بندی درخواستی:**

الف) واحد MMU آدرس مجازی تولید شده توسط برنامه را دریافت می‌کند.

ب) آدرس مجازی توسط MMU به دو قسمت شماره صفحه (p_no) و انحراف در صفحه (offset) تجزیه میشود.

ج) با استخراج شماره صفحه (p_no)، شماره صفحه به جدول صفحه اندیس شده و ردیف مربوط به این صفحه مشخص می‌گردد.

د) ابتدا بیت حضور (V) مربوط به صفحه بررسی می‌شود. اگر بیت V برابر "یک" باشد، یعنی صفحه در حافظه اصلی قرار دارد و شماره بلاکی که این صفحه در آن قرار دارد از جدول بازیابی می‌شود و کار تمام است. در غیر اینصورت مراحل ذیل ادامه می‌یابد.

ه) اگر بیت V برابر صفر باشد، یعنی صفحه در حافظه اصلی قرار نداشته و در حافظه جانبی بر روی دیسک است. پس باید ابتدا این صفحه از حافظه جانبی به حافظه اصلی آورده شود که اینکار باید توسط سیستم‌عامل صورت گیرد. لذا یک وقفه نقص صفحه رخداده که در نتیجه آن فرآیند درحال اجرا موقتا بلوکه شده و سیستم‌عامل اجرا می‌شود.

و) سیستم‌عامل به حافظه جانبی (دیسک) مراجعه کرده و صفحه مورد نیاز را به حافظه اصلی آورده و دریک بلاک قرار میدهد. همچنین ممکن است برای یافتن یک بلاک جایگزینی صفحه صورت گیرد.

**جایگزینی صفحه:** در حافظه‌ی مجازی به روش صفحه بندی به هر برنامه تعدادی بلاک اختصاص داده می‌شود هنگامی که این بلاک ها توسط صفحه‌های آن برنامه اشغال شد برای آوردن یک صفحه جدید باید یکی از صفحه‌های قبلی از حافظه حذف شود و صفحه جدید بجای صفحه قبلی در حافظه اصلی قرار گیرد؛ به این اتفاق جایگزینی صفحه گویند.

ز) جدول صفحه تصحیح گردد: بیت حضور صفحه مربوطه برابر "یک" شده و اگر جایگزینی صفحه صورت گرفته است، بیت حضور صفحه خارج شده برابر "صفر" شود.

ح) حال صفحه در حافظه اصلی قرار دارد و شماره بلاکی که این صفحه در آن قرار دارد از جدول بازیابی می‌شود و کار تمام است.

**رشته‌ی مراجعه:** به دنباله‌ی شماره صفحه‌هایی که یک برنامه به آن‌ها نیاز دارد رشته‌ی مراجعه می‌گویند.

**نقص صفحه یا فقدان صفحه:** هرگاه برنامه صفحه ای را بخواهد که در حافظه ی اصلی موجود نباشد نقص صفحه رخ داده است.

**7- الگوریتم های جایگزینی صفحه:**

**1) الگوریتم FIFO:** این الگوریتم صفحه ای را خارج می کند که زودتر از بقیه وارد حافظه ی اصلی شده است. برای پیاده سازی این الگوریتم دو راه وجود دارد:

**الف)** در نظر گرفتن یک فیلد ساعت در جدول صفحه برای ثبت زمان ورود صفحه به حافظه اصلی: در این روش برای هر صفحه یک فیلد ساعت در جدول صفحه در نظر گرفته می‌شود و لحظه ورود صفحه به حافظه اصلی ثبت می‌گردد. بدیهی است صفحه‌ای که کوچکترین زمان ورود را دارد صفحه‌ای است که زودتر از بقیه وارد حافظه اصلی شده است و برای خروج انتخاب می‌گردد.

**ب)** استفاده از یک صف است: در این روش سیستم‌عامل یک صف بصورت لیست پیوندی درست می‌کند که صفحه ها به ترتیب ورود در انتهای لیست پیوندی (صف) قرار گیرند. صفحه‌هایی که زودتر وارد شده‌اند در صف جلوتر هستند و صفحه‌ای که زودتر از همه وارد شده است در سر صف قرار دارد. بنابراین صفحه‌ای که در سر صف قرار دارد برای جایگزینی انتخاب می‌شود.

مزیت الگوریتم **FIFO** سادگی پیادسازی است ولی دو مشکل دارد: اولاً کورکورانه عمل میکند یعنی ممکن است صفحه های پرکاربرد را حذف کند، یعنی صفحه‌ای را از حافظه اصلی خارج کند که برنامه در آینده نزدیک به آن صفحه احتیاج دارد. ثانیاً ***ناسازگاری بلیدی*** دارد یعنی گاهی موارد امکان دارد با زیاد کردن تعداد بلاک های در اختیار یک برنامه تعداد نقص صفحه‌های برنامه افزایش یابد در حالی که منطقاً باید کاهش یابد.

| Page number p_no | Block number b_no | Valid/Invalid (V) | Modify (M) | Enterance Time |
|---|---|---|---|---|
| 0 | | | | |
| 1 | | | | |
| 2 | | | | |
| . | | | | |
| . | | | | |
| . | | | | |

**2) الگوریتم BO (Best option) یا Optimal یا جایگزینی بهینه:**

در این الگوریتم، با فرض اینکه دنباله مراجعات برنامه به صفحه‌هایش در آینده مشخص و در دسترس است، صفحه ای خارج می‌شود که تا اولین مراجعه ی بعدی به آن صفحه در آینده زمان بیشتری باقی مانده باشد. این الگوریتم بهترین کارایی و کمترین تعداد نقص صفحه را دارد ولی به دلیل نیاز به اطلاع از آینده قابل پیاده سازی عملی نیست و فقط یک معیار برای سنجش سایر الگوریتم هاست.

**3) الگوریتم کمترین استفاده در گذشته اخیر(LRU یا Least Recently Used) :**

این الگوریتم صفحه ای را خارج میکند که از آخرین مراجعه به آن در گذشته زمان بیشتری سپری شده باشد. بعبارت دیگر این صفحه گذشته اخیر را تقریبی از آینده قرار میدهد و فرض میکند صفحه‌ای که در گذشته اخیر مورد استفاده نبوده است، در آینده نزدیک هم مورد استفاده قرار نگیرد پس این صفحه را خارج می‌کند. این الگوریتم بهترین الگوریتم قابل پیاده سازی بوده و ناسازگاری بلیدی ندارد.

**برای پیاده سازی این الگوریتم دو راه وجود دارد:**

**راه اول** بکاربردن همان صفی است که در روش FIFO اشاره شده است. در اینجا چنانچه به صفحه‌ای مراجعه شود که خودش در حافظه اصلی قرار دارد، آنگاه شماره‌ی این صفحه به انتهای صف منتقل می‌شود. بنابراین صفحه‌ای که از آخرین مراجعه به آن زمان بیشتری سپری شده است در سر صف و صفحه‌ای که اخیرا مورد استفاده بوده است در ته صف قرار دارد. مشکل این روش این است که پیدا کردن یک صفحه در لیست، حذف آن و یا انتقال آن به انتهای صف به ازاء هر دسترسی به حافظه پیچیدگی زمانی و هرینه پیاده‌سازی بالایی دارد.
**راه دوم** در نظر گرفتن یک فیلد ساعت در جدول صفحه برای ثبت زمان آخرین استفاده از صفحه در حافظه اصلی است. در این روش برای هر صفحه یک فیلد ساعت در جدول صفحه در نظر گرفته می‌شود و لحظه ورود صفحه به حافظه اصلی در آن ثبت می‌گردد. بعد از آن نیز با هر بار استفاده از صفحه، این زمان بهنگام (update) می‌شود. بدیهی است صفحه‌ای که کوچکترین زمان را دارد صفحه‌ای است که از آخرین مراجعه به آن زمان بیشتری سپری شده و برای خروج انتخاب می‌گردد.

| Page number p_no | Block number b_no | Valid/Invalid (V) | Modify (M) | Enterance/Use Time |
|---|---|---|---|---|
| 0 | | | | |
| 1 | | | | |
| 2 | | | | |
| . | | | | |
| . | | | | |

**4) الگوریتم کمترین دفعات استفاده (LFU یا Least Frequently Used):**

این الگوریتم در جدول صفحه، برای هر صفحه یک شمارنده در نظر می‌گیرد. در ابتدای کار مقدار این فیلد شمارنده برابر صفر است و هنگامی که صفحه از حافظه جانبی به حافظه اصلی منتقل می‌شود نیز مقدار این فیلد شمارنده برابر صفر باقی می‌ماند. پس از آن با هر بار مراجعه مجدد به صفحه (استفاده مجدد از صفحه) هنگامی که صفحه در حافظه اصلی قرار دارد یک واحد به فیلد شمارنده آن صفحه اضافه میشود. بنابراین فیلد شمارنده یک صفحه نشان دهنده دفعات مراجعه به آن صفحه در حافظه اصلی است. در نهایت این الگوریتم صفحه‌ای را برای جایگزینی انتخاب می‌کند که کمترین مقدارشمارنده را دارد یعنی کمترین دفعات استفاده را داشته است. با خروج صفحه از حافظه اصلی مجددا مقدار شمارنده برابر صفر می‌شود.

این الگوریتم دو نقطه ضعف دارد: اولا برای پیدا کردن مقدار کوچکترین شمارنده (cnt) زمان را تلف میکند. ثانیا، گذشته دور در این الگوریتم تاثیرگذار است؛ یعنی چنانچه صفحه‌ای در گذشته دور بدفعات زیاد مورد استفاده قرار گرفته باشد و به این دلیل مقدار فیلد شمارنده‌اش زیاد شده باشد، حتی اگر در حال حاضر و یا در آینده مورد استفاده نباشد نیز برای جایگزینی انتخاب نمی‌شود.

| Page number p_no | Block number b_no | Valid/Invalid (V) | Modify (M) | Counter |
|---|---|---|---|---|
| 0 | | | | |
| 1 | | | | |
| 2 | | | | |
| . | | | | |
| . | | | | |
| . | | | | |

**5) الگوریتم بیشترین دفعات استفاده (MFU یا Meast Frequently Used):**

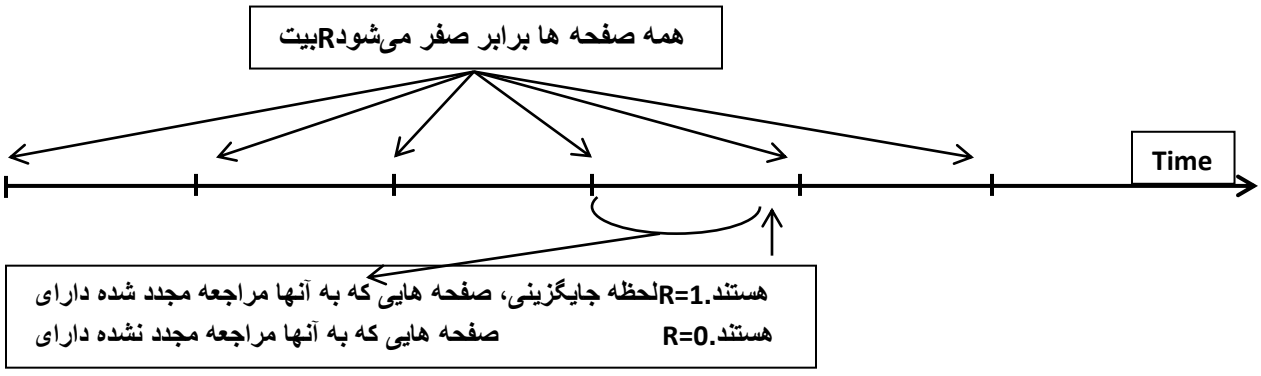
این الگوریتم نیز همانند الگوریتم LFU از فیلد شمارنده برای هر صفحه استفاده میکند و دقیقا همانند آن الگوریتم مقدار شمارنده تغییر می‌کند. ولی صفحه‌ای را برای جایگزینی انتخاب می‌کند که بیشترین مقدار شمارنده را دارد یعنی بیشترین دفعات استفاده را داشته است. با خروج صفحه از حافظه اصلی مجددا مقدار شمارنده برابر صفر می‌شود. نقطه ضعف این الگوریتم این است که برای یافتن بیشترین شمارنده (cnt) زمان را تلف میکند.

### 6) الگوریتم زیاد استفاده نشده (NFU یا Not Frequently Used):

برای جلوگیری از اتلاف زمان یافتن کمترین یا بیشترین مقدار شمارنده که در دو الگوریتم قبلی وجود داشت، در الگوریتم NFU سیستم عامل یک حد آستانه بنام a در نظر میگیرد که معمولا عدد کوچکی است. برای جایگزینی صفحه، از ابتدای جدول صفحه بررسی نموده و اولین صفحه‌ای که مقدار فیلد شمارنده‌اش از a کمتر باشد را برای جایگزینی انتخاب می‌کند. در واقع این الگوریتم یافتن مینیمم نسبی را به یافتن مینیمم مطلق ترجیح میدهد و بهمین دلیل از اتلاف زمان جلوگیری می‌نماید.

### 7) الگوریتم اخیراً استفاده نشده (NUR یا Not Used Recently ):

در این روش سیستم عامل با استفاده از بیت R موجود در جدول صفحه به این صورت عمل می‌کند: زمان به بازه های مساوی تقسیم میشود و در رأس این بازه ها بیت R همه صفحه‌ها "صفر" میشود. هنگام جایگزینی صفحه، نسبت به آخرین باری که بیت R همه صفحه ها "صفر" شده بود به بعضی از صفحه ها مراجعه ی مجدد شده که طبیعتاً بیت R آن صفحه‌ها برابر "یک" شده است؛ و به بعضی از صفحه‌ها هم مراجعه‌ی مجدد نشده است که طبیعتاً بیت R آن صفحه‌ها "صفر" باقی مانده است. صفحه‌هایی که بیت R آنها برابر "صفر" است، همان صفحه های اخیراً استفاده نشده هستند. بنابراین، سیستم عامل جدول صفحه را از ابتدا بررسی نموده و اولین صفحه که مقدار بیت R اش برابر "صفر" باشد را برای جایگزینی انتخاب میکند.




### 8) الگوریتم دومین شانس (Second chance):

این الگوریتم ترکیب الگوریتم FIFO و بیت R است. یعنی صفحه‌ها براساس زمان ورود به حافظه بررسی می-شوند و اگر بیت R صفحه‌ای برابر با "یک" باشد (یعنی یک صفحه پرکاربرد است) بیت R آن صفحه برابر با

"صفر" شده و به صفحه یک شانس مجدد داده می‌شود که در حافظه باقی بماند. با توجه به اینکه دو روش برای پیاده‌سازی الگوریتم FIFO وجود دارد، این الگوریتم نیز به دو صورت پیاده سازی می‌شود:

**روش اول-با استفاده از صف لیست پیوندی:** صف صفحه‌های موجود در حافظه اصلی (که بصورت لیست پیوندی است)از ابتدا بررسی می‌شود. اگر بیت R صفحه‌ای برابر با "یک" بود، این بیت برابر "صفر" شده و صفحه به انتهای صف منتقل می‌شود و به این صفحه یک شانس مجدد داده می‌شود که در حافظه باقی بماند. اگر بیت R صفحه‌ای برابر با "صفر" بود این صفحه برای جایگرینی انتخاب می‌شود. بنابراین، اولین صفحه‌ای که بیت R اش برابر صفر باشد برای جایگرینی انتخاب می‌شود.

**روش دوم- با استفاده از جدول و فیلد زمان ورود:** صفحه‌های موجود در حافظه اصلی به ترتیب صعودی زمان ورود بررسی شوند. اگر بیت R صفحه‌ای برابر با "یک" بود، این بیت برابر "صفر" شده و زمان ورود صفحه به "همین لحظه" بهنگام شده و به این صفحه یک شانس مجدد داده می‌شود که در حافظه باقی بماند و به سراغ صفحه ی بعدی براساس زمان ورود می‌رود. اگر بیت R صفحه‌ای برابر با "صفر" بود این صفحه برای جایگرینی انتخاب می‌شود. بنابراین، اولین صفحه بر اساس زمان ورود که بیت R اش برابر صفر باشد برای جایگرینی انتخاب می‌شود.

**نقطه ضعف این الگوریتم:** انتقال دادن صفحه‌ها به انتهای صف در روش اول ( که باید گره‌ها در لیست پیوندی جابجا شوند) و بهنگام سازی فیلد زمان ورود در روش دوم است.

| pno | bno | v | Enterence Time | R |
|---|---|---|---|---|
| 0 | | | | |
| 1 | | | | |
| 2 | | | | |
| . | | | | |
| . | | | | |
| . | | | | |

### 9) الگوریتم ساعت (Clock):

برای برطرف کردن مشکل الگوریتم "دومین شانس" الگوریتم ساعت ارائه شده است. این الگوریتم هم از ترکیب الگوریتم FIFO و بیت R استفاده می‌کند. و بصورت زیر پیاده سازی می‌شود:

**با استفاده از صف لیست پیوندی:** صف صفحه‌های موجود در حافظه اصلی (که بصورت لیست پیوندی است)از ابتدا بررسی می‌شود. اگر بیت R صفحه‌ای برابر با "یک" بود، این بیت برابر "صفر" شده و به این صفحه یک

شانس مجدد داده می‌شود که در حافظه باقی بماند. اگر بیت R صفحه‌ای برابر با "صفر" بود این صفحه برای جایگرینی انتخاب می‌شود و اشاره گر به گره بعدی در لیست پیوندی اشاره می‌کند. هنگامی که به انتهای لیست پیوندی برسیم، مجددا اشاره‌گر به ابتدای لیست آورده می‌شود.

**10) الگوریتم Not Recently Used یا NRU :**

ترکیبی از الگوریتم NUR و بیت M است، بعبارت دیگر بر اساس دو بیت مراجعه (R) و تغییر (M) صفحه‌ای را برای جایگزینی انتخاب می‌کند (یادآوری: در الگوریتم NRU، زمان به بازه های مساوی تقسیم میشود و در رأس این بازه ها بیت R همه صفحه‌ها "صفر" میشود. هنگام جایگزینی صفحه، نسبت به آخرین باری که بیت R همه صفحه ها "صفر" شده بود به بعضی از صفحه ها مراجعه ی مجدد شده که طبیعتاً بیت R آن صفحه‌ها برابر "یک" شده است؛ و به بعضی از صفحه‌ها هم مراجعه‌ی مجدد نشده است که طبیعتاً بیت R آن صفحه‌ها "صفر" باقی مانده است.). حال با ترکیب دو بیت R و M چهار حالت به صورت زیر داریم:

1- R=0 , M=0

2- R=0 , M=1

3- R=1 , M=0

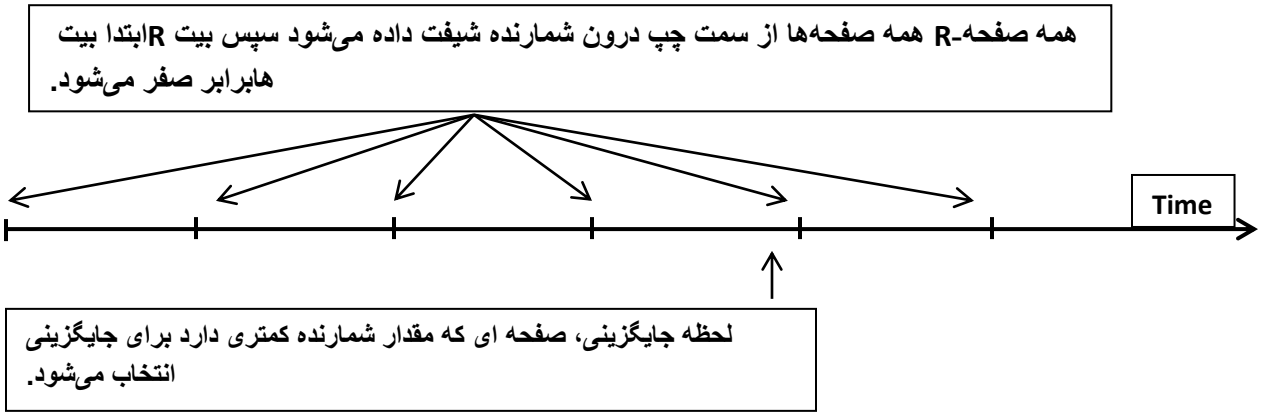
4- R=1 , M=1

در نهایت سیستم عامل سعی می‌کند اولا صفحه‌ای از نوع 1 را برای جایگزینی بیابد، اگر موفق نشد از نوع 2، اگر موفق نشد از نوع 3 و در نهایت در بدترین حالت از نوع 4.

در واقع این الگوریتم در درجه اول ترجیح میدهد صفحه‌های مراجعه شده را در حافظه نگه‌دار و از بین صفحه-های مراجعه نشده صفحه‌ای را برای جایگزینی انتخاب نماید. در درجه دوم ترجیح میدهد که صفحه‌های تغییر کرده را در حافظه اصلی نگه‌دارد و از بین صفحه‌های تغییر نکرده صفحه‌ای را برای جایگزینی انتخاب نماید؛ چرا که اگر صفحه ای بخواهد خارج شود که بیت M آن برابر "یک" است، یعنی محتویات این صفحه در حافظه اصلی تغییر کرده است پس باید این صفحه ابتدا روی هارد نوشته شود و سپس خارج گردد. بنابراین خارج کردن صفحه‌های تغییر یافته برای سیستم‌عامل هزینه اضافه در بر دارد.

**11) الگوریتم سالمندی یا Aging:**

این الگوریتم ترکیبی از الگوریتم NUR و شمارنده است: به این صورت که در رأس بازه های زمانی در الگوریتم NUR، ابتدا بیت R از سمت چپ وارد شمارنده می‌شود سپس بیت R صفر می‌گردد. هنگام خارج کردن صفحه، صفحه ای خارج می شود که شمارنده‌ی کوچکتری دارد. شمارنده کوچکتر نشانه‌ی این است که یا تعداد "یک" های کمتری در شمارنده است یعنی این صفحه در گذشته تعداد مراجعه‌های کمتری داشته است و یا این "یک-"ها در مکان‌های سمت راست شمارنده (با ارزش مکانی پایین‌تر)واقع هستند یعنی از مراجعه‌های به این صفحه در گذشته زمان بیشتری سپری شده است.




| Page number p_no | Block number b_no | Valid/Invalid (V) | Reference Bit (R) | Counter |
|---|---|---|---|---|
| 0 | | | | |
| 1 | | | | |
| 2 | | | | |
| . | | | | |
| . | | | | |
| . | | | | |

**8- مدیریت مجموعه صفحه های مقیم فرآیند:**

**1- سیاست تخصیص:** در مورد تعداد بلاک های در اختیار برنامه می باشد، و به چهار صورت زیر است:

**1-1- تخصیص ثابت:** تعداد بلاک های در اختیار برنامه در کل زمان اجرا ثابت است.

**1-2- تخصیص متغیر:** تعداد بلاک های در اختیار برنامه در طول اجرای برنامه می تواند تغییر کند.

**1-3- تخصیص یکسان:** تعداد بلاک های در اختیار همه ی برنامه ها یکسان است.

**1-4- تخصیص متناسب:** تعداد بلاک های در اختیار برنامه ها متناسب با نیاز آنها خواهد بود.

**2- سیاست واکشی:** در مورد زمان وارد کردن صفحه های یک برنامه به حافضه ی اصلی می باشد.

**2-1- واکشی درخواستی:** صفحه فقط هنگامی به حافضه اصلی منتقل می شود که توسط برنامه برای ادامه اجرا درخواست شده است(لازم است). این روش ورودی/خروجی بیهوده انجام نمی دهد ولی باعث افزایش انتظار برنامه برای انتقال صفحه ها میشود.

**2-2- پیش صفحه بندی:** در این روش به همراه صفحه درخواست شده صفحه های دیگر نیز وارد حافظه اصلی می شود که اگر مراجعات برنامه به صفحه هایش به ترتیب باشد باعث کاهش زمان انتظار می شود، در غیر این صورت باعث ورودی/خروجی بیهوده خواهد شد.

**3- سیاست پاکسازی:** در مورد زمان نوشتن صفحه های تغییر کرده(M=1) روی حافظه جانبی می باشد.

**3-1- پاکسازی درخواستی:** در این روش صفحه ی تغییر کرده هنگامی روی حافظه جانبی نوشته می شود که قرار است صفحه از حافظه اصلی خارج شود.این روش ورودی/خروجی بیهوده ندارد ولی باعث افزایش زمان انتظار هنگام جایگزینی صفحه می گردد.

**3-2- پیش پاکسازی:** در این روش تغییرات صفحه بلافاصله پس از تغییر صفحه بر روی حافظه جانبی نوشته می شود و به زمان جایگزینی موکول نمیشود که باعث کاهش زمان انتظار هنگام جایگزینی می گردد، ولی اگر تغییرات صفحه زیاد باشد زمان ورودی/خروجی زیاد خواهد بود.

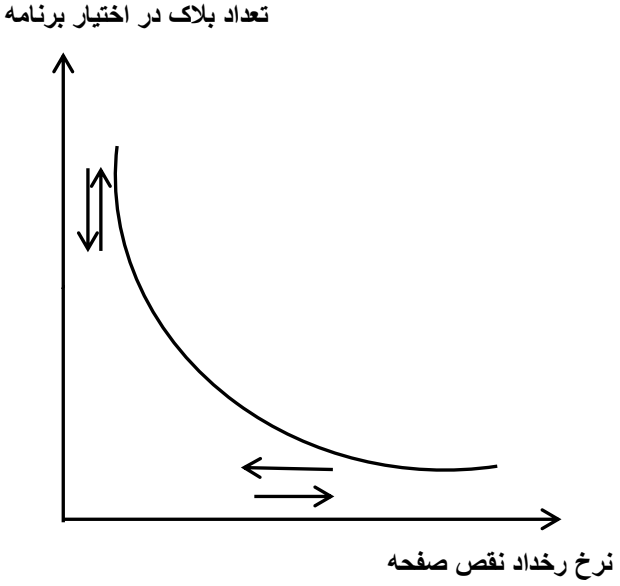
**4- قلمرو جایگزینی:** در مورد اینکه برای آوردن صفحه ای از یک برنامه به حافظه اصلی،صفحه ای از کدام برنامه خارج شود می باشد.

**4-1- قلمرو جایگزینی محلی:** برای آوردن صفحه ای از یک برنامه به حافظه اصلی، صفحه ای از همان برنامه از حافظه اصلی خارج می شود.

**4-2- قلمرو جایگزینی سراسری:** برای آوردن صفحه ای از یک برنامه به حافظه اصلی، صفحه ای از برنامه دیگر خارج می شود که این روش با تخصیص متغیر پیاده سازی می شود.

## 9- کوبیدگی(Thrashing):

**9-1- رابطه میان تعداد بلاک در اختیار برنامه و نرخ نقص صفحه:** هر چه تعداد بلاک در اختیار برنامه افزایش یابد، می‌تواند تعداد صفحه‌های بیشتری را وارد حافظه اصلی نماید. بنابراین تعداد صفحه‌های کمتری از برنامه در حافظه جانبی باقی می‌ماند. پس درصد مواقعی که برنامه به صفحه‌ای احتیاج دارد و این صفحه در حافظه اصلی نیست (در حافظه جانبی است) کاهش می‌یابد یعنی نرخ رخداد نقص صفحه برنامه کاهش می یابد. بدیهی است که با کاهش تعداد بلاک‌های در اختیار برنامه، نرخ رخداد نقص صفحه برنامه افزایش می‌یابد.

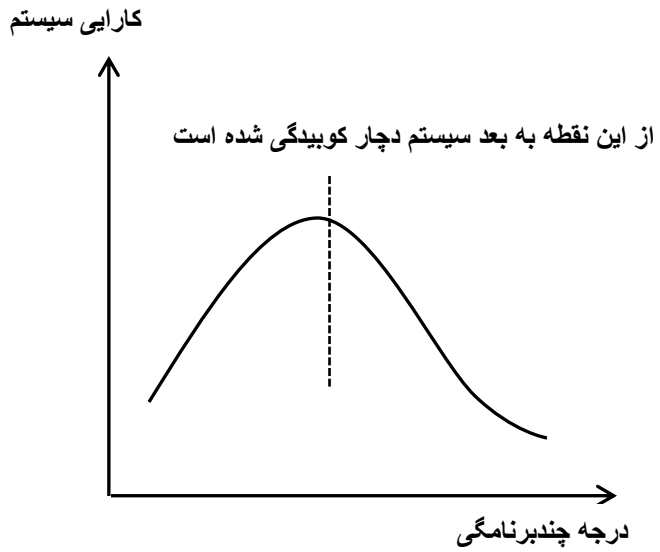



**9-2- درجه چند برنامگی و تأثیر آن در حافظه مجازی و کارایی سیستم:**

**کوبیدگی:** با افزایش تعداد برنامه های در حال اجرا(درجه ی چند برنامگی) تعداد بلاک های در اختیار برنامه ها کاهش می یابد که این به نوبه ی خود باعث افزایش رخداد نقص صفحه و اتلاف توان و زمان سیستم برای جایگزینی صفحه ها می شود در نتیجه کارایی سیستم به شدت افت می کند در این حالت سیستم دچار کوبیدگی شده است.

در شرایط کوبیدگی: 2-5%=کارایی پردازنده

90-93%=کارایی ورودی/خروجی




**10- حافظه مجازی به روش قطعه بندی نیازی:**

این روش همانند قطعه بندی معمولی است با این تفاوت که به جای اینکه کلیه قطعه های برنامه جهت اجرا وارد حافظه اصلی شوند فقط آن قطعه هایی وارد می شوند که برای اجرا لازم هستند.

**11- ترکیب قطعه بندی با صفحه بندی نیازی:**

این روش بهترین روش مدیریت حافظه مجازی بوده و روشی است که در سیستم‌عامل‌های امروزی استفاده می-شود. روش تخصیص و تبدیل آدرس مجازی به فیزیکی در ادامه آمده است.

**روش تخصیص :** در این روش برنامه توسط مترجم و یا برنامه نویس به قطعه هایی با اندازه های متفاوت و محتویات متفاوت بنام قطعه (همانند قطعه بندی معمولی) تقسیم بندی شده است. حافظه اصلی توسط سیستم عامل به قست‌های هم اندازه یا بلاک (همانند صفحه بندی) تقسیم می شود .در هنگام اجرای برنامه هر قطعه به صورت جداگانه به صفحه هایی هم اندازه با بلاک ها شکسته می شود یعنی هر قطعه از چند صفحه تشکیل می-شود، سپس از هر قطعه فقط صفحه هایی وارد حافظه ی اصلی می شود که جهت اجرا لازم است و بقیه صفحه-ها روی حافظه جانبی باقی می‌مانند.

**روش تبدیل آدرس منطقی به فیزیکی :** در این روش هر فرآیند دارای یک جدول قطعه بوده (چون کل برنامه قطعه‌بندی شده است) و هر قطعه دارای یک جدول صفحه منحصر به‌خود است (چون هر قطعه صفحه‌بندی شده است). هر خانه جدول قطعه حاوی آدرس جدول صفحه مربوط به آن قطعه در حافظه است، و جدول

صفحه هر قطعه نشان دهنده بلاک‌هایی است که صفحه‌های آن قطعه در آنها قرار دارند. روند تبدیل آدرس به-این صورت است:

1- آدرس منطقی به سه قسمت شماره ی قطعه (S_no)، شماره ی صفحه در قطعه (P_no) و انحراف درصفحه (offset) تجزیه می شود.

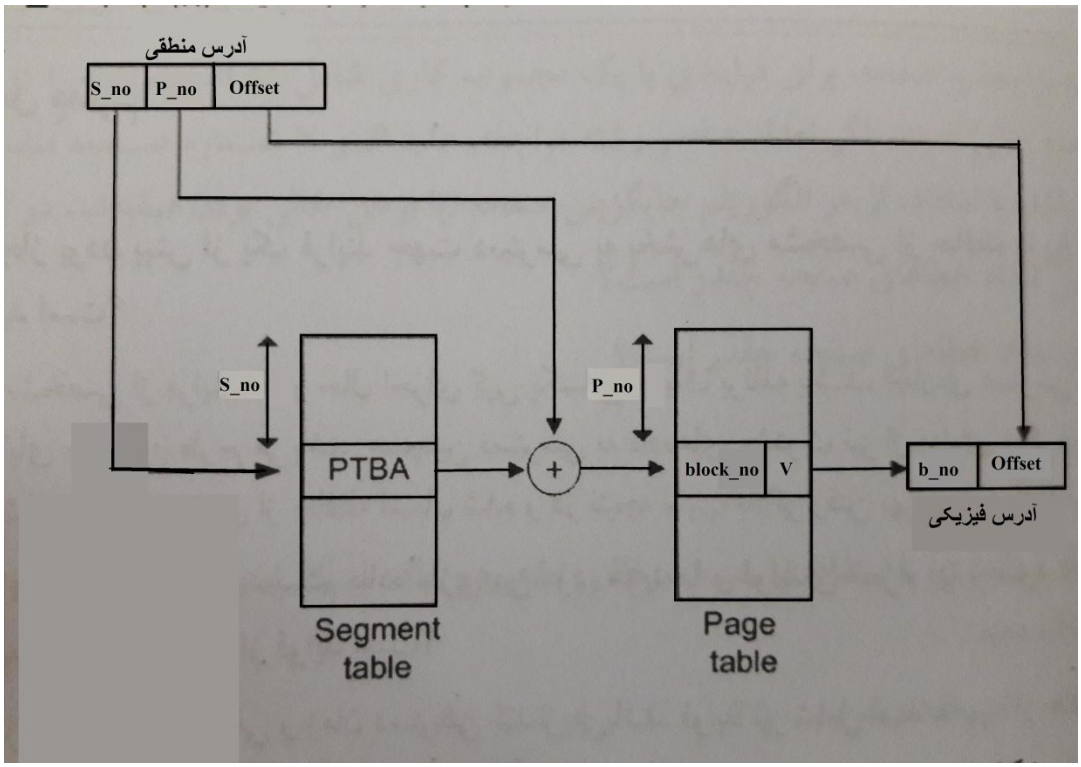
2- شماره‌ی قطعه به جدول اندیس می‌شود تا آدرس شروع جدول صفحه مربوط به این قطعه بدست آید.

3- مقدار P_no با آدرس شروع جدول صفحه جمع شده تا ردیف مربوط به این صفحه، در جدول صفحه‌ی این قطعه مشخص شود (به ردیفی از جدول صفحه برسیم که مربوط به صفحه مورد نظر ماست).

4- در نهایت اگر صفحه در حافظه اصلی باشد (v=1) شماره بلاک مربوطه از جدول صفحه برداشته شده و در کنار Offset قرار می‌گیرد و آدرس فیزیکی تولید می‌شود. اگر صفحه در حافظه اصلی نباشد (v=0)، با رخداد نقص صفحه ابتدا صفحه از حافظه جانبی به حافظه اصلی منتقل شده و شماره بلاک آن مشخص شده و با کنار هم قرار دادن شماره بلاک و offset، آدرس فیزیکی تولید می شود.

نکته مهم: در صورت رخداد نقص صفحه و نیاز به جایگزینی صفحه از هریک از الگوریتم‌های ذکر شده برای جایگزینی صفحه میتوان استفاده نمود که مطابق با آن ساختار جدول صفحه تنظیم خواهد شد.

- PTBA مخفف Page Table Base Address یا آدرس شروع جدول صفحه است.



**12- بهبود صفحه بندی:**

12-1- استفاده از بافر دم دستی ترجمه(TLB) برای افزایش سرعت تبدیل آدرس منطقی به فیزیکی : در این روش بخشی از جدول صفحه در بافر دم دستی ترجمه قرار میگیرد. برای تبدیل آدرس منطقی به فیزیکی، پس از تجزیه آدرس منطقی به دو قسمت شماره صفحه و انحراف در صفحه، ابتدا شماره‌ی صفحه به بافر دم دستی ترجمه برده می شود تا شاید بخشی از جدول صفحه که شامل این شماره صفحه است در آنجا یافت شود. در این صورت شماره‌ی بلاک حاوی این صفحه از آنجا بازیابی می شود. در صورتی که این بخش از جدول صفحه در TLB نباشد و شماره صفحه مورد نظر یافت نشود، شماره‌ی صفحه به جدول صفحه اصلی در حافظه‌ی اصلی اندیس می‌شود. در انجا قطعا شماره‌ی بلاک حاوی این صفحه مشخص خواهد شد. در نهایت با کنار هم قرار دادن شماره بلاک و انحراف در صفحه، آدرس فیزیکی تولید شده و متغیر مورد نظر مورد دسترسی قرار می‌گیرد.

$$Access\ Time = h_{TLB} * t_{TLB} + (1 - h_{TLB})(t_{TLB} + t_{Mem}) + t_{Mem}$$

زمان دسترسی به حافظه: $Access\ Time$

نسبت برخورد حافظه TLB: $h_{TLB}$

زمان دسترسی در حافظه TLB: $t_{TLB}$

زمان دسترسی در حافظه اصلی: $t_{Mem}$

- TLB مخفف Translation Lookaside Buffer یا بافر دم‌دستی ترجمه است.

# فصل ششم: بن بست(DEADLOCK)

در یکی سیستم چندبرنامگی که همواره چندین فرآیند برای در اختیار گرفتن منابع سیستم با یکدیگر رقابت می‌کنند، شرایطی پیش می‌آید که فرآیندها راه استفاده از منابع را برای یکدیگر سد می‌کنند. منابع سیستم می-توانند سخت-افزاری مانند دیسک سخت، چاپگر، اسکنر،... و یا نرم‌افزاری مانند محتوای یک فایل یا رکوردی از یک پایگاه داده باشند. بنابراین تعریف رسمی زیر برای بن‌بست ارائه شده است:

**1- تعریف بن بست:** مجموعه‌ای از فرایندها در حالت بن بست هستند اگر هر فرایند منتظر آزاد شدن منبعی باشد که این منبع در اختیار از فرایند دیگری از همین مجموعه است.

**2- مراحل استفاده از یک منبع توسط فرآیند:**

- درخواست منبع (Resource Request): در این مرحله فرآیند منبع مورد نیاز خود را از سیستم‌عامل درخواست می‌کند.
- زمان‌بندی منبع (Resource Scheduling): اگر چندین فرآیند بصورت همزمان برای استفاده از یک منبع درخواست داده باشند و آن منبع آزاد باشد، سیستم‌عامل باید برای ترتیب استفاده آن فرآیند از منابع زمان‌بندی انجام دهد. در واقع زمان‌بندی منبع، ترتیب استفاده فرآیندها از یک منبع را تعیین می‌کند.
- تخصیص و استفاده از منبع (Resource Allocation and Use): در این مرحله، طبق زمان‌بندی انجام شده منبع در اختیار فرآیند تعیین شده در زمان‌بندی قرار می‌گیرد و فرآیند از منبع استفاده می-کند.

- رهاسازی منبع (Resource Release): در مرحله آخر پس ازا استفاده فرآیند از منبع، فرآیند منبع در اختیار خود را رها می‌کند. بدیهی است که منبع رها شده می‌تواند در اختیار فرآیند دیگری که درخواست استفاده از این منبع را داده است قرار گیرد.

بعنوان مثال فرض کنید دو فرآیند A و B در حین اجرا به دو منبع هارد دیسک (HDD) و چاپگر (PRN) احتیاج داشته باشند. فرآیند A ابتدا به هارد دیسک احتیاج دارد و سپس به چاپگر، در حالیکه فرآیند B ابتدا به چاپگر احتیاج دارد و سپس به هارد دیسک. بنابراین دنباله دستورات زیر در این دو فرآیند وجود دارد:

| Process A | Process B |
|---|---|
| Request HDD | Request PRN |
| Request PRN | Request HDD |
| Use HDD and PRN | Use HDD and PRN |
| Release HDD | Release PRN |
| Release PRN | Release HDD |

حال اگر این دو فرآیند در یک سیستم چندبرنامه‌ای با یک الگوریتم زمان‌بندی پردازنده غیرانحصاری مانند گردش‌نوبت (Round Robin) اجرا شوند، اگر پس از دستور Request HDD در فرآیند A و در اختیار گرفتن هارد دیسک توسط فرآیند A تعویض متن به فرآیند B صورت گیرد آنگاه فرآیند B نیز با اجرای دستور Request PRN چاپگر را در اختیار می‌گیرد. در نهایت چون هارد دیسک در اختیار فرآیند A و چاپگر در اختیار فرآیند B است، هر دو فرآیند در اجرای دستور بعدی که درخواست برای منبع دیگر است مسدود خواهند شد (چون هریک منبعی را می‌خواهند که در اختیار فرآیند مقابل است). بنابراین این دو فرآیند دچار بن‌بست می‌شوند.

**3- شرایط چهارگانه کافمن برای وقوع بن‌بست:** شرایط چهارگانه کافمن بیان کننده عوامل زمینه‌ساز بن‌بست در سیستم است. بعبارت دیگر چنانچه در سیستمی این چهار شرط بصورت همزمان وجود داشته باشند، در این سیستم ***احتمال بن‌بست*** وجود دارد اما در صورتی که بن‌بست رخ دهد حتما این شرایط در سیستم‌بوده است. این شرایط عبارتند از:

- انحصار متقابل (Mutual Exclusion): طبق این شرط هر منبع در هر لحظه یا آزاد است یا توسط فقط یک فرایند قابل استفاده است، بعبارت دیگر امکان استفاده اشتراکی از یک منبع در یک لحظه وجود ندارد.
- نگهداری و انتظار (Hold and Wait): بر طبق این شرط یک فرایند میتواند در حال در اختیار داشتن منابعی برای منابع دیگر نیز در خواست دهد و در صورت آزاد نبودن آن منابع منتظر بماند.
- منابع بدون تخلیه پیش‌هنگام (Non-Preemptable Resource): بر طبق این شرط هر فرایند منابع در اختیار خود را پس از استفاده داوطلبانه آزاد میکند و سیستم‌عامل نمی‌تواند منابع را به اجبار از فرآیندها بازپس بگیرد.
- انتظار چرخشی (Circular Wait): بر طبق این شرط باید زنجیره ی بسته شده ای از فرایند های منتظر وجود داشته باشد.(حتما باید زنجیره بسته شود) که هریک منتظر آزاد شدن منبعی توسط فرآیند دیگری از همین مجموعه هستند.

نکته مهم: برای وجود احتمال رخداد بن‌بست، باید این چهار شرط بصورت همزمان در سیستم وجود داشته باشند و چنانچه حتی یکی از این شرایط در سیستم وجود نداشته باشد احتمال وقوع بن‌بست در سیستم وجود ندارد پس بن‌بست رخ نخواهد داد. همچنین شرط چهارم را شرط لازم و کافی می‌گویند چرا که با وجود این شرط، سه شرط قبلی نیز برقرار هستند در حالیکه سه شرط اول فقط شرایط لازم هستند.

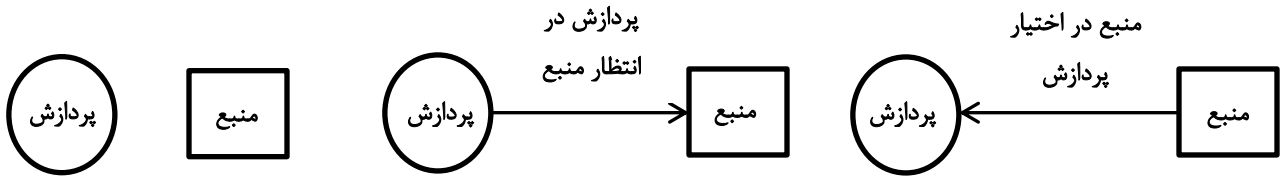
**4- راه های مدیریت بن بست:**

**4-1- روش شترمرغ (Ostrich):** بدلیل اینکه در عمل احتمال وقوع بن‌بست در سیستم‌عامل کم است و همچنین آنگونه که در مباحث آتی بررسی خواهد شد، راههای مقابله یا جلوگیری از وقوع بن‌بست پرهزینه و یا بعضا غیرممکن هستند، در این روش سیستم عامل وقوع بن بست را نادیده می گیرد و حل آن را بر عهده ی کار بر می گذارد که کاربر میتواند با بستن تعدادی از برنامه ها به بن بست خاتمه دهد. بعبارت دیگر این روش بن بست وقوع بن‌بست را قبول ندارد.

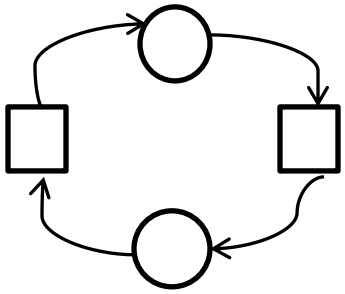
**4-2- روش کشف و ترمیم (Detect & Recovery):** در این روش سیستم عامل در خصوص جلوگیری وقوع بن بست هیچ اقدامی انجام نمی‌دهد بلکه آن را کشف کرده و ترمیم (برطرف) می‌کند.

**الف)روش کشف بن بست:** سیستم عامل به کمک گراف تخصیص منابع وقوع بن بست را کشف می کند.گراف تخصیص منابع گرافی جهت دار است که گره‌های گراف پردازش‌ها و منابع سیستم را نشان می‌دهند. اگر گره بصورت دایره باشد نشان دهنده پردازش است و اگر بصورت مربع/مستطیل باشد نشان دهنده منبع است. اگر

کمانی از سوی یک پردازش به سوی یک منبع باشد آنگاه پردازش در انتظار در اختیار گرفتن(استفاده) آن منبع است. اگر کمانی از سوی یک منبع به سوی یک پردازش باشد آنگاه پردازش در حال استفاده از آن منبع است. به صورت زیر است:




- اگر منابع یک نمونه ای باشند انگاه وجود حلقه‌ی جهت دار در گراف شرط لازم و کافی برای وقوع بن‌بست بوده و پردازش‌های درون حلقه در حالت بن بست هستند. حلقه جهت‌دار، حلقه‌ای است که از یک گره آغاز شده و با طی کردن یال‌ها موافق با جهت کمان‌ها مجددا به همان گره اولیه رسید.



- اگر منابع چند نمونه‌ای باشند آنگاه وقوع حلقه جهت دار در گراف تخصیص منابع شرط لازم ولی غیرکافی برای وقوع بن‌بست است. اگر همه فرآیندها به نحوی اجرا شوند، بن‌بست رخ نداده است. اگر بعضی از فرآیندها اجرا نشوند، بن‌بست رخ داده است.

**نکته مهم:** فرآیندی که همه منابع مورد نیاز خود را در اختیار دارد، اجرا شده، خاتمه یافته و منابع در اختیارش آزاد می‌شود و میتوان این منابع را در اختیار سایر فرآیندهایی گذاشت که این منابع را درخواست کرده‌اند.

**ب) روش ترمیم بن‌بست:** برای ترمیم بن بست دو صورت عمل خواهد شد:

- **کشتن فرآیند (Kill Process):** این کار به دو صورت انجام می‌شود، **روش اول** انتخاب یکی از فرآیندهای قرار گرفته در حلقه بن‌بست بعنوان قربانی و خاتمه یافتن این فرآیند توسط سیستم‌عامل است. سپس منابع در

اختیار این فرآیند که آزاد شده اند به سایر فرآیندهای درون حلقه که این منابع را درخواست کرده‌اند اختصاص می‌یابد. اگر با خاتمه این فرآیند حلقه بن‌بست شکسته نشود، فرآیند دیگری از حلقه انتخاب شده و همین عمل تکرار می‌شود. سربار این روش زیاد است چون پس از خاتمه هر فرآیند توسط سیستم‌عامل باید مجددا الگوریتم کشف بن‌بست اجرا شده و تعیین کند که آیا حلقه بن‌بست برطرف شده است یا خیر؟. روش دوم خاتمه کلیه فرآیندهای قرار گرفته در حلقه بن‌بست است، بدیهی است که در این روش در همان لحظه حلقه بن‌بست از بین می‌رود.

**نکته مهم:** در حالت کلی هزینه و اثرات جانبی این روش زیاد است زیرا اولا، تمامی محاسبات و پردازش‌های فرآیند تا این لحظه از بین رفته و باید فرآیند مجددا از ابتدا اجرا شود، حتی پردازش‌ها و خدمات سیستم‌عامل در جهت اجرای این فرآیندها هدر می‌رود. ثانیا، ممکن است خاتمه فرآیندها در وسط اجرا باعث بروز ناسازگاری در سیستم‌گردد؛ بعنوان مثال فرآیندی که در حال بهنگام سازی یک پایگاه داده یا رکوردی از یک فایل داده‌ای است و در حین این بهنگام‌سازی بصورت پیش‌بینی نشده خاتمه می‌یابد.

- **بازپس گیری منابع:** برای جلوگیری از بروز مشکلات ذکر شده، در این روش یکی از فرآیندهای قرار گرفته در حلقه بن‌بست بعنوان قربانی انتخاب شده ولی فرآیند قربانی توسط سیستم‌عامل خاتمه نمی‌یابد و فقط منابع در اختیارش از او گرفته شده و به سایر فرآیندهای درون حلقه که این منابع را درخواست کرده‌اند اختصاص می‌یابد. اگر با این عمل حلقه بن‌بست شکسته نشود، فرآیند دیگری از حلقه انتخاب شده و همین عمل تکرار می‌شود.

**ج) معیارهای انتخاب قربانی:**

- فرآیندی که منابع بیشتری در اختیار دارد.
- فرآیندی که اولویت پایین تری دارد.
- فرآیندی که زمان کمتری از اجرای آن گذشته است.
- فرآیندی که درخواست وی باعث وقوع بن‌بست شد.
- فرآیندی که کمترین خروجی را تا الان داشته است.

**3-4- روش پیشگیری از بن‌بست (Deadlock Prevention):** در این خانواده از روش‌ها، سیستم عامل با از بین بردن یکی از شرایط چهارگانه کافمن از وقوع بن‌بست پیشگیری می‌کند.

**الف) نقض شرط انحصار متقابل:** در این روش برای منابع خاصی که دسترسی مستقیم فرآیند به منبع مورد نیاز نباشد مانند چاپگر میتوان توسط Spool کردن منابع، شرط انحصار متقابل را از بین برد و به چندین فرآیند اجازه داد تا بصورت همزمان به یک چاپگر مجازی دسترسی پیدا کنند. شایان ذکر است که این روش برای همه منابع قابل بکارگیری نیست.

سیستم Spooling راه‌حلی برای کارکردن با دستگاه‌های I/O انحصاری، در یک سیستم چندبرنامگی است. در این روش یک فرآیند ویژه بنام شبح (Deamon) و یک فهرست مخصوص بنام فهرست spooling ایجاد می-شود. برای چاپ کردن یک فایل، ابتدا یک فرآیند تمامی فایل را در فهرست spooling قرار می دهد. چاپ فایل های درون این فهرست بر عهده شبح است که تنها فرآیندی است که اجازه استفاده از فایل مخصوص چاپگر را دارد.

Spooling تنها در چاپگر به کار نمی رود، بلکه در وضعیت‌های دیگری نیز استفاده می شود. برای مثال، انتقال فایل از طریق یک شبکه معمولاً توسط یک شبح شبکه‌ای انجام می شود. برای ارسال فایل به مقصدی مشخص، کاربر آن را در فهرست spooling شبکه قرار می دهد .سپس شبح شبکه آن را خارج ساخته و منتقل می کند.

در واقع تکنیک Spooling دسترسی انحصاری به منبع مورد نظر واقعی را از بین برده و بجای آن فرآیندها با یک منبع مجازی در ارتباط هستند.

**ب) نقض شرط نگهداری و انتظار:** به دو روش زیر میتوان شرط نگهداری و انتظار را نقض کرد:

**1- تخصیص کلی یا یکجا (Total Allocation):** در این روش هر فرآیند در ابتدای اجرا منابع مورد نیاز خود را به سیستم‌عامل اعلام می‌کند. اگر این منابع آزاد باشند، به فرآیند تخصیص داده می‌شوند و فرآیند اجرا می-شود؛ ولی در حین اجرا حق ندارد منبع دیگری را در خواست نماید. اگر همه یا بعضی از این منابع آزاد نباشند، هیچ منبعی به فرآیند تخصیص داده نشده و فرایند مسدود می‌شود. در یک جمله تخصیص بصورت همه یا هیچ است. **معایب این روش عبارتند از:** دانستن منابع مورد نیاز فرآیند در ابتدای اجرا غیرممکن بوده و این روش غیرقابل پیاده‌سازی است، باعث استفاده غیر بهینه از منابع می‌شود (اتلاف منابع) زیرا ممکن است یک فرآیند منابع مورد نیاز خود را از ابتدای اجرا در اختیار بگیرد و فقط مدت زمان کوتاهی از آنها استفاده نموده و در بقیه زمان اجرا آنها را بیکار نگه‌دارد، باعث تاخیر بیش‌از حد در شروع اجرای فرآیند می‌شود زیرا ممکن است همگی منابع مورد نیاز فرآیند بصورت همزمان آزاد نباشند و فرآیند باید مدت زمان زیادی منتظر بماند تا همگی این منابع با هم آزاد شوند.

**2- رهاسازی و درخواست مجدد:** در این روش هرگاه فرآیندی می‌خواهد برای منبع جدید درخواست دهد، ابتدا منابع در اختیار خود را موقتا آزاد کرده و سپس درخواست خود را مطرح می‌کند. اگر منبع جدید آزاد باشد سیستم‌عامل منبع جدید را تخصیص داده و منابع قبلی را نیز به فرآیند باز می‌گرداند؛ در غیراینصورت (منبع جدید آزاد نباشد) منابع آزاد شده قبلی به فرآیند بازگردانده نشده و فرآیند مسدود خواهد شد. این روش غیر قابل پیاده‌سازی است چرا که طبق شرط سوم کافمن منابع دارای خاصیت تخلیه پیش‌هنگام نیستند و نمی‌توان فرآیند را مجبور کرد که بر خلاف میل خود و بصورت غیرداوطلبانه منابع در اختیار خود را آزاد نماید. بدیهی است اگر شرط سوم کافمن برقرار نباشد اصلا بن‌بستی رخ نخواهد داد.

**ج) نقض شرط منابع بدون تخلیه پیش‌هنگام:** در این روش هرگاه فرآیندی در خواست استفاده از منبعی را به سیستم‌عامل میدهد اگر این منبع آزاد باشد به فرآیند تخصیص داده می‌شود. اگر این منبع آزاد نباشد و در اختیار فرآیند دیگری باشد، سیستم‌عامل منبع مذکور از آن فرآیند گرفته و به فرآیند متقاضی تخصیص می‌دهد. این روش هنگامی قابل پیاده‌سازی است که فرآیندها اولویت یکسان نداشته باشند یعنی سیستم‌عامل بتواند تصمیم بگیرد که منبعی را از کدام فرآیند گرفته و در اختیار فرآیند دیگری قرار دهد. همچنین این روش برای منابعی قابل بکارگیری است که وضعیت منبع در هنگام گرفته شدن از فرآیند قابل ذخیره شدن باشد تا در زمانی بعدتر فرآیند بتواند ادامه کار خود با آن منبع را انجام دهد (مثلا پردازنده، گذرگاه حافظه، دیسک سخت)، بعنوان مثال نمیتوان در حین چاپ یک فایل چاپگر را از فرآیندی گرفته و آن را در اختیار فرآیند دیگری قرار داد. عیب این روش این است که باعث بروز مشکل ***گرسنگی*** برای فرایندهای کم اولویت می‌شود.

**د) نقض شرط انتظار چرخشی:** برای از بین بردن شرط انتظار چرخشی، یک ترتیب در اختیارگیری منابع برای فرآیندها تعریف می‌کنیم. در این روش به هر منبع یک شماره یکتا اختصاص داده می‌شود (مثلا دیسک سخت = 1، صفحه کلید = 2، چاپگر = 3، اسکنر = 4 و ...). حال اگر فرآیندی منبع شماره i را در اختیار داشته باشد فقط مجاز است برای منابع با شماره بیشتر از i در خواست دهد یا بعبارت دیگر درخواست تخصیص منبع بر اساس صعودی شماره صورت می‌گیرد. همچنین اگر فرآیندی منبع شماره j را رها نماید، منابع با شماره بالاتر از j نیز بصورت خودکار آزاد خواهند شد یا بعبارت دیگر رهاسازی منبع بصورت نزولی شماره انجام می‌شود.
**این روش از وقوع بن‌بست جلوگیری می‌کند، چرا که:** برای وقوع بن‌بست لازم است فرآیند A که منبع شماره m را در اختیار دارد برای منبع شماره n در خواست دهد و فرآیند B که منبع شماره n را در اختیار دارد برای منبع شماره m درخواست دهد و این ایجاب می‌کند که همزمان m<n وn<m که این غیر ممکن است.

مشکل این روش این است که در برنامه‌های از قبل نوشته شده، برنامه نویس ترتیب شماره گذاری منابع را نمی-داند که در برنامه به ترتیب صعودی شماره‌ها در خواست تخصیص منابع انجام شود. یک راه حل این است که وقتی فرآیندی برای دراختیار گرفتن منبعی در خواست می‌دهد ابتدا سیستم‌عامل بررسی نماید که اگر در ادامه اجرای برنامه به منبعی با شماره کمتر احتیاج است، ابتدا آن منبع را به فرآیند اختصاص دهد. این کار باعث اتلاف منابع شده و همچنین نیاز به بررسی نیازهای آینده فرآیند دارد. راه‌حل دیگر است که هنگام شماره گذاری منابع به ترتیب درخواست فرآیندها توجه شود که این کار نیز غیر ممکن است چرا که یافتن ترتیب شماره‌گذاری که برای همه فرآیندها مناسب باشد غیر ممکن است.

### 4-4- روش اجتناب از بن‌بست (Deadlock Avoidance)

در روش اجتناب از بن بست سیستم عامل با تخصیص هوشمندانه منابع، سیستم را در وضعیت امن نگهداری میکند. وضعیت امن وضعیتی است که در آن احتمال وقوع بن بست نباشد، از طرف دیگر در وضعیت ناامن احتمال وقوع بن‌بست وجود دارد و وقوع بن‌بست قطعی نیست. در وضعیت امن ترتیبی از تخصیص منابع وجود دارد که میتوان طبق آن منابع را در اختیارفرآیندهای متقاضی قرار داده و آنها را اجرا نمود. این روش به دو صورت زیر اجرا می‌شود:

**الف) عدم شروع فرآیندی که در خواست هایش ممکن است منجر به بن بست گردد:** در این روش هر فرآیند در ابتدای اجرا منابع مورد نیاز خود را اعلام می‌کند. اگر مجموع نیازهای فرآیند مذکور و نیازهای فرآیندهای از قبل پذیرفته شده از کل منابع موجود در سیستم کوچکتر یا مساوی باشد، فرآیند جدید پذیرفته شده و اجرا می‌شود در غیر اینصورت اجرای فرآیند آغاز نخواهد شد. مشکلات این روش عبارتند از:

1- نیاز به دانستن منابع مورد نیاز فرآیند.

2- قحطی زدگی در شروع فرآیند.

3- کاهش درجه چندبرنامگی و کاهش کارایی سیستم.

**ب) عدم پاسخ مثبت به درخواستی که پاسخگویی به آن ممکن است منجر به بن بست گردد:** در این روش از الگوریتم بانکداران استفاده می‌شود.

**1- ماتریس های الگوریتم بانکداران :** در این الگوریتم از 5 ماتریس استفاده می‌شود.

- ماتریس کل منابع موردنیاز فرآیندها (Max): ماتریسی n*m است که سطرهای آن نشان دهنده فرآیندها و ستون-های آن نشان دهنده منابع سیستم است. درایه $Max_{ij}$ نشان دهنده کل تعداد نیاز فرآیند i به منبع شماره j است.

| | $R_1$ | $R_2$ | $R_3$ | $R_4$ | $R_5$ | $R_6$ |
|---|---|---|---|---|---|---|
| $P_1$ | | | | | | |
| $P_2$ | | | | | | |
| $P_3$ | 5 | 7 | 3 | 6 | 1 | 2 |
| $P_4$ | | | | | | |
| $P_5$ | | | | | | |

- ماتریس منابع تخصیص یافته به فرآیندها (Allocation): ماتریسی n*m است که سطرهای آن نشان دهنده فرآیندها و ستون‌های آن نشان دهنده منابع سیستم است. درایه $Allocation_{ij}$ نشان دهنده تعداد منبع شماره j در اختیار فرآیند i است.

| | $R_1$ | $R_2$ | $R_3$ | $R_4$ | $R_5$ | $R_6$ |
|---|---|---|---|---|---|---|
| $P_1$ | | | | | | |
| $P_2$ | | | | | | |
| $P_3$ | 2 | 3 | 1 | 5 | 0 | 2 |
| $P_4$ | | | | | | |
| $P_5$ | | | | | | |

- ماتریس باقیمانده منابع مورد نیاز فرآیندها (Need): ماتریسی n*m است که سطرهای آن نشان دهنده فرآیندها و ستون‌های آن نشان دهنده منابع سیستم است. درایه $Need_{ij}$ نشان دهنده باقیمانده نیاز فرآیند i به منبع شماره j است.

| | $R_1$ | $R_2$ | $R_3$ | $R_4$ | $R_5$ | $R_6$ |
|---|---|---|---|---|---|---|
| $P_1$ | | | | | | |
| $P_2$ | | | | | | |
| $P_3$ | 3 | 4 | 2 | 1 | 1 | 0 |
| $P_4$ | | | | | | |
| $P_5$ | | | | | | |

- ماتریس کل منابع اولیه سیستم (R): ماتریسی خطی است که نشان دهنده کل منابع اولیه سیستم است.

| | $R_1$ | $R_2$ | $R_3$ | $R_4$ | $R_5$ | $R_6$ |
|---|---|---|---|---|---|---|
| R | 10 | 12 | 9 | 13 | 8 | 10 |

- ماتریس منابع آزاد فعلی سیستم (Available): ماتریسی خطی است که نشان دهنده منابع آزاد فعلی سیستم است.

| | $R_1$ | $R_2$ | $R_3$ | $R_4$ | $R_5$ | $R_6$ |
|---|---|---|---|---|---|---|
| R | 4 | 6 | 5 | 2 | 3 | 1 |

**نکته مهم:** Max = Alloc + Need و R = Ava + X که X جمع ستونی ماتریس Allocation است.

**2-بررسی امن و یا نا امن بودن سیستم:** اگر سیستم عامل بتواند ردیفی در ماتریس***Need*** بیابد که تک تک عناصر آن از عناصر متناظرش در ماتریس***Ava*** کوچکتر یا مساوی باشد به این معنی است که سیستم عامل می تواند با این دارایی ها به آن فرآیند پاسخ دهد. سپس این کار انجام شده و فرآیند اجرا می شود و منابع در اختیارش آزاد میشود ( ردیف مربوط به آن فرآیند در ماتریس ***Allocation*** با ***Ava*** جمع میشود) سپس با ***Ava*** جدید بدست آمده همین کار تکرار می شود.اگر در نهایت بتوان به همه ی فرآیند ها پاسخ داد سیستم در فرآیند امن می باشد

$$\exists pi : Need_{pi} \leq Ava$$

پاسخ به ***pi***

$$Ava <= Ava + Alloc_{pi}$$

**3-بررسی پاسخ مثبت یا منفی به یک درخواست:** اگر فرآیند ***pi*** درخواست ***x*** واحد از منبع $R_j$ را بدهد آنگاه سیستم شرایط را شبیه سازی میکند؛ یعنی به تعداد ***x*** واحد از $Need_{piRj}$ کم میکند، به اندازه ***x*** واحد به $Alloc_{piRj}$ اضافه می کند، به تعداد ***x*** واحد از $Ava_{Rj}$ کم میکند. اگر شرایط جدید امن باشد پاسخ مثبت است در غیر اینصورت پاسخ منفی است و عدد ها به حالت اولیه باز میگردد.

معایب الگوریتم بانکداران :

1)اگر تعداد فرآیندها و منابع زیاد باشد زمان اجرای الگوریتم زیاد میشود .

2)نیاز به داشتن (دانستن) منابع مورد نیاز فرآیندها از قبل دارد که باعث غیر عملی بودن این الگوریتم میشود.

3)گم شدن منابع که باعث برهم خوردن محاسبات الگوریتم میشود یعنی اگر منبعی در حین اجرای فرآیندها از کار بیفتد آنگاه محاسبات سیستم نادرست خواهد شد.

## همزمانی فرآیندها و نواحی بحرانی

**محاوره و ارتباط میان فرآیندها:** محاوره و ارتباط میان فرآیندها را می‌توان بر اساس میزان اطلاع فرآیندها از یکدیگر به سه دسته زیر تقسیم کرد:

**۱ـ رقابت:** در این حالت فرآیندها بی‌اطلاع از یکدیگر هستند و نمی‌خواهند که با یکدیگر کار کنند. در چنین حالتی نتایج حاصل از اجرای یک فرآیند مستقل از عملکرد فرآیندهای دیگر است. مسائلی که باید سیستم عامل تحت کنترل داشته باشد عبارتند از: انحصار متقابل، بن بست و گرسنگی.

**۲ـ همکاری به وسیله اشتراک:** در این حالت فرآیندها بصورت غیرمستقیم، مثلاً از طریق یک شیء مشترک، از وجود یکدیگر اطلاع دارند. در چنین حالتی امکان دارد نتایج حاصل از اجرای یک فرآیند به نتایج فرآیند دیگر وابستگی داشته باشد. مسائلی که باید سیستم عامل تحت کنترل داشته باشد عبارتند از: انحصار متقابل، بن بست و گرسنگی.

**۳ـ همکاری به وسیله ارتباط:** در این حالت فرآیندها بصورت مستقیم از وجود یکدیگر مطلع هستند و امکان وابستگی نتایج یک فرآیند به فرآیند دیگر وجود دارد. در این حالت چون فرآیندها از وجود یکدیگر مطلع هستند سیستم عامل باید مسائل مربوط به بن بست و گرسنگی را تحت کنترل داشته باشد.

**شرایط رقابتی:** شرایطی که دو یا چند فرآیند مشغول خواندن یا نوشتن داده‌های اشتراکی هستند و نتیجه نهایی بستگی به این دارد که کدام فرآیند چه موقع اجرا شود، شرایط رقابتی نام دارد.

## ناحیه بحرانی

هنگامی که فرآیند برای دسترسی به داده‌ها یا منابع مشترک با یکدیگر رقابت می‌کنند، در هر فرآیند قطعه کدی وجود دارد که طی اجرای آن به داده یامنبع مشترک دسترسی انجام می‌شود. به این قطعه کد ناحیه‌ی بحرانی یا Critical section گفته می‌شود. هنگامی که فرآیندها ناحیه بحرانی خود را اجرا می‌کنند شرایط رقابتی ایجاد می‌شود زیرا ترتیب اجرای فرآیندها می‌تواند بر روی نتیجه تأثیرگذار باشد. اگر بتوانیم ترتیبی ایجاد کنیم که هیچ دوپروسسی در یک زمان وارد ناحیه بحرانی نشوند از شرایط رقابتی پرهیز نموده‌ایم.

**شرایط راه‌حل‌های ارائه شده برای مسأله ناحیه بحرانی**

**۱ـ انحصار متقابل اعمال گردد:** بر اساس این شرط هیچ دو پروسسی نباید به طور همزمان وارد نواحی بحرانی خود شوند به عبارت دیگر تنها یک فرآیند مجاز است در بخش بحرانی خود باشد.

**۲ـ شرط پیشرفت:** هیچ پروسسی نباشد از بیرون ناحیه بحرانی خود امکان بلوکه کردن پروسس‌های دیگر را داشته باشد. به عبارت دیگر فرآیندی که در بخش غیربحرانی خود متوقف می‌شود باید به طوری عمل کند که هیچ دخالتی در فرآیندهای دیگر نداشته باشد.

**۳ـ شرط انتظار مقید:** هیچ پروسسی نباید بی‌نهایت منتظر ورود به ناحیه بحرانی خود باشد، یعنی برای فرآیندی که نیاز به دسترسی به ناحیه بحرانی دارد نباید امکان به تأخیر انداختن نامحدود آن وجود داشته باشد.

۴ـ هیچ فرض در مورد سرعت وتعداد cpuها و سرعت نسبی فرآیندها و تعداد آنها نمی‌توان داشت.

۵ـ هر فرآیندی تنها برای زمان محدودی در داخل بخش بحرانی خود می‌ماند.

## راه‌حل‌های ارائه شده برای مسأله ناحیه بحرانی

**الف ـ راه‌حل‌های نرم‌افزاری:** در این راه‌حل‌ها با استفاده از دستورات یک زبان برنامه‌سازی نسبت به حل مشکل اقدام می‌شود:

**۱ـ استفاده از متغیرهای قفل (Lock):** در این راه می‌توان یک متغیر Lock یا قفل بصورت مشترک میان فرآیندهای موردنظر قرار داد. مقدار اولیه این متغیر برابر صفر است. هر فرآیند برای ورود به ناحیه بحرانی مقدار Lock را بررسی می‌کند. اگر مقدار آن برابر صفر باشد، یعنی قفل باز است پس آن را برابر یک قرار می‌دهد و وارد ناحیه بحرانی می‌شود. اگر مقدار آن برابر یک باشد، یعنی فرآیند دیگری قبلاً وارد ناحیه بحرانی شده است، باید منتظر بماند تا برابر صفر شود. این روش انحصار متقابل را برآورده نمی‌سازد زیرا اگر فرآیند اول مقدار Lock را برابر صفر ببیند اما قبل از اینکه مقدار Lock را به یک تغییر دهد برش زمانی‌اش تمام شود، بازهر دو فرآیند وارد ناحیه بحرانی می‌شوند. ساختار کلی این الگوریتم در زیر آمده است:

```
int   Lock = 0;

P۱ :                    P۲ :                    P۳ :
≡                       ≡                       ≡
while(Lock == ۱)do;     while(Lock == ۱)do;     while(Lock == ۱)do;
Lock = ۱;               Lock = ۱;               Lock = ۱;
ناحیه بحرانی            ناحیه بحرانی            ناحیه بحرانی
Lock = ۰;               Lock = ۰;               Lock = ۰;
≡                       ≡                       ≡
```

این راه‌حل چند پردازشی و چند پردازنده‌ای با حافظه مشترک است. بدلیل عدم انحصار متقابل راه‌حل قابل قبولی نیست.

**۲ـ تناوب قطعی:** در این روش از یک متغیر نوبت (Turn) استفاده می‌شود. مقدار این متغیر در هر لحظه برابر صفر یا یک خواهد بود. از طرفی تنها دو فرآیند می‌توانند برای دسترسی به ناحیه بحرانی رقابت کنند که $P_0$ و $P_1$ نامیده می‌شوند. هر فرآیند برای ورود به ناحیه بحرانی مقدار Turn را بررسی می‌کند و اگر مقدار Turn با شماره فرآیند برابر نباشد به این معنی است که نوبت متعلق به فرآیند دیگر است. پس این فرآیند در حلقه گرفتار شده و نمی‌تواند وارد ناحیه بحرانی شود. در پایان ناحیه بحرانی نیز هر فرآیند هنگام خروج نوبت را به فرآیند دیگر واگذار می‌کند.

این روش انحصار متقابل را برقرار می‌سازد زیرا مقدار Turn در هر لحظه برابر صفر یا یک است و تنها یک فرآیند اجازه ورود پیدا کرده و فرآیند دیگر نمی‌تواند وارد ناحیه بحرانی شود. شرط انتظار مقید را بر آورده می‌سازد زیرا نوبت بصورت یک در میان عوض می‌شود وهیچ فرآیندی نمی‌تواند بصورت نامحدود مانع از ورود فرآیند دیگر شود.

شرط پیشرفت را برآورده نمی‌سازد زیرا نوبت دقیقاً یک در میان عوض می‌شود و اگر فرآیندی در حال اجرای ناحیه باقیمانده خود باشد و یا اصلاً خاتمه یابد فرآیند رقیب نمی‌تواند وارد ناحیه بحرانی شود یعنی باید فرآیندها بصورت قاطعانه یک درمیان وارد ناحیه بحرانی شوند.

ساختار این الگوریتم در زیر آمده است:

```
                    Int Turn =0;
P0 :                          P1 :
while(1)                      while(1)
{while(Turn!=0)do;            {while(Turn!=1)do;
        ناحیه بحرانی                  ناحیه بحرانی
Turn = 1;                     Turn = 0;
ناحیه باقیمانده               ناحیه باقیمانده
}                             }
```

این راه‌حل دو پردازشی، چند پردازنده‌ای با حافظه مشترک و دارای انتظار مشغول است. بدلیل مشکل گفته شده (عدم رعایت شرط پیشرفت) نمی‌تواند راه‌حل کاملی باشد.

**۳ـ استفاده از متغیر Flag ـ روش اول:** در این روش یک متغیر منطقی بنام Flag دارای دو خانه (بصورت یک آرایه) در نظر می‌گیریم که مقدار اولیه آن False است. هر خانه‌ی این آرایه مربوط به یک فرآیند است. [0] Flag مربوط به $P_0$ و [1] Flag مربوط به $P_1$ می‌باشد. هر فرآیند برای ورود به ناحیه بحرانی Flag رقیب را بررسی می‌کند، اگر True باشد به معنی این است که رقیب درون ناحیه بحرانی است پس باید در حلقه منتظر بماند. در غیر اینصورت فرآیند Flag خود را True کرده تا مانع ورود رقیب شود و خود وارد ناحیه بحرانی می‌شود. ساختار این الگوریتم در زیر آمده است:

```
                Boolean Flag[2]=False;
P0 :                          P1 :
≡                             ≡
while(flag[1] == True)do;     while(Flag[0] == True)do;
Flag[0] = True;               Flag[1] = True;
        ناحیه بحرانی                  ناحیه بحرانی
Flag[0] = False;              Flag[1] = False;
≡                             ≡
```

این راه‌حل انحصار متقابل را برآورده نمی‌سازد زیرا اگر مثلاً فرآیند $P_0$، Flag[1] را False ببیند و از خط حلقه عبور کند اما قبل از دستور Flag [0] = True برش زمانی‌اش تمام شود ونوبت پردازنده به $P_1$ برسد حال $P_1$ هم Flag [0] را برابر False می‌بیند و از خط حلقه عبور می‌کند. در این حالت هر دو فرآیند وارد ناحیه بحرانی می‌شوند.

شرط پیشرفت را برآورده می‌سازد چون اگر فرآیندی متقاضی ورود به ناحیه بحرانی نباشد مقدار Flag را برابر True قرار نمی‌دهد و تأثیری بر فرآیند رقیب نمی‌گذارد. شرط انتظار مقید را برآورده نمی‌سازد چون هنگامی که فرآیندی در حلقه While گرفتار می‌شود فرآیند دیگر که درون ناحیه بحرانی است می‌تواند از ناحیه بحرانی خارجی شده و مجدداً وارد ناحیه بحرانی گردد. این روش نیز انتظار مشغول دارد و دو پردازشی است و راه‌حل صحیحی محسوب نمی‌شود.

**۴ـ استفاده از متغیر Flag ـ روش دوم:** این روش همانند روش قبلی است فقط جای دستور حلقه و مقدار دهی Flag عوض شده است. ساختار این الگوریتم در زیر آمده است:

```
Boolean Flag[2]=False;
P۰ :                                P۱ :
≡                                   ≡
Flag[۰] = True;                     Flag[۱] = True;
while(flag[۱ == True)do;            while(Flag[۰] == True)do;
        ناحیه بحرانی                        ناحیه بحرانی
Flag[۰] = False;                    Flag[۱] = False;
≡                                   ≡
```

این راه‌حل انحصار متقابل را برآورده می‌سازد زیرا فرآیندی وارد ناحیه بحرانی می‌شود که اولاً Flag رقیب را False دیده است یعنی رقیب در ناحیه بحرانی نبوده است و ثانیاً Flag خود را True کرده است تا مانع از ورود رقیب شود. شرط پیشرفت را برآورده می‌سازد زیرا هر فرآیند هنگام خروج از ناحیه بحرانی Flag خود را False می‌کند بنابراین فرآیند دیگر می‌تواند بارها وارد ناحیه بحرانی شود و خارج شود. شرط انتظار مقید نیز برآورده نخواهد شد. همچنین اگر فرآیند مثلاً $P_0$، Flag خود را True کند اما قبل از حلقه برش زمانی‌اش تمام شود و نوبت پردازنده به $P_1$ برسد، حال $P_1$ هم Flag خود را True می‌کند و در حلقه گرفتار می‌شود در این شرایط اگر نوبت پردازنده مجدداً به $P_0$ برسد، $P_0$ هم در حلقه گرفتار خواهد شد و در نتیجه دو فرآیند در بن‌بست قرار می‌گیرند. این روش نیز راه حل قابل قبولی نیست. انتظار مشغول دارد و دو پردازشی است.

**۵ـ راه حل پیترسون:** این روش ترکیبی از روشهای قبلی است. یعنی بصورت همزمان از متغیر نوبت و Flag استفاده می‌کند. در این روش هر فرآیند ابتدا Flag خود را True می‌کند و در یک اقدام پیشگیرانه نوبت را به رقیب واگذار می‌کند. حال اگر Flag رقیب برابر True باشد، چون نوبت هم به او واگذار شده است، پس فرآیند مورد نظر در حلقه گرفتار خواهد شد. اما اگر flag رقیب برابر False باشد یا نوبت متعلق به او نباشد فرآیند موردنظر دارد ناحیه بحرانی خواهد شد. ساختار این الگوریتم در ادامه آمده است:

```
Boolean Flag[2]=False;In  Turn = ۰;
P۰ :                                          P۱ :
≡                                             ≡
Flag[۰] = True;                               Flag[۱] = True;
Turn = ۱;                                     Turn = ۰;
while(Flag[۱] == True and Turn==۱)do;         while(Flag[۰] == True and Turn==۰)do;
                ناحیه بحرانی                                  ناحیه بحرانی
Flag[۰] = False;                              Flag[۱] = False;
≡                                             ≡
```

این روش انحصار متقابل را برآورده می‌سازد زیرا فرآیند در صورتی وارد ناحیه بحرانی می‌شود که اولاً Flag رقیب False بوده باشد و یا اگر ثانیاً Flag هر دو True باشد، Turn به یک فرآیند اجازه ورود می‌دهد و فرآیند دیگر موفق به ورود نخواهد شد. شرط انتظار مقید را بر آورده می‌سازد زیرا نوبت دائماً تعویض می‌شود. شرط پیشرفت را برآورده می‌سازد زیرا هر فرآیند هنگام خروج از ناحیه بحرانی Flag خود را False می‌کند و بنابراین رقیبش می‌تواند بارها وارد ناحیه بحرانی شود و خارج شود، یعنی اگر فرآیندی نخواهد وارد ناحیه بحرانی شود، رقیب می‌تواند وارد ناحیه بحرانی شود. پس هر سه شرط برقرار بوده و راه حل قابل قبولی محسوب می‌شود. این راه‌حل دو پردازشی بوده و انتظار مشغول دارد و چند پردازنده ای با حافظه مشترک است.

### ب ـ راه حلهای سخت افزاری

در این خانواده از راه‌حل‌ها به کمک دستورات زبان ماشین و یااز کار انداختن وقفه‌ها نسبت به حل مشکل اقدام می‌شود.

**۱ـ از کار انداختن وقفه‌ها:** یکی ازراه‌حل‌های مسأله ناحیه بحرانی این است که هر پروسس (فرآیند) هنگام ورود به ناحیه بحرانی همه وقفه‌ها را از کار بیاندازد و هنگام خروج از ناحیه بحرانی مجدداً همه وقفه‌ها را فعال سازد. در چنین حالتی دیگر وقفه پایان برش زمانی رخ نخواهد داد و بنابراین هنگامی که یک فرآیند در حال اجرای دستورات ناحیه بحرانی است قطع نخواهد شد و پردازنده از این فرآیند به فرآیند دیگری سوییچ نمی‌کند. ساختار این الگوریتم به صورت زیر است.

```
Pi : i = ۱,۲,۳,...
≡
Disable INT
    ناحیه بحرانی
Enable INT
≡
```

این روش انحصار متقابل را فراهم می‌آورد چون اولین فرآیندی که موفق به غیر فعال کردن وقفه‌ها شود وارد ناحیه بحرانی می‌شود و سایر فرآیندها موفق به ورود نخواهند شد. شرط پیشرفت را دارد چون هر فرآیند هنگام خروج از ناحیه بحرانی وقفه ها را فعال ساخته و مانع ورود سایر فرآیندها نخواهد شد. شرط انتظار مقید را ندارد چون هیچ نظام صف‌بندی برای فرآیندهای متقاضی ورود به ناحیه بحرانی ندارد. از کار انداختن وقفه ها می‌تواند برای سیستم عامل خطرناک باشد و راه حلی چند پردازشی ولی تک پردازنده ای است. استفاده از این روش برای خود سیستم عامل هنگام اجرای نواحی بحرانی مفید است. بر اساس مطالب ذکر شده این روش نیز راه‌حل کامل و مناسبی نیست.

**۲ـ استفاده از دستورالعمل TSL:** دستورالعمل TSL یا Test and Set Lock (بررسی و بستن قفل) در سطح سخت افزار (زبان ماشین) تعریف می‌شود. برای اینکه بتوان از این دستور در جهت حل مسأله ناحیه بحرانی استفاده کرد باید دو مورد از سوی سخت‌افزار تضمین شود: اولاً عملیات خواندن کلمه حافظه و ذخیره سازی مقدار در آن بصورت تجزیه ناپذیر (اتمیک) باشد، ثانیاً در حین اینعمل هیچ فرآیند دیگری وحتی پردازنده دیگری حق دسترسی به این کلمه حافظه را نداشته باشد. بنابراین پردازنده‌ای که این دستور را اجرا می‌کند گذرگاه حافظه را قفل می‌کند تا از دسترسی دیگر پردازنده‌ها به حافظه جلوگیری کند. دستور TSL محتویات یک کلمه از حافظه را خوانده ودر یک ثبات قرار می‌دهدو سپس مقدار آن خانه حافظه را یک می‌کند که این عملیات بصورت اتمیک است.

هنگامی که فرآیند می‌خواهد وارد ناحیه بحراین شود تابع Enter و هنگامی که می‌خواهد خارج شود تابع Exit را صدا می‌زند.

```
                                int Lock=۰;// متغیر مشترک میان فرآیندها
Enter :                         Exit :
    TSL Reg, Lock               Mov Lock,۰      Pi : i = ۱,۲,...
    Cmp Reg, ۰                                  ≡
    Jne Enter                                   Enter;
                                                ناحیه بحرانی
                                                Exit;
                                                ≡
```

این روش انحصار متقابل را دارد زیرا اولین فرآیندی که Enter را فراخوانی کند مقدار صفر را از Lock به Reg منتقل می‌کند و مقدار Lock را برابر یک قرار می‌دهد. چون مقدار Reg برابر صفر شده است این فرآیند وارد ناحیه بحرانی می‌شود ولی سایر فرآیندها نمی‌توانند وارد ناحیه بحرانی شوند. شرط پیشرفت را دارد زیرا هر فرآیند هنگام خروج از ناحیه بحرانی مقدار Lock را بوسیله تابع Exit برابر صفر قرار داده یعنی مجدداً قفل را باز می‌کند. شرط انتظار مقید را ندارد چون هیچ نظام نوبت‌دهی برای فرآیندهای متقاضی ورود ندارد پس راه حل قابل قبولی نیست. دارای انتظار مشغول است. چند پردازشی و چند پردازنده‌ای با حافظه مشترک است.

**۳ـ استفاده از دستورالعمل معاوضه یا Swap یا Exchange:** این دستورالعمل زبان ماشین محتوای یک ثبات را با محتوای محلی از حافظه معاوضه می‌کند. انجام عمل معاوضه اولاً بصورت اتمیک بوده و ثانیاً در طی اجرای معاوضه هیچ فرآیند یا پردازنده دیگری نمی‌تواند به همان محل حافظه مراجعه کند. (گذرگاه حافظه قفل می‌شود). ساختار این روش در ادامه آمده است:

```
                    Int Lock =۰; متغیر مشترک میان فرآیندها
P۱ :                P۲ :                P۳ :
≡                   ≡                   ≡
Key۱ = ۱            Key۲ = ۱            Key۳ = ۱
do                  do                  do
Exch(Key۱, Lock);   Exch(Key۲, Lock);   Exch(Key۳, Lock);
while(Key۱ == ۱);   while(Key۲ == ۱);   while(Key۳ == ۱);
    ناحیه بحرانی        ناحیه بحرانی        ناحیه بحرانی
Lock = ۰;           Lock = ۰;           Lock = ۰;
≡                   ≡                   ≡
```

در این روش هر فرآیند متغیر Key اختصاص خود را دارد ولی متغیر Lock بین همه فرآیندها مشترک است. این روش انحصار متقابل را دارد چون اولین فرآیندی که دستور Exch را اجرا کند مقدار Lock را برابر یک کرده ولی مقدار key خودش برابر صفر می‌شود و وارد ناحیه بحرانی می‌شود ولی دیگر فرآیندها نمی‌توانند وارد ناحیه بحرانی شوند چون با اجرای دستور Exch مقدار Key آنها برابر یک می‌شود. همانند روش TSL شرط پیشرفت را داشته ولی شرط انتظار مقید را ندارد و راه‌حل کاملی محسوب نمی‌شود. دارای انتظار مشغول بوده، چند پردازشی و چند پردازنده‌ای اما با حافظه مشترک است.

**انتظار مشغول:** راه حل‌هایی که در آنها فرآیندی که مجاز به ورود به ناحیه بحرانی نیست، در یک حلقه می‌چرخد و زمان برش زمانی خود را تلف می‌کند دارای انتظار مشغول هستند. این راه‌حل‌ها باعث کاهش کارایی پردازنده، اتلاف زمان، کاهش گذردهی سیستم، رخداد اولویت معکوس (Priority Inversion) و گاهاً وقوع بن بست خواهند شد.

**ج ـ راه‌حل ارائه شده توسط سیستم عامل**

**سمافور Semaphor:** یک سمافور یک ساختار سطح بالای برنامه‌نویسی، یک شیء یا کلاس، است که دارای داده و تابع است. داده های یک سمافور عبارتند از متغیر صحیح Value و یک صف Q برای قرار دادن نام فرآیندها. توابع سمافور عبارتند از دوتابع Wait یا Down یا P و Signal یا Up یا V متغیر Value هنگام تعریف یک متغیر از نوع سمافور مقدار دهی اولیه می‌شود. سیستم عامل گارانتی می‌کند که اجرای توابع Wait و Signal بصورت اتمیک است. یعنی به هیچ عنوان هنگامی که فرآیندی تابع Wait یا Signal را فراخوانی کرده. است، برش زمانی پایان نمی‌پذیرد و نوبت پردازنده به هیچ دلیلی به فرآیند‌دیگری داده نمی‌شود.

**تابع Wait:** از مقدار Value یک واحد کم می‌کند و اگر این مقدار منفی شود فرآیند فراخواننده مسدود شده وبه صف Q اضافه می‌شود.

**تابع Signal:** به مقدار Value یک واحد اضافه می‌کند و اگر این مقدار کوچکتر یا مساوی صفر باشد یکی از فرآیندهای مسدود را از صف Q بیدار کرده و به صف Ready اضافه می‌کند.

در ادامه دو پیاده‌زی از توابع Wait و Signal ارائه شده است:

**پیاده سازی ۱:**

```
Wait(S)
   {S.Value = S.Value – ۱;
     if (S.Value<۰)
   {block Process P;
    Add p to S.Q;
   }
}
```

```
Signal(S)
   {S.Value = S.Value + ۱;
     if (S.Value<۰)
   {Wakeup Process P from S.Q;
    Add p to S.Q;
   }
}
```

**پیاده سازی ۲:**

```
Wait(S)
{if (S.Value>۰)
     S.Value = S.Value – ۱;
  else
     {block Process P;
      Add P to S.Q;
     }
  }
```

```
Signal(S)
   {if (S.Q is empty)
       S.Value = S.Value + ۱;
else
   {Wakeup Process P from S.Q;
    Add p to Ready;
   }
}
```

**استفاده از سمافور برای حل مسأله ناحیه‌ی بحرانی:** برای حل مسأله ناحیه بحرانی به کمک سمافور، یک سمافور S با مقدار اولیه یک بصورت مشترک میان تمام فرآیندهای متقاضی در نظر می‌گیریم. هر فرآیند جهت ورود به ناحیه بحرانی Wait (S) را فراخوانی می‌کند. بدیهی است که

اولین فرآیندی که Wait (S) را فراخوانی کند مقدار Value را از یک به صفر تغییر داده و بدون اینکه مسدود شود وارد ناحیه بحرانی می‌شود، اما فرآیندهای بعدی با انجام Wait (S) مقدار Value را منفی خواهند کرد و مسدود خواهند شد و نام آنها در صف Q قرار می‌گیرد پس شرط انحصار متقابل فراهم می‌شود. هر فرآیند هنگام خروج از ناحیه بحرانی Signal (S) را فراخوانی می‌کند که این عمل باعث افزایش یک واحدی S. Value می‌شود. همچنین اگر S. Value <= ۰ باشد به این معنی است که فرآیند مسدود شده وجود دارد، پس فرآیند سر صف Q بیدار شده و وارد ناحیه بحرانی می‌شود. این روش انتظار مقید را برآورده می‌سازد چون فرآیندهایی که نمی‌توانند وارد ناحیه بحرانی شوند در صف قرار داده می‌شوند و به ترتیب از صف برداشته شده و واردناحیه بحرانی می‌شوند. شرط پیشرفت برآورده می‌شود چون هر فرآیند هنگام خروج از ناحیه بحرانی Signal انجام می‌دهد و اثر Wait خود را از بین می‌برد و در ورود سایر فرآیندها تأثیرگذار نخواهد بود. بنابراین این روش نیز راه‌حل قابل قبول محسوب می‌شود. این روش انتظار مشغول ندارد زیرا فرآیندی که موفق به ورود به ناحیه بحرانی نمی‌شود مسدود شده و درحلقه چرخش نمی‌کند. راه‌حلی چند پردازشی و چند پردازنده ای با حافظه مشترک است. ساختار این روش بصورت زیر است:

```
Semaphor S(۱);// سمافور مشترک با مقدار اولیه یک
Pi : i = ۱,۲,۳,...
≡
Wait(S);
ناحیه بحرانی
Signal(S);
≡
```

**نکته** : اگر S.Value کوچکتر از صفر باشد، قدر مطلق آن تعداد فرآیندهای مسدود را نشان می‌دهد.

**نکته** : اگر S.Value بزرگتر از صفر باشد نشان دهنده تعداد فرآیندی است که می‌توانند وارد ناحیه بحرانی شوند.

**نکته** : نام دیگر سمافور، راهنما است.

**د ــ راه حل ارائه شده توسط زبان‌های برنامه‌سازی**

**مانیتور یا ناظر (Monitor)**: یک مانیتور یک ساختار سطح بالای برنامه‌نویسی، یک شیء یا کلاس، است که دارای داده و تابع است. داده‌های یک مانیتور عبارتند از هر داده دلخواه برنامه‌نویس از هر نوع دلخواه. به علاوه یک نوع داده جدید بنام نوع داده شرطی (Conditional) در مانیتور وجود دارد. متغیرهای شرطی فاقد مقدار هستند و فقط دارای یک صف Q هستند، یعنی هر متغیر شرطی دارای یک صف Q است. توابع مانیتور می‌توانند هر تابع دلخواه کاربر باشند همچنین دو تابع Cwait و Csignal بر روی متغیرهای شرطی درون مانتیور قابل انجام هستند. موارد زیر برای توابع و متغیرهای مانیتور قابل ذکر هستند:

۱ـ متغیرهای درون مانیتور از نوع خصوصی (Private) هستند و فقط توسط توابع عضوی مانیتور قابل دسترسی هستند.

۲ـ هرگاه فرآیندی از طریق فراخوانی یک تابع از یک مانیتور، اصطلاحاً، وارد مانیتور گردد تا زمانیکه اجرای این تابع تمام نشود و فرآیند از مانیتور خارج نشود؛ کامپایلر تضمین می‌کند که هیچ فرآیند دیگری حق فراخوانی هیچ تابع دیگری از مانیتور را ندارد و نمی‌تواند وارد مانیتور گردد.

**نکته:** موارد ۱ و ۲ باعث می‌شود که برای هر داده تعریف شده درون مانیتور انحصار متقابل برقرار شود یعنی در هر لحظه تنها یک فرآیند می‌تواند درون مانیتور باشد و به داده‌های درون مانیتور دسترسی کند.

۳ـ تابع Cwait (x) روی متغیر شرطی x فراخوانی می‌شود و بدون قید و شرط فرآیند فراخواننده رامسدود می‌کند و نام آن را به صف Q متعلق به متغیر شرطی x اضافه می‌کند.

۴ـ تابع Csignal (x) روی متغیر شرطی x فراخوانی می‌شود و بدون قید و شرط یک پیام بیدار باش برای صف Q متعلق به متغیر x ارسال می‌کند. اگر فرآیندی مسدود شده باشد، فرآیند سر صف بیدار می‌شود و در غیر اینصورت پیام به هدر می‌رود.

**نکته:** هرگاه فرآیندی تابع Cwait یا Csignal را فراخوانی کند، فرآیند فراخواننده از مانیتور خارج می‌شود. در حالت فراخوانی Cwait فرآیند فراخواننده مسدود می‌شود اما در حالت فراخوانی Csignal فرآیند فراخواننده مسدود نمی‌شود.

**نکته:** استفاده از مانتیور برای حل مسأله ناحیه بحرانی یک راه حل قابل قبول محسوب می‌شود.

**هـ ــ تبادل پیام (Message passing)**

تمامی روش‌های گفته شده برای حل مسأله ناحیه بحرانی مبتنی بر وجود حافظه مشترک هستند. همانگونه که در فصل اول بیان شد در سیستم‌های توزیع شده فرآیندها روی کامپیوترهای مستقل اجرا می‌شوند، بنابراین حافظه مشترک وجود ندارد. در سیستم‌های توزیع شده برای حل مسأله ناحیه بحرانی و ایجاد همزمانی میان فرآیندها از مکانیزم تبادل پیام استفاده می‌شود. در مکانیزم تبادل پیام دو تابع Send و Receive برای ارسال و دریافت پیام میان فرآیندها بکار می‌رود. تابع Send (mes, dest) پیام mes را به مقصد dest ارسال می‌کند. تابع Receive (mes, sor) پیام mes را از مبدأ Sor دریافت می‌کند. عموماً تابع Receive مسدود شونده است یعنی وقتی فرآیندی این تابع را فراخوانی

می‌کند، اگر پیامی از سوی مبداء دریافت نکند مسدود می‌شود. ولی تابع send مسدود شونده نیست با ساختار زیر می‌توان میان دو فرآینـد $P_۱$ و $P_۲$ انحصار متقابل با استفاده از تبادل پیام را ایجاد کرد:

```
P۱ :                                P۲ :
   send(mes, p۲)                        ≡
   while(۱)                             while   (۱)
   {                                    {Receive(mes, P۱);
   Receive(mes, p۲)                     ناحیه بحرانی
   ناحیهٔ بحرانی                       send(mes, p۱);
   send(mes, p۲)                    }
   }                                    ≡
   ≡
```

این راه‌حل انحصار متقابل را تضمین می‌کند. ولی شرط پیشرفت را ندارد چون باید فرآیندها بصورت یک در میان وارد ناحیه بحرانی شوند. ولی بـه همین دلیل انتظار مقید را دارد. و راه‌حلی دو پردازشی است.

اگر بخواهیم روش تبادل پیام را برای بیش از دو فرآیند تعمیم دهیم باید یک صندوق پستی مشترک میان فرآیندها تعریف کنیم. در ابتدا تنها یک پیام تهی در این صندوق قرار داده می‌شود. هر فرآیندی که قصد ورود به ناحیه بحرانی را دارد باید ابتدا موفق شود ایـن پیـام را از صـندوق پـستی دریافت کند. سپس وارد ناحیه بحرانی شود و در نهایت پس از خروج از ناحیه بحرانی مجدداً این پیام را بـه صـندوق پـستی بـاز گردانـد تـا سـایر فرآیندها بتواند وارد ناحیه بحرانی شوند. ساختار کلی این روش بصورت زیر است:

```
Pi : i = ۱,۲,۳
≡
while(۱)
{
 Receive(mes, mailbox); // mailbox دریافت پیام mes از صندوق پستی
ناحیه بحرانی
send(mes, mailbox); // mailbox ارسال پیام mes به صندوق پستی
ناحیه باقیمانده
≡
}
main :
create mailbox; // ساختن صندوق پستی مشترک میان فرآیندها
sand (null, mailbox) // قرار دادن یک پیام تهی در صندوق پستی
start  Pi  ; i = ۱,۲,۳,... آغاز نمودن فرآیندها
```