یادداشت‌های درس سیستم‌های عامل - بخش دوازدهم

از این بخش از درس به حافظه‌ی مجازی می‌پردازیم. چون پیاده‌سازی حافظه‌ی مجازی برای پردازنده‌های رایج امروزی مبتنی بر صفحه‌بندی است، در این بخش از درس فرض می‌کنیم حافظه با کمک صفحه‌بندی مدیریت می‌شود.

- در بخش قبل فرض کرده‌ایم وقتی پردازه‌ای اجرا می‌شود، همه‌ی حافظه‌ی آن پردازه در حافظه قرار می‌گیرد.
- به عبارت دیگر، همه‌ی صفحه‌های مربوط به پردازه در حافظه قرار دارند و سطرهای متناظر آنها در جدول صفحه معتبر هستند.

اما اگر سیستم عامل حافظه‌ی مجازی (Virtual Memory) را پشتیبانی کند، فقط بخشی از حافظه‌ی یک پردازه را در حافظه‌ی اصلی نگه می‌دارد.

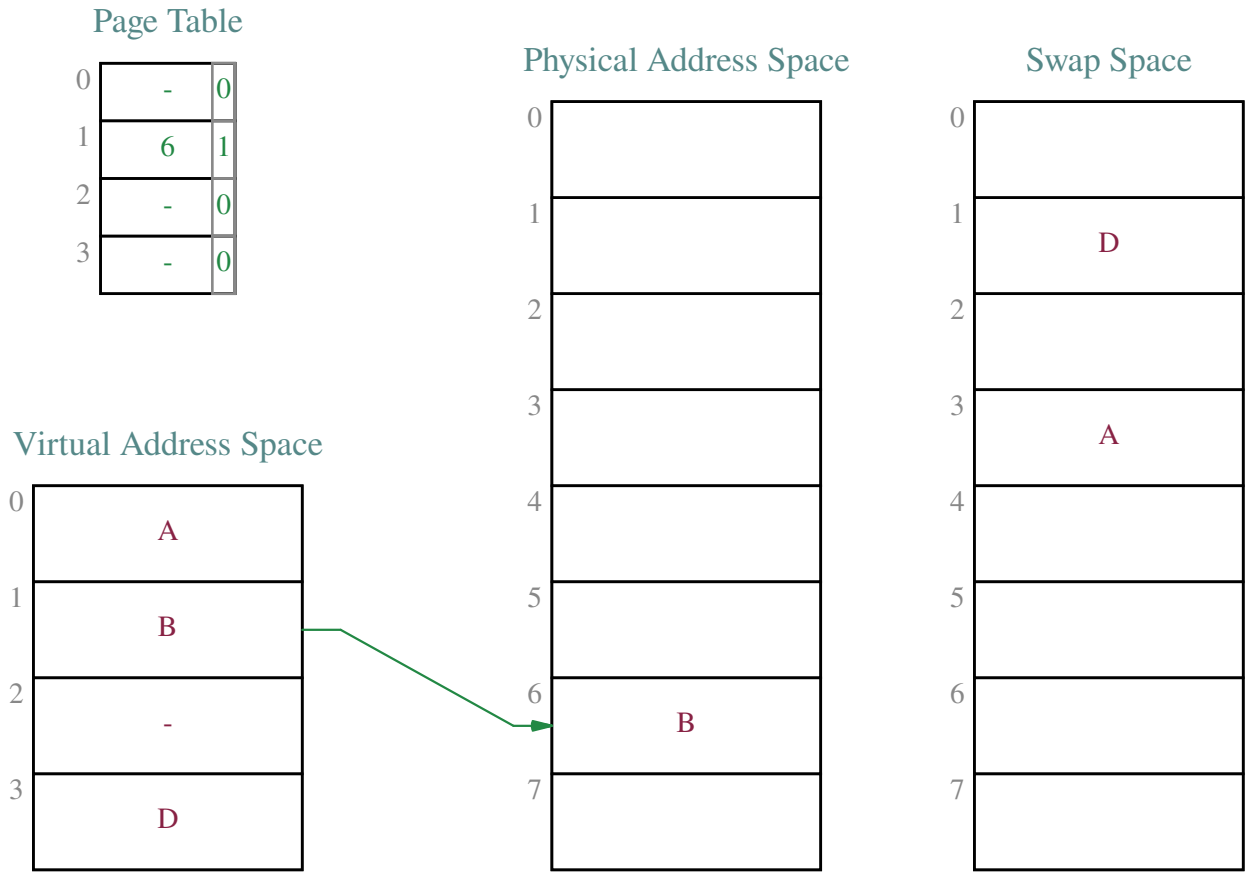
- بنابراین ایده‌ی اصلی در حافظه‌ی مجازی این است: فقط بخشی از صفحات یک پردازه در حافظه‌ی اصلی قرار گیرند.
- اما پردازه‌ها تصور می‌کنند همه‌ی صفحات آنها در حافظه‌ی اصلی هستند. به این دلیل به این شیوه‌ی مدیریت حافظه، حافظه‌ی مجازی گفته می‌شود: هر پردازه حافظه‌ی مجازی دارد که ممکن است در حافظه‌ی اصلی باشد یا خارج از آن نگهداری شود.
- در ادامه به جای فضای آدرس منطقی از فضای آدرس مجازی استفاده می‌کنیم و به جای آدرس منطقی از آدرس مجازی استفاده می‌کنیم.

دلایل استفاده از حافظه‌ی مجازی

- پردازه‌ها در هر لحظه فقط از بخشی از حافظه‌ی خود استفاده می‌کنند؛ برای نمونه، به برخی از خانه‌های آرایه‌هایی که تخصیص داده می‌شوند به ندرت دسترسی انجام می‌شود یا برخی از حالاتی که در کد در نظر گرفته شده است به ندرت اجرا می‌شوند.
- با کمک حافظه‌ی مجازی می‌توان چند پردازه را اجرا کرد که مجموع حافظه‌ای که به آنها تخصیص داده شده است بیشتر از اندازه‌ی حافظه‌ی اصلی باشد.
- حتی یک پردازه می‌تواند بیشتر از کل حافظه‌ی اصلی که در اختیار سیستم عامل هست حافظه تخصیص دهد.
- حافظه‌ی مجازی امکان اشتراک حافظه توسط پردازه‌ها و نگاشت محتویات فایل‌ها به حافظه را به وجود می‌آورد.
- حافظه‌ی مجازی می‌تواند حجم عملیات ورودی یا خروجی را کاهش دهد.

اصول حافظه‌ی مجازی

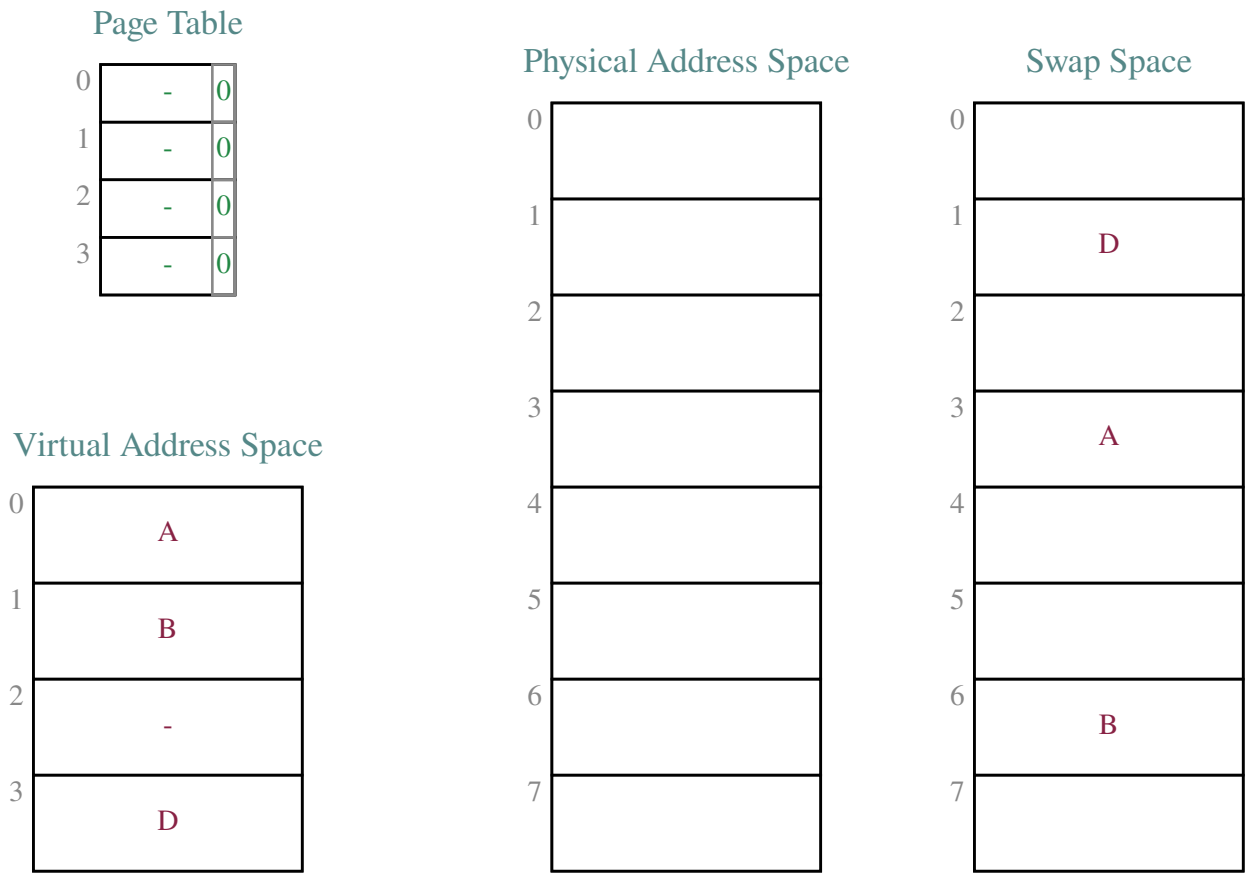
- در حافظه‌ی مجازی، از حافظه‌ی کمکی به نام فضای Swap (Swap Space) استفاده می‌شود (معمولا فضای Swap در دیسک نگهداری می‌شود).
- شکل زیر را در نظر بگیرید.



- این شکل مربوط به یک پردازه است که فضای آدرس منطقی آن شامل چهار صفحه است که به سه صفحه از آنها قاب تخصیص داده شده است.
- اما از سه صفحه‌ی این پردازه، فقط یک صفحه در حافظه‌ی اصلی قرار گرفته است و دو عدد از آنها در فضای Swap قرار دارند.
- در جدول صفحه نیز فقط سطر مربوط به صفحه‌ی B که در حافظه‌ی اصلی قرار دارد معتبر است و سایر سطرها نامعتبر هستند.
- صفحاتی که در فضای Swap قرار دارند، در صورت نیاز به حافظه‌ی اصلی انتقال می‌یابند و سطر مربوط به آنها پر می‌شود.

- در صفحه‌بندی مبتنی بر نیاز (Demand Paging) وقتی پردازه‌ای به صفحه‌ای دسترسی دارد که در فضای Swap قرار دارد، آن صفحه به حافظه‌ی اصلی انتقال می‌یابد (اما ممکن است صفحاتی هم به حافظه انتقال داده شوند که هنوز به آنها دسترسی انجام نشده است).
- در صفحه‌بندی مبتنی بر نیاز خالص (Pure Demand Paging) فقط صفحاتی از فضای Swap به حافظه‌ی اصلی انتقال داده می‌شوند که پردازه به آنها دسترسی داشته باشد.

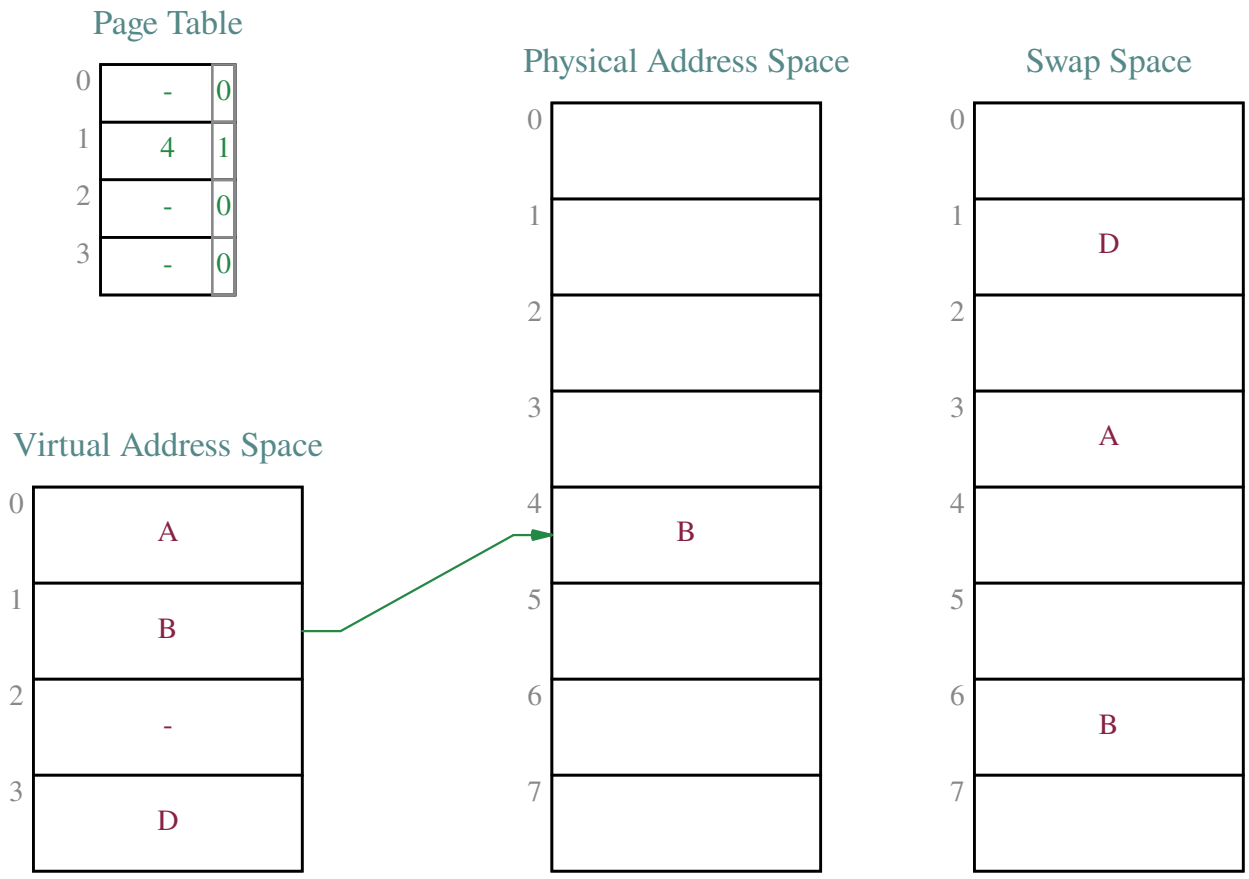
- در ابتدا فرض کنید هیچ صفحه‌ای از پردازه در حافظه‌ی اصلی موجود نباشد.



- فرض کنید اندازه‌ی صفحه شانزده بایت باشد. اگر پردازه به آدرس **15** (در مبنای شانزده) دسترسی داشته باشد، در واقع به صفحه‌ی شماره‌ی یک دسترسی دارد.
- چون این صفحه در جدول صفحه معتبر نیست، خطای صفحه ایجاد می‌شود و سیستم عامل خطا را بررسی می‌کند.
- سیستم عامل مشاهده می‌کند که این صفحه در حافظه‌ی مجازی پردازه موجود است ولی فعلا در حافظه‌ی کمکی است. از این رو، ابتدا یک قاب خالی در حافظه‌ی اصلی در نظر می‌گیرد (مثلا قاب چهارم) و صفحه را از حافظه‌ی اصلی به آن قاب انتقال می‌دهد.
- سپس سطر مربوط به صفحه‌ی یکم را تغییر می‌دهد و با بازگشت به فضای کاربری، دستوری که موجب خطای صفحه شده بود دوباره اجرا می‌شود.

روش کار صفحه‌بندی مبتنی بر نیاز

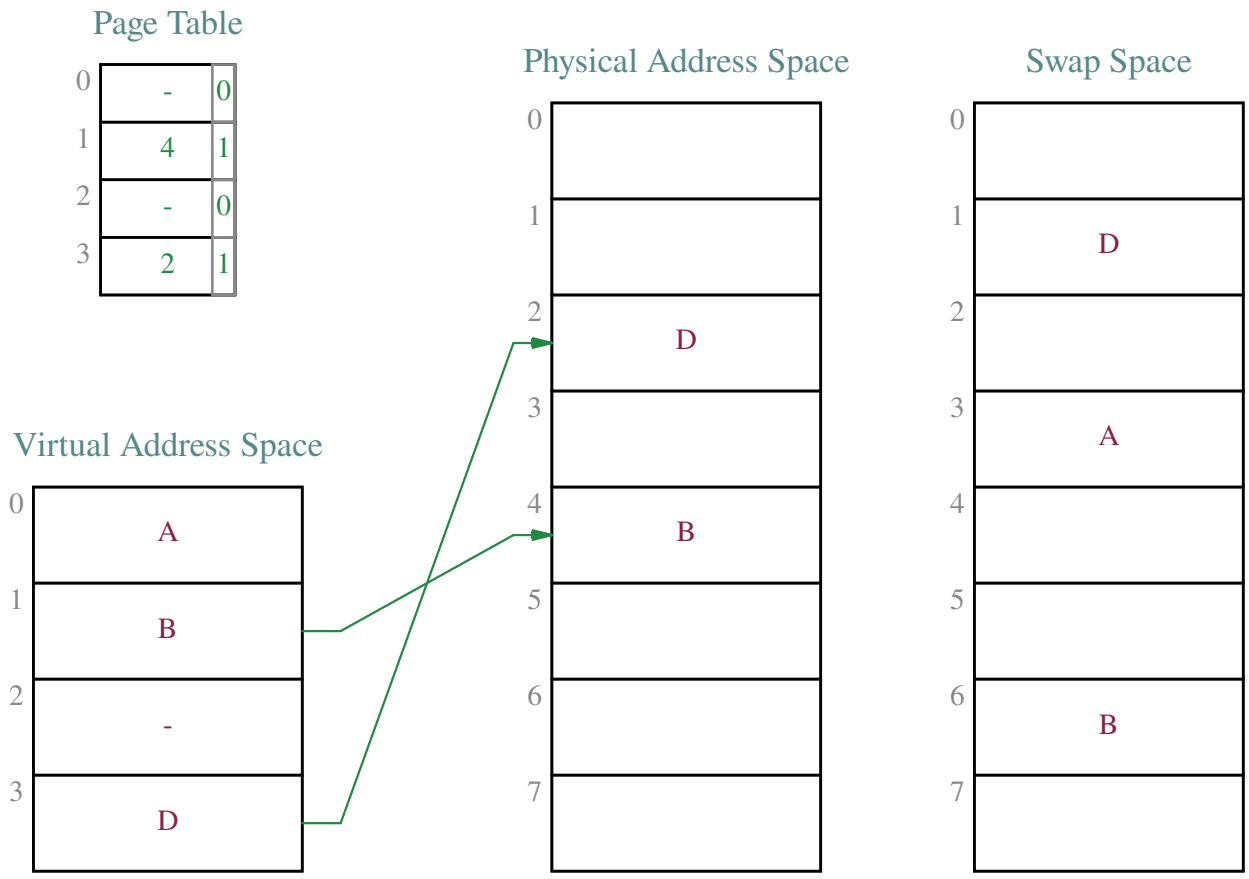
- جدول صفحه به صورت زیر تغییر می‌کند.



- سپس سطر مربوط به صفحه‌ی یکم را تغییر می‌دهد و با بازگشت به فضای کاربری، دستوری که موجب خطای صفحه شده بود دوباره اجرا می‌شود.
- در ادامه اگر پردازه به آدرس **3A** دسترسی داشته باشد، به خاطر دسترسی به صفحه‌ی سوم که در جدول صفحه نامعتبر است، خطا صفحه رخ می‌دهد.
- سیستم عامل خطا را بررسی می‌کند و چون صفحه‌ی سوم جزء حافظه‌ی مجازی پردازه هست و در حافظه‌ی کمکی قرار دارد، آن را به یک قاب خالی در حافظه‌ی اصلی انتقال می‌دهد (برای نمونه قاب دوم) و جدول صفحه را به روز می‌کند.

روش کار صفحه‌بندی مبتنی بر نیاز

- جدول صفحه به صورت زیر تغییر می‌کند.



- با بازگشت به فضای کاربری، دستوری که موجب خطای صفحه شده بود دوباره اجرا می‌شود.
- دقت کنید که ممکن است پردازه به یک آدرس مجازی غیر مجاز دسترسی داشته باشد (مثل **28**). در این صورت سیستم عامل خطا را به کاربر گزارش می‌دهد یا پردازه را خاتمه می‌دهد.

زمان مؤثر دسترسی به حافظه

- اگر زمان هر دسترسی به یک آدرس منطقی t_l، احتمال بروز خطای صفحه p و زمان پاسخ دادن به خطای صفحه (انتقال صفحه به حافظه) t_p باشد، زمان مؤثر دسترسی به هر آدرس مجازی از رابطه‌ی زیر بدست می‌آید.

$$(۱ - p)t_l + pt_p$$

مسائل مهم در صفحه‌بندی مبتنی بر نیاز

دو مسئله‌ی مهم در صفحه‌بندی مبتنی بر نیاز وجود دارد:

- الگوریتم تخصیص قاب (Frame Allocation): قاب‌ها چگونه بین پردازه‌ها تقسیم شوند. در هر لحظه تعدادی پردازه در سیستم عامل اجرا می‌شوند و سیستم عامل باید مشخص کند به هر پردازه چند قاب تخصیص یابد.
- الگوریتم جایگزینی صفحه (Page Replacement): اگر قاب خالی در حافظه‌ی اصلی موجود نباشد، چه قابی به حافظه‌ی کمکی انتقال یابد تا یک قاب خالی ایجاد شود.

به هر یک از این دو مسئله خواهیم پرداخت.

الگوریتم‌های متفاوتی برای تخصیص قاب وجود دارند. چند نمونه را بررسی می‌کنیم.

- در تخصیص برابر (Equal Allocation)، قاب‌های حافظه‌ی اصلی به صورت مساوی بین پردازه‌ها تقسیم می‌شود. برای مثال، اگر چهار پردازه موجود باشند و تعداد قاب‌ها صد عدد باشد، به هر پردازه بیست و پنج قاب تخصیص داده می‌شود.
- تخصیص نسبی (Proportional Allocation) تعداد قاب‌های تخصیص داده شده به هر پردازه با توجه به اندازه‌ی حافظه‌ی مجازی آن تعیین می‌شود. برای نمونه فرض کنید فقط دو پردازه در سیستم عامل موجود باشند: حافظه‌ی مجازی پردازه‌ی A شامل نود صفحه و حافظه‌ی مجازی پردازه‌ی B شامل سی صفحه باشد. اگر صد قاب در سیستم عامل موجود باشد، هفتاد و پنج عدد از این قاب‌ها ($\frac{۹۰}{۱۲۰} \times ۱۰۰$) به پردازه‌ی A و بیست و پنج عدد از آنها ($\frac{۳۰}{۱۲۰} \times ۱۰۰$) به پردازه‌ی B تخصیص می‌یابند.
- تخصیص قاب با توجه به نرخ خطای صفحه (Page Fault Frequency): اگر تعداد قاب‌هایی که به یک پردازه تخصیص یافته است کم باشد، پردازه با نرخ بالا دچار خطای صفحه می‌شود تا صفحاتی از حافظه‌ی کمکی جایگزین صفحات موجود در حافظه‌ی اصلی شوند. در صفحه‌ی بعدی بیشتر توضیح داده می‌شود.

تخصیص قاب با توجه به نرخ خطای صفحه

- نرخ بالای خطای صفحه‌ی یک پردازه نشان می‌دهد که چون تعداد قاب‌های پردازه کافی نیست، پردازه مجبور است برای دسترسی به صفحات، تعداد از صفحه‌های موجود در حافظه را جایگزین کند.
- اگر نرخ خطای صفحه‌ی یک پردازه پایین باشد، نشان می‌دهد تعداد قاب‌های تخصیص داده شده به پردازه کافی یا زیاد است.
- سیستم عامل می‌تواند با توجه به نرخ خطای صفحه، قاب‌ها را بین پردازه‌ها تقسیم کند.
- قاب‌های بیشتری را به پردازه‌هایی که خطای صفحه‌ی آنها زیاد است تخصیص دهد و قاب‌های تخصیص یافته به پردازه‌هایی که به ندرت دچار خطای صفحه می‌شوند را کاهش دهد.

- اگر پردازه‌ای برای دسترسی به صفحه‌ای که در حافظه‌ی کمکی قرار دارد دچار خطای صفحه شود، سیستم عامل باید یک قاب خالی در حافظه‌ی اصلی تخصیص دهد.
- اگر همه‌ی قاب‌ها پر باشند، سیستم عامل مجبور است یکی از قاب‌هایی که در حافظه‌ی اصلی موجود هستند را انتخاب کند و به حافظه‌ی کمکی انتقال دهد؛ به این کار جایگزینی صفحه (Page Replacement) گفته می‌شود.

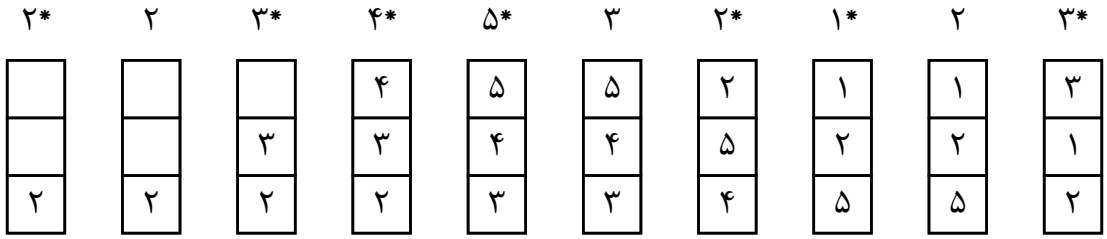
الگوریتم‌های جایگزینی صفحه در دو دسته قرار می‌گیرند:

- در جایگزینی محلی (Local Replacement) یکی از صفحه‌های خود پردازه‌ای که دچار خطای صفحه شده است جایگزین می‌شود.
- در جایگزینی سراسری (Global Replacement) یکی از صفحه‌های موجود در حافظه جایگزین می‌شود که ممکن است به پردازه‌ای که دچار خطای صفحه شده است تعلق نداشته باشد.

در ادامه چند الگوریتم برای جایگزینی صفحه را مطالعه می‌کنیم.

الگوریتم FIFO

- در الگوریتم FIFO (First In First Out) صفحه‌ای جایگزین می‌شود که زودتر از همه به حافظه انتقال داده شده است.
- برای نمونه، فرض کنید پردازه‌ای به ترتیب به صفحه‌های شماره‌ی ۲، ۲، ۳، ۴، ۵، ۳، ۲، ۱، ۲، ۳ دسترسی داشته باشد.
- شکل زیر، عملکرد الگوریتم جایگزینی صفحه‌ی FIFO را برای این دسترسی‌ها با فرض اینکه سه قاب موجود باشند نشان می‌دهد.



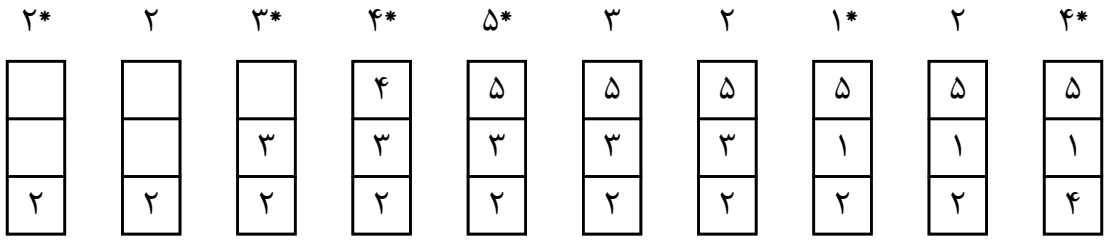
- خطاهای صفحه با علامت ستاره مشخص شده‌اند. برای این دسترسی‌ها هفت خطا صفحه رخ می‌دهد.

- فرض کنید پردازه‌ای به ترتیب به صفحه‌هایی با شماره‌ی شکل زیر به حافظه دسترسی داشته باشد.

1	2	3	4	1	2	5	1	2	3	4	5

- اگر در الگوریتم FIFO سه قاب موجود باشند نه خطای صفحه و اگر چهار قاب موجود باشند ده خطای صفحه ایجاد می‌شود.
- یعنی ممکن است تعداد قاب‌های اختصاص یافته به یک پردازه افزایش یابد ولی تعداد خطاها نیز افزایش یابد (ما انتظار داشتیم هر چه تعداد قاب‌ها بیشتر باشد، احتیاج به جایگزینی صفحات و خطای صفحه کمتر باشد).
- به این اتفاق ناهنجاری Belady (Belady Anomaly) گفته می‌شود.
- ناهنجاری Belady فقط در برخی از الگوریتم‌های جایگزینی صفحه مثل FIFO رخ می‌دهد.

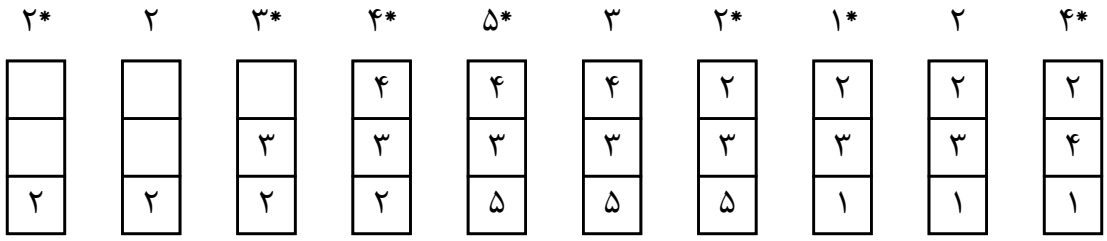
- الگوریتم بهینه (Optimal) صفحه‌ای را جایگزین می‌کند که در آینده به آن دیرتر از سایر صفحه‌ها دسترسی انجام می‌شود.
- مثال زیر را در نظر بگیرید.



- در این مثال شش خطای صفحه رخ می‌دهد.
- می‌توان اثبات کرد که تعداد خطای صفحه در الگوریتم بهینه کمینه است.
- بدی الگوریتم بهینه این است که لازم است از دسترسی‌های آینده مطلع باشد؛ ولی در عمل معمولا سیستم عامل نمی‌داند یک پردازه در آینده به چه صفحاتی دسترسی خواهد داشت.

الگوریتم LRU

- الگوریتم LRU (Least Recently Used) صفحه‌ای را جایگزین می‌کند که آخرین دسترسی به آن قبل از سایر صفحه‌های موجود در حافظه است.
- ایده‌ی الگوریتم LRU این است که به جای اینکه مثل الگوریتم بهینه از دسترسی‌های آینده استفاده کنیم، دسترسی‌های گذشته را در نظر می‌گیریم.
- مثال زیر را در نظر بگیرید.

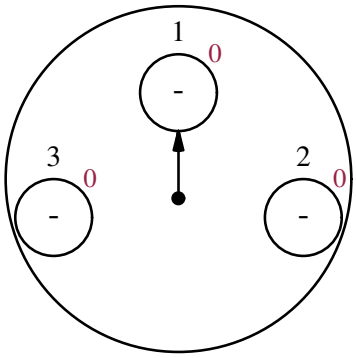


- در این مثال هفت خطای صفحه رخ می‌دهد.
- اگر چه الگوریتم LRU عملکرد خوبی در عمل دارد، اما پیاده‌سازی آن سخت است چون در MMU در هنگام هر دسترسی به حافظه باید زمان دسترسی به آن در قسمتی از حافظه (مثلا در ستونی از جدول صفحه) نوشته شود. در زمان جایگزینی نیز باید صفحه‌ای جایگزین شود که زمان آخرین دسترسی به آن از بقیه کمتر است.
- به این دلیل، از الگوریتم‌هایی ارائه شده‌اند که LRU را تقریب می‌زنند.

الگوریتم ساعت عقربه‌ای

الگوریتم ساعت عقربه‌ای (Clock) یا شانس مجدد (Second Chance) یکی از الگوریتم‌هایی است که برای تقریب LRU استفاده می‌شود.

- این الگوریتم از بیت Reference در جدول صفحه استفاده می‌کند. یادآوری می‌کنم که مقدار این بیت در هنگام دسترسی به یک صفحه به یک تغییر می‌کند.
- در این الگوریتم فرض کنید قاب‌ها در دایره‌ای قرار گرفته‌اند و یک عقربه به یکی از این قاب‌ها اشاره می‌کند. در شکل زیر عقربه به قاب شماره‌ی یک اشاره می‌کند و بیت Reference هر صفحه با رنگ قرمز در کنار هر قاب نمایش داده می‌شود.



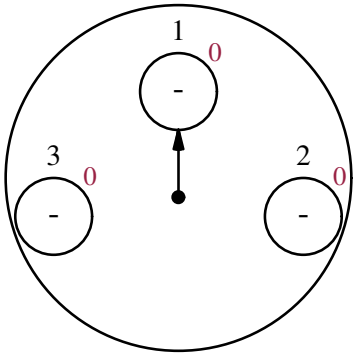
- در ابتدا فرض کنید همه‌ی قاب‌ها خالی باشند.

در الگوریتم ساعت عقربه‌ای اگر خطای صفحه رخ داد به سراغ قابی می‌رویم که عقربه به آن اشاره می‌کند. دو حالت وجود دارد:

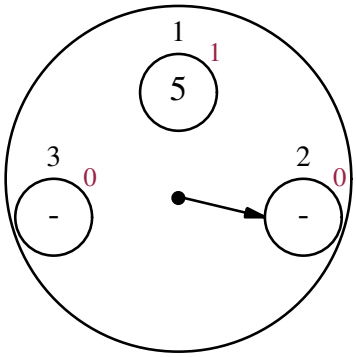
- حالت اول: اگر بیت Reference مربوط به آن صفر باشد، آن را جایگزین می‌کنیم و عقربه را به قاب بعدی اشاره می‌دهیم.
- حالت دوم: اگر بیت Reference یک باشد، آن را صفر می‌کنیم، عقربه را به قاب بعدی اشاره می‌دهیم و برای آن قاب تکرار می‌کنیم و دو حالت را در نظر می‌گیریم.

در واقع در حالت دوم به صفحه شانس دیگری داده می‌شود چون اخیرا به آن دسترسی انجام شده است و مقدار بیت Reference آن یک شده است؛ ما به دنبال صفحه‌ای هستیم که اخیرا به آن دسترسی انجام نشده است.

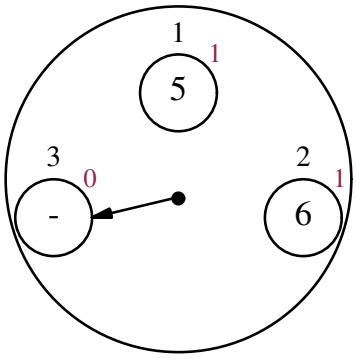
- در شروع همه‌ی قاب‌ها خالی هستند.



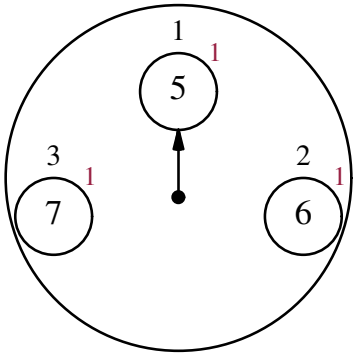
- فرض کنید به صفحه‌ی شماره‌ی پنج دسترسی انجام شود؛ چون عقربه به قابی اشاره می‌کند که شماره‌ی بیت Reference آن صفر است، صفحه‌ی شماره‌ی پنج را در قاب شماره‌ی یک قرار می‌دهیم.



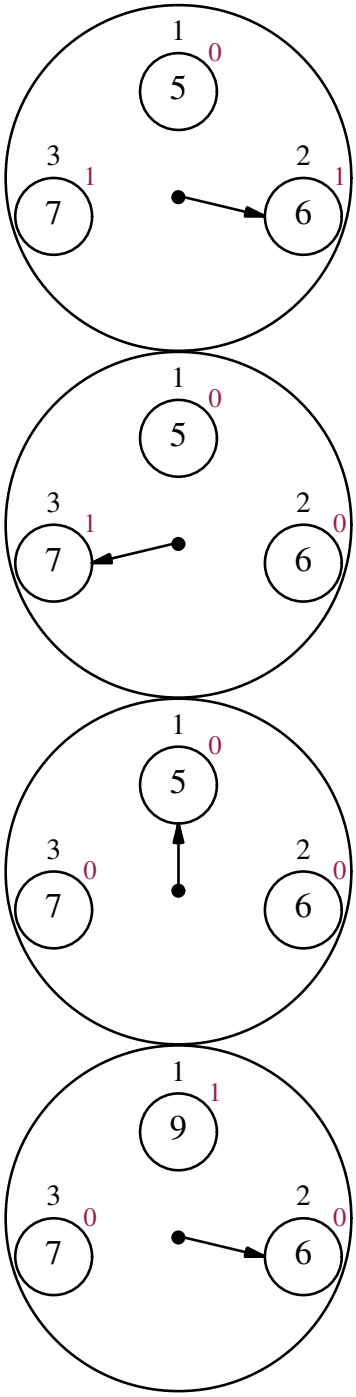
- پس از آن به صفحه‌ی شماره‌ی شش دسترسی انجام می‌شود.



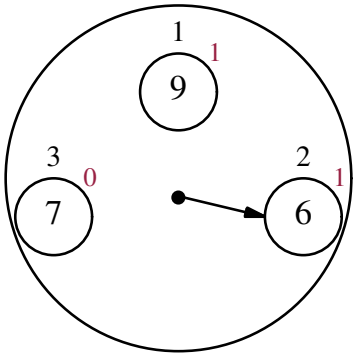
- سپس به صفحه‌ی شماره‌ی هفت دسترسی انجام می‌شود.



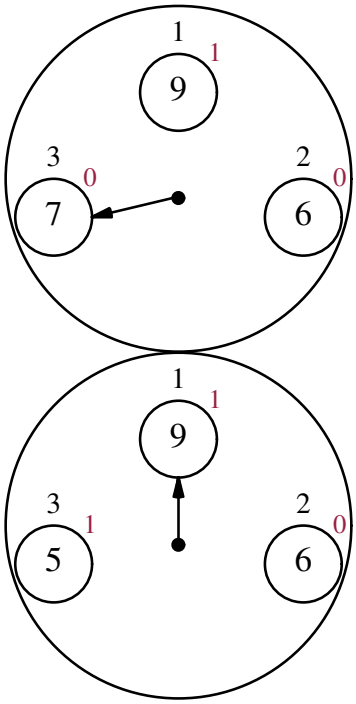
- سپس به صفحه‌ی شماره‌ی نه دسترسی انجام می‌شود؛ پس از این دسترسی مراحل زیر انجام می‌شود.



- سپس فرض کنید دسترسی به صفحه‌ی شماره‌ی شش انجام شود. چون این صفحه در حافظه موجود است، مکان عقربه تغییر نمی‌کند و فقط بیت Reference صفحه به یک تغییر می‌کند.



- سپس دسترسی به صفحه‌ی شماره‌ی پنج انجام شود.



در این قسمت موضوعات جالبی را در مورد حافظه‌ی مجازی بررسی می‌کنیم.

پردازه‌ای دچار کوبیدگی (Thrashing) می‌شود که بیشتر وقتش را صرف جایگزینی صفحه کند.

- دلیل رخداد کوبیدگی کافی نبودن تعداد قاب‌ها است که نرخ بالای خطای صفحه را موجب می‌شود.
- کوبیدگی اتفاق بدی است چون به جای اینکه پردازه‌ها توسط پردازنده اجرا شوند، وقت پردازه‌ها برای انتقال صفحه‌ها بین حافظه‌ی اصلی و حافظه‌ی کمکی صرف می‌شود.
- برای اینکه مشکل کوبیدگی را بهتر درک کنید، فرض کنید پردازه‌ای به صورت چرخشی به پنج صفحه دسترسی دارد ولی فقط سه قاب در اختیارش باشد. چه اتفاقی می‌افتد؟
- در شرایط کوبیدگی اگر سیستم عامل با توجه به اینکه پردازنده زمان زیادی بیکار است و بیشتر پردازه‌ها منتظر عملیات ورودی و خروجی هستند (انتقال صفحات)، درجه‌ی چند برنامگی را افزایش دهد، وضعیت بدتر می‌شود (چرا؟).

روش‌های متفاوتی برای مقابله با کوبیدگی وجود دارد.

- یکی این است که برخی از پردازه‌ها را کاملا به حافظه‌ی کمکی انتقال دهیم تا حافظه‌ی بیشتری در اختیار سایر پردازه‌ها قرار بگیرد. پس از اینکه مشکل برطرف شد، پردازه‌های انتقال یافته را ادامه دهیم.
- بهتر است در کوبیدگی از یک الگوریتم جایگزینی صفحه‌ی محلی استفاده شود تا مشکل کوبیدگی برای سایر پردازه‌ها نیز بوجود نیاید.
- روش دیگر استفاده از مدل مجموعه‌ی کاری هست.

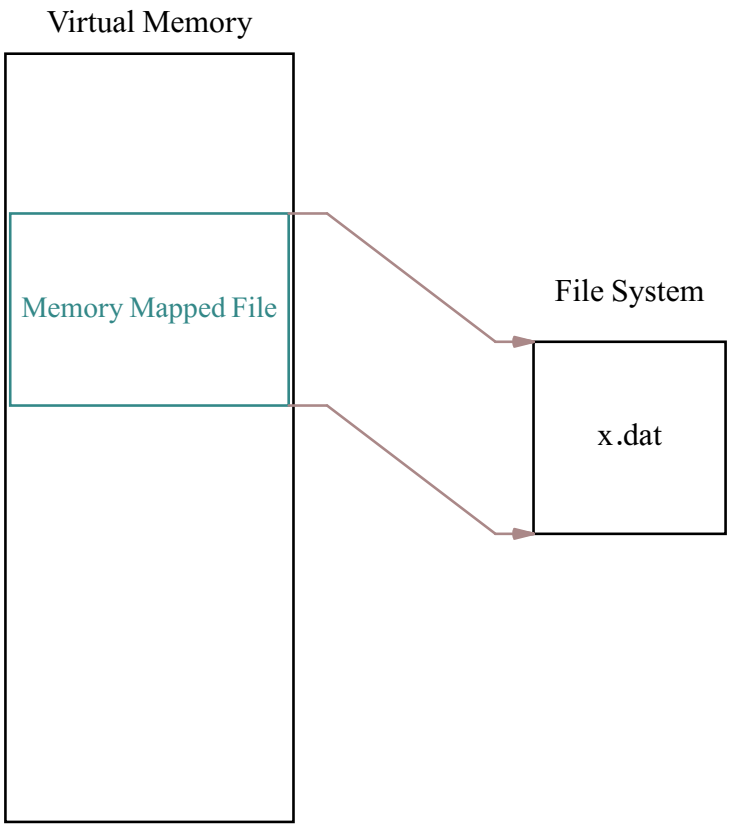
این روش بر پایه‌ی اصل محلی‌گرایی (Locality) برنامه‌ها است یعنی در هر لحظه بخشی از حافظه هست که برنامه‌ها به آن با فراوانی بالا دسترسی دارند.

به مجموعه‌ی همه‌ی صفحه‌هایی که در بازه‌ی کوتاه اخیری (مثلا ده ثانیه) پردازه‌ای به آنها دسترسی داشته است مجموعه‌ی کاری (Working Set) آن پردازه گفته می‌شود.

اگر مجموع اندازه‌ی مجموعه‌ی کاری پردازه‌ها از تعداد کل قاب‌های حافظه‌ی اصلی بیشتر باشد، کوبیدگی رخ می‌دهد و باید تعدادی از پردازه‌ها را به حافظه‌ی کمکی انتقال داد.

با کمک حافظه‌ی مجازی می‌توان محتویات فایل‌ها را به حافظه نگاشت کرد. به این فایل‌ها، فایل‌های نگاشت شده به حافظه (Memory-mapped Files) گفته می‌شود.

- خواندن از قسمت‌هایی از حافظه که فایلی به آن نگاشت شده است موجب می‌شود محتویات فایل خوانده شود.
- برای نمونه اگر فایلی از آدرس **800** تا **900** حافظه نگاشت شده باشد، خواندن از آدرس **808** مثل این است که بایت هشتم فایل خوانده شود (به صورت مشابه، نوشتن به این آدرس موجب می‌شود بایت هشتم از فایل تغییر کند).



- عملکرد فایل‌های نگاشت شده به حافظه مشابه صفحه‌بندی مبتنی بر نیاز است با این تفاوت که به جای اینکه صفحه‌ها از فضای Swap به حافظه انتقال یابند، محتویات فایل از دیسک به حافظه انتقال می‌یابد.

تکثیر در هنگام نوشتن

یکی از عملیات پر تکرار در سیستم عامل fork() هست که پردازه‌ی جدیدی را ایجاد می‌کند که محتویات حافظه‌ی آن کاملا مشابه پردازه‌ی پدر است.

- اگر اندازه‌ی حافظه‌ی مجازی پردازه‌ای که fork() را صدا می‌زند زیاد باشد، حجم زیادی از حافظه باید برای پردازه‌ی فرزند تکثیر شود.
- در روش تکثیر در حال نوشتن (Copy-on-Write) جدول صفحه‌ی پردازه‌ی جدید به همان صفحات پردازه‌ی پدر اشاره می‌کند.
- این کار با استفاده از پرچم Read-only در جدول صفحه انجام می‌شود.

	Frame	Valid	Read-only
0	3	1	0
1	0	1	0
2		0	0
3	1	1	0

- بعد از fork()

	Frame	Valid	Read-only
0	3	1	1
1	0	1	1
2		0	0
3	1	1	1

	Frame	Valid	Read-only
0	3	1	1
1	0	1	1
2		0	0
3	1	1	1

- اما هنگامی که یکی از دو پردازه بخواهد صفحه‌ای را تغییر دهد، سیستم عامل یک کپی از آن صفحه می‌گیرد و نسخه‌ی کپی شده را تغییر می‌دهد که حافظه‌ی پردازه‌ی دیگر تغییر نکند.