موسسه بابان

انتشارات بابان و انتشارات راهیان ارشد

درس و کنکور ارشد

سیستم عامل

(مدیریت حافظه مجازی)

ویژه‌ی داوطلبان کنکور کارشناسی ارشد مهندسی کامپیوتر و IT

براساس کتب مرجع

آبراهام سیلبرشاتز، ویلیام استالینگز و اندرو اس تنن‌بام

ارسطو خلیلی‌فر

۵ حافظه مجازی

حافظه مجازی (Virtual Memory)

در همه روش‌های پیشینِ مدیریت حافظه، یک فرآیند تنها زمانی می‌توانست اجرا گردد که تمام فرآیند بتواند در آن واحد در حافظه حاضر باشد. در این حالت اگر فرآیند قدری بزرگ باشد، پیدا کردن فضای خالی مناسب برای آن مشکل خواهد بود.

ایده اصلی حافظه مجازی این است که حتی اگر به اندازه کافی فضای خالی بر روی حافظه در اختیار نداشته باشیم، به یک فرآیند اجازه اجرا بدهیم. نکته اصلی این است که یک فرآیند در آنِ واحد به همه داده‌ها و کد خود نیاز ندارد بلکه در هر لحظه فقط به بخشی از داده و قسمتی از کد نیازمند است. در تکنیک حافظه مجازی فقط قسمتی از فرآیند به حافظه آورده می‌شود که در حال حاضر به آن نیاز است و مابقی فرآیند می‌تواند کماکان بر روی دیسک قرار گیرد و در طول اجرای یک فرآیند این جابه‌جایی‌ها بین حافظه و دیسک مرتباً صورت گیرد. با این کار می‌توانیم فرآیندهای بیشتری را در داخل حافظه‌ی اصلی نگهداری کنیم.

اگر این تکنیک از دید فرآیند مخفی بماند، فرآیند گمان می‌کند تمام فضای درخواستی وی به او تخصیص داده شده است، در صورتی که چنین نیست و در واقع سیستم عامل با تلاشی مضاعف و با انجام جابه‌جایی‌های پی در پی این دید را برای فرآیند ایجاد کرده است.

نکته: با توجه به مسائل مطرح شده، مجموع کلیه فضای آدرس فرآیندهایی که در حال اجرا هستند، می‌تواند از اندازه حافظه فیزیکی بیشتر شود و **این یعنی حافظه مجازی**. در واقع همه فرآیندها گمان

می‌کنند به طور کامل در حافظه قرار دارند، غافل از اینکه قسمت اعظم هر یک از آنها بر روی دیسک است.

نکته: ایده‌ی حافظه‌ی مجازی معمولاً با تکنیک صفحه‌بندی راحت‌تر پیاده‌سازی می‌شود که به آن صفحه‌بندی برحسب نیاز (Demand Paging) گویند. البته حافظه مجازی را می‌توان با روش‌های دیگری، مانند قطعه‌بندی نیز پیاده‌سازی کرد، اما روال کار پیچیده‌تر می‌شود.

۴ ـ ۱ ـ ۱ ـ صفحه‌بندی برحسب نیاز (Demand Paging)

یکی از بهترین روش‌ها جهت پیاده‌سازی ایده حافظه مجازی، استفاده از تکنیک صفحه‌بندی است. در این حالت تنها تعداد اندکی از صفحه‌های یک فرآیند به حافظه آورده و مابقیِ صفحات بر روی دیسک نگهداری می‌شوند. در این شیوه اگر به صفحات موجود بر روی دیسک نیاز پیدا کردیم، آن صفحات به جای صفحات قدیمی به حافظه آورده شده و در عوض صفحات قدیمی به دیسک منتقل می‌شوند.

نکته: در این روش باید مشخص شود کدام صفحات در حافظه و کدام صفحات بر روی دیسک قرار دارند. برای این منظور روش‌های مختلفی وجود دارد اما عموماً از یک بیت در جدول صفحه استفاده می‌کنند. این بیت که آن را **بیت اعتبار** می‌نامیم، مشخص می‌کند صفحه موردنظر در حافظه قرار دارد یا خیر. برای مثال اگر مقدار این بیت به ازای یک درایه در جدول صفحه یک بود، به این معناست که صفحه موردنظر معتبر (valid) است و در حافظه قرار دارد، اما اگر این بیت صفر بود به این معناست که صفحه موردنظر نامعتبر (invalid) است و بر روی دیسک قرار دارد.

نکته: اگر فرآیندی به یکی از صفحاتش که در حافظه موجود نیست (بیت اعتبار آن با مقدار نامعتبر پر شده است) نیاز داشته باشد، یک وقفه خطای صفحه (Page Fault) رخ می‌دهد که سیستم عامل باید برای این صفحه، یک قاب در حافظه تهیه کرده و آن را به حافظه منتقل کند.

نکته: آدرسی که توسط فرایند مورد ارجاع قرار می‌گیرد، **آدرس مجازی** نام دارد و آدرس‌های حافظه اصلی را **آدرس‌های حقیقی** گویند. با این تعریف محدوده آدرس‌های مجازی که یک فرآیند می‌تواند به آنها ارجاع کند، **فضای آدرس مجازی** نام دارد و محدوده آدرس‌های حقیقی موجود در یک سیستم را **فضای آدرس حقیقی** آن کامپیوتر گویند. هنگامی که فرآیندی در حال اجراست، آدرس‌های مجازی باید به آدرس‌های حقیقی تبدیل شوند.

نکته: هنگامی که یک صفحه با وقفه نقص صفحه مواجه شد، سیستم عامل باید هر چه سریع‌تر آن را به یکی از قاب‌های حافظه منتقل کند. در این حالت اگر هیچ قاب آزادی در حافظه موجود نباشد، یکی از صفحات باید به دیسک منتقل شود تا یک قاب حافظه برای صفحه جدید آزاد گردد. با این شرایط، چگونگی انتخاب یک صفحه برای ترک حافظه بسیار مهم است و تأثیر مستقیمی بر کارآیی و تعداد وقفه‌های نقص صفحه در آینده دارد.

الگوریتم‌های جایگزینی صفحه (Page Replacement)

هنگامی که یک وقفه نقص صفحه رخ می‌دهد سیستم عامل باید یکی از صفحات را از حافظه بیرون ببرد تا جا برای صفحه جدید باز شود. اگر صفحه قدیمی در مدت زمانی که در حافظه بوده، تغییر کرده باشد، باید محتویات آن در دیسک نوشته شود تا تغییرات از دست نرود، اما اگر تغییر نکرده باشد، کپی موجود بر روی دیسک همچنان معتبر است و نیازی به نوشتن محتویات صفحه بر روی دیسک نیست و صفحه جدید به سادگی بر روی صفحه قدیمی نوشته می‌شود.

هنگامی که نقص صفحه رخ می‌دهد، می‌توان هر صفحه‌ای را برای خروج از حافظه انتخاب کرد، اما برای مثال اگر صفحه‌ای انتخاب شود که زیاد مورد استفاده قرار می‌گیرد به احتمال زیاد کمی بعد باید دوباره آن را به حافظه برگردانیم و این یعنی تصمیم اشتباه.

برای جایگزینی صفحه الگوریتم‌های زیادی وجود دارند که ایده‌ها و کارآیی متفاوتی دارند. تعدادی از آنها را بررسی می‌کنیم:

الگوریتم FIFO (First In First Out)

این الگوریتم ساده‌ترین الگوریتم از نظر پیاده‌سازی است. در این روش سیستم عامل لیستی از صفحات را به ترتیب ورود به حافظه نگه می‌دارد. وقتی یک خطای نقص صفحه رخ می‌دهد، سیستم عامل **قدیمی‌ترین** صفحه را برای بیرون رفتن انتخاب می‌کند. ایده این روش این است که قدیمی‌ترین صفحه شانس مورد استفاده قرار گرفتن را به اندازه کافی در اختیار داشته و اکنون باید این شناس به صفحه دیگری داده شود.

نکته: نقص الگوریتم FIFO این است که حتی اگر صفحه‌ای بارها و به طور مکرر استفاده شود، سرانجام به قدیمی‌ترین صفحه تبدیل و حذف می‌شود، در صورتی که احتمالاً بلافاصله باید دوباره به حافظه آورده شود.

مثال: فرض کنید در سیستمی ۳ قاب حافظه وجود دارد، اگر درخواست‌های زیر از چپ به راست، به این سیستم وارد شود، چند وقفه نقص صفحه رخ می‌دهد؟ ۵، ۱، ۲، ۳، ۴، ۵، ۳، ۴، ۱، ۲، ۳، ۴

حل: فرض می‌کنیم در ابتدا هر ۳ قاب خالی هستند، با جدول زیر تعداد نقص صفحه را به دست می‌آوریم:

ورودی	۴	۳	۲	۱	۴	۳	۵	۴	۳	۲	۱	۵
قاب ۱	۴	۴	۴	۱	۱	۱	۵	۵	۵	۵	۵	۵
قاب ۲		۳	۳	۳	۴	۴	۴	۴	۴	۲	۲	۲
قاب ۳			۲	۲	۲	۳	۳	۳	۳	۳	۱	۱
وقفه خطای صفحه	×	×	×	×	×	×	×			×	×	

جمعاً ۹ وقفه خطای صفحه رخ می‌دهد.

نکته: در نگاه اول به نظر می‌رسد با افزایش قاب‌هایی از حافظه که در اختیار یک فرآیند است، تعداد نقص صفحه‌ها همواره کاهش می‌یابد، اما در الگوریتم FIFO و در بعضی الگوهای خاصِ ارجاع به صفحه‌ها، با افزایش تعداد قاب‌ها، تعداد وقفه‌های نقص صفحه نیز افزایش می‌یابد. این پدیده را **ناهنجاری FIFO** (FIFO Anomaly) و یا **ناهنجاری بی‌لیدی** (Belady Anomaly) گویند.

برای روشن شدن قضیه، همان مثال قبل را این بار با ۴ قاب حافظه بررسی می‌کنیم:

ورودی	۴	۳	۲	۱	۴	۳	۵	۴	۳	۲	۱	۵
قاب ۱	۴	۴	۴	۴	۴	۴	۵	۵	۵	۵	۱	۱
قاب ۲		۳	۳	۳	۳	۳	۳	۴	۴	۴	۴	۵
قاب ۳			۲	۲	۲	۲	۲	۲	۳	۳	۳	۳
قاب ۴				۱	۱	۱	۱	۱	۱	۲	۲	۲
وقفه خطای صفحه	×	×	×	×			×	×	×	×	×	×

مشاهده می‌کنیم با همان دنباله ارجاع و با ۴ قاب، تعداد نقص صفحه‌ای که رخ می‌دهد به ۱۰ می‌رسد. البته اگر تعداد قاب‌ها را به ۵ افزایش دهیم تعداد نقص صفحه در این مثال یکباره به ۵ نقص صفحه کاهش می‌یابد.

الگوریتم بهینه (Optimal)

اساس کار این الگوریتم کاملاً منطقی است. در این الگوریتم صفحه‌ای باید برای ترک حافظه انتخاب شود که در آینده، دیرتر از همه به آن نیاز پیدا می‌کنیم. به عنوان مثال از بین دو صفحه A و B، صفحه A را تا ۲ میلیون دستور دیگر و صفحه B را تا ۳ میلیون دستور دیگر نیاز نداریم. بنابراین کاملاً منطقی است که صفحه B را از حافظه خارج کنیم تا خطاهای نقص صفحه را تا حد امکان به تأخیر بیندازیم.

نکته: الگوریتم **بهینه** ناهنجاری بی‌لیدی **ندارد**.

نکته: با اجرای این الگوریتم کمترین تعداد خطای نقص صفحه رخ می‌دهد.

مثال: حافظه‌ای را با ۳ قاب در نظر بگیرید. با استفاده از روش بهینه و برای دنباله ارجاعات زیر چند خطای نقص صفحه رخ می‌دهد؟ ۱، ۴، ۳، ۵، ۲، ۱، ۳، ۴، ۱، ۲، ۳

ورودی	۳	۲	۱	۴	۳	۱	۲	۵	۳	۴	۱
قاب ۱	۳	۳	۳	۳	۳	۳	۳	۳	۳	۳	۱
قاب ۲		۲	۲	۴	۴	۴	۴	۴	۴	۴	۴
قاب ۳			۱	۱	۱	۱	۲	۵	۵	۵	۵
نقص صفحه	×	×	×	×			×	×			×

در این مثال جمعاً ۷ خطای نقص صفحه رخ می‌دهد.

نکته: همان‌گونه که ذکر شد الگوریتم بهینه کمترین تعداد نقص صفحه ممکن را باعث می‌شود، اما نقص عمده آن این است که در عمل قابل پیاده‌سازی نیست. زیرا سیستم عامل باید بداند در آینده چه صفحه‌هایی و به چه ترتیبی موردنیاز هستند که این مسئله در عمل غیرممکن است.

نکته: از الگوریتم بهینه فقط می‌توان برای ارزیابی دیگر الگوریتم‌ها استفاده کرد. از آنجا که این الگوریتم کمترین خطای نقص صفحه را باعث می‌شود، الگوریتم‌های قابل پیاده‌سازی را با این الگوریتم مقایسه می‌کنند.

الگوریتم LRU (Last Recently Used)

ایده اصلی این الگوریتم این است که اگر صفحه‌ای در چند دستور اخیر مراجعات زیادی داشته است، به احتمال قوی در دستورات بعدی هم ارجاعات زیادی خواهد داشت، همچنین اگر یک صفحه، اخیراً هیچ مراجعه‌ای نداشته، احتمالاً در آینده نزدیک هم ارجاعی نخواهد داشت.

در واقع این الگوریتم بیان می‌کند هنگام وقوع خطای نقص صفحه، صفحه‌ای را حذف کنید که طولانی‌ترین زمان عدم استفاده را دارد.

می‌توان گفت LRU تقریبی از الگوریتم بهینه می‌باشد که در آن به جای توجه به آینده، به گذشته توجه می‌شود.

نکته: الگوریتم **LRU** ناهنجاری بی‌لیدی **ندارد**.

مثال: حافظه‌ای را با ۳ قاب صفحه در نظر بگیرید، با استفاده از روش LRU **و برای دنباله ارجاعات زیر، چند خطای نقص صفحه رخ می‌دهد؟** ۲، ۳، ۱، ۵، ۳، ۱، ۴، ۲، ۴، ۳، ۴، ۲، ۱، ۳

ورودی	۳	۱	۲	۴	۳	۴	۲	۴	۱	۳	۵	۱	۳	۲
قاب ۱	۳	۳	۳	۴	۴	۴	۴	۴	۴	۴	۵	۵	۵	۲
قاب ۲		۱	۱	۱	۳	۳	۳	۳	۱	۱	۱	۱	۱	۱
قاب ۳			۲	۲	۲	۲	۲	۲	۲	۳	۳	۳	۳	۳
نقص صفحه	×	×	×	×	×				×	×	×			×

جمعاً ۹ خطای صفحه رخ می‌دهد.

نکته: الگوریتم LRU در عمل قابل پیاده‌سازی می‌باشد اما هزینه پیاده‌سازی آن قدری بالاست. در این روش به یک لیست از تمامی صفحات حافظه نیاز داریم که در آن صفحات به ترتیبِ ارجاعات اخیر آنها مرتب شده باشند، در واقع این لیست باید در هر ارجاع به حافظه به روز شود. در عمل الگوریتم LRU را جهت افزایش سرعت، به صورت سخت‌افزاری پیاده‌سازی می‌کنند. برای پیاده‌سازی LRU

روش‌های مختلفی پیشنهاد شده است. مانند پشته، استفاده از شمارنده و نوعی ماتریس دوبعدی. برای بررسی این روش‌ها می‌توان به کتاب پروفسور تنن‌باوم رجوع کرد.

الگوریتم دومین شانس (Second Chance)

این الگوریتم از خانواده الگوریتم FIFO می‌باشد. این ایده موجب می‌شود در الگوریتم FIFO، صفحاتی که زیاد استفاده شده‌اند از حافظه خارج نشوند. در این الگوریتم به ازای هر صفحه در جدول صفحه، یک بیت با عنوان R در نظر می‌گیریم (حرف R از ابتدای کلمه Referenced گرفته شده است). اگر به یک صفحه موجود در حافظه ارجاع شود، بیت R آن صفحه یک می‌شود.

در این روش برای حذف، باز هم به سراغ قدیمی‌ترین صفحه می‌رویم (همانند FIFO)، اما قبل از حذف قدیمی‌ترین صفحه، بیت R آن را چک می‌کنیم، اگر این بیت صفر باشد به این معناست که این صفحه هم قدیمی است و هم ارجاعی به آن صورت نگرفته است، بنابراین با خیالی آسوده آن را حذف می‌کنیم. اما اگر بیت R آن یک باشد، سیستم عامل بیت R آن را صفر کرده و این صفحه را به انتهای صف می‌فرستد. در واقع با این کار به این صفحه یک شانس دیگر داده می‌شود تا در حافظه باقی بماند.

نکته: الگوریتم **دومین شانس**، ناهنجاری بی‌لیدی **دارد**.

نکته: اگر هنگام یک نقص صفحه، بیت R مربوط به همه صفحات یک باشد، این الگوریتم همانند FIFO عمل می‌کند (البته با قدری اتلاف وقت) زیرا در حالتی که بیت R همه صفحه‌ها یک باشد، این الگوریتم تمام صفحه‌ها را به انتهای لیست برده و بیت R آنها را صفر می‌کند و دوباره قدیمی‌ترین صفحه در ابتدای لیست قرار می‌گیرد اما این بار با بیت R برابر صفر.

نکته: دقت کنید در الگوریتم دومین شانس، اگر صفحه‌ای زیاد مورد ارجاع واقع شود، این شانس را دارد که در حافظه باقی بماند. به عنوان مثال فرض کنید یک صفحه با بیت R=۱ به ابتدای صف برسد، در این حالت سیستم عامل بیت R آن را صفر کرده و این صفحه را به ابتدای صف منتقل می‌کند، حال اگر این صفحه طی مدت زمانی که دوباره به ابتدای صف نزدیک می‌شود باز هم ارجاع شود، بیت R آن مجدداً یک می‌شود و باز هم در حافظه باقی می‌ماند.

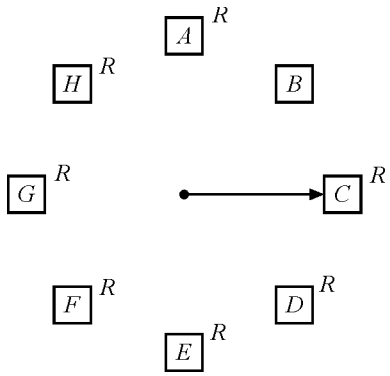
الگوریتم ساعت (Clock)

الگوریتم دومین شانس الگوریتم خوبی است اما پیوسته در حال جابه‌جا کردن صفحات در لیست است، این کار قدری کارآیی آن را کاهش می‌دهد.

یک ایده برای بهبود پیاده‌سازی الگوریتم دومین شانس، این است که ابتدا و انتهای لیست صفحات را به هم وصل کنیم. در واقع از یک لیست پیوندی حلقوی استفاده می‌کنیم که مانند یک ساعت عقربه‌ای عمل می‌کند. در این ساعت، عقربه همواره قدیمی‌ترین صفحه را نشان می‌دهد. هنگامی که یک خطای نقص صفحه رخ می‌دهد، سیستم عامل صفحه‌ای را که عقربه ساعت بر روی آن است بررسی می‌کند،

اگر بیت R آن صفر باشد، این صفحه حذف و عقربه روی صفحه بعدی برده می‌شود. اما اگر بیت R آن یک باشد، سیستم عامل فقط بیت R را صفر کرده و عقربه را یک صفحه جلو می‌برد.

نکته: دقت کنید الگوریتم ساعت همان الگوریتم دومین شانس است. در واقع می‌توان گفت یکی از راه‌های پیاده‌سازی الگوریتم دومین شانس، ایده ساعت می‌باشد.



مفهوم الگوریتم ساعت

مثال: حافظه‌ای با ۳ قاب آزاد را در نظر بگیرید، با استفاده از الگوریتم دومین شانس (ساعت) و با دنباله ارجاعات زیر چند نقص صفحه رخ می‌دهد؟

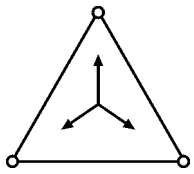
دنباله ارجاعات از چپ به راست: ۲، ۵، ۳، ۴، ۳، ۵، ۲، ۶، ۵، ۳، ۴، ۳، ۲، ۱

حل: فرض کنید اندیس R مشخص کننده مقدار ۱ برای بیت R می‌باشد.

ورودی	۱	۲	۳	۴	۳	۵	۶	۲	۵	۳	۴	۳	۵	۲
حافظه	۱ ↑	۱ ↑۲	۱ ↑۲ ۳	۴ ↘۲ ۳_R	۴ ↘۲ ۳_R	۴ ↙۵ ۳_R	۶ ↘۵ ۳	۶ ↙۲ ۳	۶ ↑۲ ۵	۳ ↘۲ ۵	۳ ↙۴ ۵	۳ ۳_R ↙۴ ۵	۳_R ↙۴ ۵_R	۳ ↙۴ ۵
نقص صفحه	*	*	*	*		*	*	*	*	*	*			*

جمعاً ۱۱ وقفه نقص صفحه رخ می‌دهد.

در این مثال در واقع ساعت و حالت‌های ممکن عقربه آن به صورت زیر است:



الگوریتم سالمندی (Aging)

در الگوریتم Aging به ازای هر صفحه یک شمارنده n بیتی (معمولاً ۸ بیتی) تعبیه می‌کنیم. در

پریودهای زمانی منظم، شمارنده همه صفحات یک بیت به سمت راست شیفت داده می‌شود و در واقع سمت راست‌ترین بیت از بین می‌رود و جای خالی بیت سمت چپ با بیت R هر صفحه پر می‌شود.

به عنوان مثال اگر شمارنه یک صفحه 00101101 و بیت R آن نیز یک باشد، بعد از وقوع وقفه تایمر، محتوای شمارنده این صفحه، 10010110 خواهد شد.

می‌توان گفت این شمارنده‌های ۸ بیتی، تاریخچه استفاده از هر صفحه را برای هشت پریود آخر نگهداری می‌کنند. سیستم عامل این شمارنده‌ها را به صورت اعداد بدون علامت در نظر می‌گیرد و هر چه شمارنده مربوط به یک صفحه، عدد بزرگ‌تری را نشان دهد، به این معناست که آن صفحه اخیراً بیشتر استفاده شده است و هر چه شمارنده عدد کوچکتری را نشان دهد، به این معناست که آن صفحه اخیراً کمتر استفاده شده است.

با این مقدمه، هنگام رخ دادن وقفه نقص صفحه، سیستم عامل صفحه‌ای را خارج می‌کند که شمارنده آن از همه کمتر باشد.

نکته: به عنوان مثال اگر شمارنده یک صفحه 0000 0000 باشد، به این معناست که این صفحه در ۸ پریود زمانی اخیر، اصلاً استفاده نشده است و اگر شمارنده یک صفحه 1111 1111 باشد، به این معناست که این صفحه در هر پریود حداقل یکبار مورد استفاده قرار گرفته است.

نکته: در این الگوریتم صفحه‌ای که شمارنده آن 10000001 می‌باشد، نسبت به صفحه‌ای با شمارنده 01111111 اولویت بیشتری جهت ماندن در حافظه دارد. در واقع صفحه دوم از حافظه خارج می‌شود زیرا از دید این الگوریتم صفحه اول در گذشته نزدیک‌تری به کار گرفته شده است.

نکته: هنگام وقوع وقفه نقص صفحه، ممکن است شمارنده چندین صفحه با هم برابر بوده و از همه کمتر باشند، در این حالت می‌توان بین آنها براساس الگوریتم FIFO یک صفحه را برای خروج انتخاب کرد.

نکته: الگوریتم Aging در واقع یک روش برای پیاده‌سازی ایده LRU می‌باشد.

الگوریتم NRU (Not Recently Used)

برای پیاده‌سازی این روش باید به ازای هر صفحه از دو بیت وضعیت استفاده کرد. این دو بیت عبارتند از بیت‌های M و R که به ترتیب از کلمات Modified و Referenced اقتباس شده‌اند. بیت R هنگامی برای یک صفحه Set می‌شود که به آن صفحه ارجاع شده باشد (چه این ارجاع برای خواندن باشد و چه برای نوشتن). اما بیت M هنگامی برای یک صفحه Set می‌شود که محتویات آن صفحه تغییر کرده باشد. در واقع در هر درایه جدول صفحه (به ازای هر صفحه) این دو بیت وجود دارند.

نحوه عملکرد الگوریتم NRU و شیوه استفاده از این دو بیت به صورت زیر است:

وقتی فرآیندی شروع به کار می‌کند، سیستم عامل بیت‌های R و M را به ازای همه صفحات آن با صفر پر می‌کند. هنگامی که به یک صفحه ارجاع می‌شود (فقط خواندن) بیت R آن صفحه یک می‌شود و هنگامی که یک صفحه تغییر کند (ارجاع جهت نوشتن) بیت M آن صفحه یک می‌شود. از طرفی برای اینکه بین صفحه‌ای که اخیراً به آن ارجاع شده و صفحه‌ای که قبلاً به آن ارجاع شده، تفاوت وجود داشته باشد، سیستم عامل در پریودهای زمانیِ منظم **همه** بیت‌های R را صفر می‌کند.

با اعمال این روش، هنگامی که یک نقص صفحه رخ می‌دهد، ابتدا سعی می‌شود صفحه‌ای برای خارج کردن انتخاب شود که به آن ارجاع نشده باشد، یعنی بیت R آن صفر باشد، اما اگر چنین صفحه‌ای پیدا نشد، ناچاریم از بین صفحاتی که بیت R آنها یک است، صفحه‌ای را انتخاب کنیم که از میان آنها، صفحه‌ای که تغییر نکرده باشد (بیت M آن صفر باشد) اولویت دارد زیرا نیازی به نوشتن دوباره آن در دیسک نداریم. در نهایت اگر چنین صفحه‌ای هم پیدا نشد، باید یک صفحه که بیت M آن یک است را انتخاب کنیم.

با دقت در الگوریتم NRU متوجه می‌شویم این الگوریتم صفحات را به ۴ گروه تقسیم می‌کند:

M	R	رده
۰	۰	گروه ۰
۰	۱	گروه ۱
۱	۰	گروه ۲
۱	۱	گروه ۳

با این تفسیر، گروه صفر صفحاتی هستند که اصلاً به آنها ارجاعی نشده است. گروه یک صفحاتی هستند که به آنها رجوع شده اما ارجاع‌ها فقط برای خواندن بوده و چیزی در آنها تغییر نکرده است. گروه ۲ در واقع صفحاتی هستند که کمی قبل به آنها ارجاع شده و این ارجاع آنها را تغییر هم داده است اما سیستم عامل با منقضی شده تایمر، بیت R آنها را صفر کرده است. گروه ۳ صفحاتی هستند که اخیراً به آنها ارجاع شده و تغییر هم یافته‌اند.

به این ترتیب مشاهده می‌شود بهترین گزینه برای ترک حافظه، گروه ۰، پس از آن گروه ۱، سپس گروه ۲ و در نهایت گروه ۳ می‌باشد.

نکته: الگوریتم **NRU** امکان ناهنجاری بیِ‌لیدی را **دارد**.

الگوریتم LFU **یا** NFU (Least Frequently Used **یا** Not Frequently Used)

این الگوریتم در برخی کتاب‌ها با عنوان LFU و در برخی دیگر با عنوان NFU ذکر شده است. این روش در واقع یکی از روش‌های پیاده‌سازی LRU می‌باشد. در این ایده، یک شمارنده به هر صفحه تعلق می‌گیرد که در شروع کار برابر صفر است. در هر وقفه‌ی ساعت، سیستم عامل بیت R هر صفحه را

(که می‌تواند صفر یا یک باشد) به شمارنده هر صفحه می‌افزاید. در واقع می‌توان گفت این شمارنده تعداد ارجاعات هر صفحه را شمارش و نگهداری می‌کند.
با اعمال این روش، هنگامی که یک وقفه‌ی نقص صفحه رخ می‌دهد، صفحه‌ای که شمارنده آن **کمترین مقدار** را دارد، جهت خروج از حافظه انتخاب می‌شود.
نکته: یک ایراد الگوریتم LFU این است که این الگوریتم هرگز چیزی را فراموش نمی‌کند. برای مثال، صفحه‌ای را در نظر بگیرید که در یک بازه زمانی بارها مورد ارجاع واقع شود و در نتیجه شمارنده آن به ناگاه افزایش چشمگیری پیدا کند، اما پس از مدتی ارجاعات به این صفحه قطع شوند. الگوریتم LFU موجب می‌شود این صفحه کماکان در حافظه باقی بماند زیرا مقدار شمارنده آن بسیار بالاست.

الگوریتم MFU (Most Frequently Used)
این الگوریتم همانند LFU از یک شمارنده به ازای هر صفحه استفاده می‌کند. نحوه افزایش این شمارنده نیز همانند LFU می‌باشد اما هنگام وقوع یک نقص صفحه، صفحه‌ای برای خروج انتخاب می‌شود که مقدار شمارنده آن از همه **بزرگ‌تر** باشد!!
ایده این الگوریتم این است که صفحه‌ای که شمارنده آن بزرگ است، به اندازه کافی در حافظه قرار داشته اما صفحه‌ای که شمارنده آن کوچک است احتمالاً تازه به حافظه وارد شده است و باید شانس استفاده شدن به وی داده شود.

تخصیص قاب به فرآیندها

جدا از اینکه الگوریتم جایگزینی صفحه در سیستم عامل چگونه باشد، چگونگی تخصیص قاب‌های فیزیکی به فرآیندها نیز تأثیر مهمی بر روی کارایی دارد.

تخصیص مساوی در برابر تخصیص متناسب

اولین مسئله‌ای که باید هنگام تخصیص قاب‌ها رعایت شود این است که در ابتدای کار قاب‌های موجود چگونه بین فرآیندها تقسیم شوند.
اولین الگوی تخصیص، الگوی **تخصیص مساوی** نام دارد. در این روش قاب‌های حافظه بین فرآیندهای موجود، به طور مساوی تقسیم می‌شوند. به عنوان مثال اگر در یک سیستم m قاب آزاد داشته باشیم و بخواهیم آنها را بین n فرآیند تقسیم کنیم، به هر یک از فرآیندها $\left(\frac{m}{n}\right)$ قاب آزاد اختصاص می‌یابد.
اما در الگوی **تخصیص متناسب** فرآیندها براساس اندازه نهایی خود، حافظه را در اختیار می‌گیرند. بنابراین فرآیندهای بزرگ‌تر تعداد قاب بیشتری اختیار خواهند کرد. فرض کنید تعداد صفحات فرآیند P_i برابر S_i و تعداد کل قاب‌های حافظه فیزیکی برابر m باشد. در این حالت به هر فرآیند، a_i قاب حافظه تعلق می‌گیرد که a_i برابر است با:

$$a_i = \frac{S_i}{\sum S_i} \times m$$

به عنوان مثال حافظه‌ای را در نظر بگیرید که ۷۰ قاب آزاد در اختیار داشته باشد. در این سیستم ۳ فرآیند P_1، P_2 و P_3 وجود دارند که اندازه P_1 برابر ۲۵ صفحه، اندازه P_2 برابر ۵۰ صفحه و اندازه P_3 برابر ۱۰۰ صفحه است. با استفاده از سیاست تخصیص متناسب، ۷۰ قاب آزاد حافظه به صورت زیر بین این ۳ فرآیند تقسیم می‌شوند:

$a_1 = \frac{۲۵}{۱۷۵} \times ۷۰ = ۱۰$ (تعداد قاب در اختیار P_1)

$a_2 = \frac{۵۰}{۱۷۵} \times ۷۰ = ۲۰$ (تعداد قاب در اختیار P_2)

$a_3 = \frac{۱۰۰}{۱۷۵} \times ۷۰ = ۴۰$ (تعداد قاب در اختیار P_3)

نکته: در عمل، سیستم عامل‌ها علاقه دارند تعداد قاب بیشتری در اختیار فرایندهای با اولویت بالاتر قرار دهند تا این فرآیندها با سرعت بیشتری اجرا شوند. به همین جهت اغلب سیستم عامل‌ها علاوه بر اندازه فرآیندها، به اولویت آنها نیز توجه می‌کنند. در واقع هنگام تخصیص قاب به فرآیندها، به فرآیندهای با اولویت بالاتر، حافظه بیشتری اختصاص می‌دهند.

تخصیص ثابت در برابر تخصیص متغیر

در سیاست **تخصیص ثابت**، تعداد ثابتی قاب به یک فرآیند داده می‌شود. در واقع در این روش هنگام بار شدن اولیه فرآیند، تعدادی قاب به آن اختصاص می‌یابد و تا انتها فرآیند تنها می‌تواند از همین تعداد قاب استفاده کند.

اما سیاست **تخصیص متغیر** اجازه می‌دهد تا تعداد قاب‌های تخصیص یافته به یک فرآیند در طول اجرای آن تغییر کند. به عنوان مثال اگر یک فرآیند به طور مداوم دچار وقفه نقص صفحه می‌شود، باید تعداد قاب بیشتری به وی داده شود تا نرخ خطای صفحه آن کاهش یابد.

تخصیص محلی در برابر تخصیص سراسری

تخصیص قاب‌ها به فرآیندها می‌تواند به دو صورت محلی و سراسری انجام شود:

در تخصیص محلی (Local)، هنگامی که برای یک فرآیند وقفه نقص صفحه رخ داد، فقط از بین قاب‌های مربوط به آن فرآیند می‌توان یک قاب را برای جایگزینی انتخاب کرد. اما در تخصیص سراسری (Global) هنگامی که یک فرآیند با نقص صفحه مواجه شد، از بین مجموعه کل قاب‌ها می‌توان یک قاب را برای جایگزینی انتخاب کرد، حتی اگر این قاب در اختیار فرآیندی دیگر باشد.

نکته: در واقع در تخصیص محلی، سهم هر فرآیند از حافظه هرگز تغییر نمی‌کند (تخصیص ثابت)، اما

در تخصیص سراسری، سهم فرآیندها از حافظه به صورت پویا تغییر می‌کند (تخصیص متغیر).

نکته: یکی از مشکلات تخصیص سراسری این است که مجموعه قاب‌های در اختیار یک فرآیند به رفتار صفحه‌بندی فرآیندهای دیگر نیز وابسته است. بنابراین یک فرایند یکسان ممکن است در اجراهای متفاوت، به گونه‌ای متفاوت عمل کند. مثلاً در یک اجرا ۱ ثانیه به طول انجامد و در اجرای دیگر ۵ ثانیه!

نکته: یکی از مشکلات تخصیص محلی این است که ممکن است یک فرآیند با نقص صفحه‌های مکرر مواجه شود و در واقع حافظه کم بیاورد، اما فرآیندهای دیگر از حافظه خود استفاده مؤثر و مفیدی نکنند، در واقع تعدادی قاب بدون استفاده باشند.

نکته: در عمل، سیستم عامل‌ها بیشتر از روش تخصیص سراسری استفاده می‌کنند.

اندازه قاب‌ها و صفحات

تعیین اندازه صفحات را می‌توان بر عهده سیستم عامل گذاشت. باید دقت کرد که انتخاب اندازه مناسب و بهینه برای صفحات، تأثیر مستقیمی بر روی کارایی دارد. برای بهبود کارایی گاهی اندازه صفحه باید کوچک باشد و گاهی بزرگ. برخی از عوامل مؤثر بر تعیین اندازه مناسب صفحه به قرار زیرند:

الف ـ معمولاً اندازه فرآیندها مضرب صحیحی از اندازه صفحه نیست و به طور متوسط نیمی از آخرین صفحه به ازای هر فرآیند خالی می‌ماند (تکه تکه شدن داخلی). بنابراین با کوچک‌تر شدن اندازه صفحات، میزان فضای تلف شده بابت تکه تکه شدن داخلی کاهش می‌یابد.

ب ـ بزرگ بودن اندازه صفحه یعنی بلااستفاده ماندن بخش بزرگی از حافظه زیرا وقتی که یک صفحه بزرگ به حافظه آورده شود، تمام قسمت‌های آن مورد استفاده قرار نمی‌گیرد. بنابراین با کوچک‌تر شدن اندازه صفحه، استفاده مؤثرتری از فضای حافظه می‌شود.

ج ـ هر چه اندازه صفحات بزرگ‌تر باشد، فرآیند به صفحات کمتری تقسیم می‌شود، در نتیجه اندازه جداول صفحه کاهش می‌یابد. در صورتی که با کوچک شدن اندازه صفحات، جداول صفحه بزرگ‌تر می‌شوند و این یعنی هزینه اضافی.

د ـ کوچک بودن صفحات و در نتیجه افزایش تعداد صفحات یک فرآیند به معنای افزایش تعداد دفعات جابجایی صفحات بین دیسک و حافظه است. با علم به اینکه مدت زمان لازم برای جابجایی یک صفحه کوچک بین دیسک و حافظه تقریباً برابر با مدت زمان جابجایی یک صفحه بزرگ می‌باشد، درمی‌یابیم کاهش اندازه صفحات و در نتیجه افزایش تعداد جابجایی‌ها منجر به تأخیر بیشتر می‌شود.

هـ ـ در برخی از سیستم‌ها هنگام تعویض متن، جداول صفحه باید در ثبات‌های سخت‌افزاری بار شوند. هر چه اندازه صفحات کوچک‌تر باشد، تعداد صفحات یک فرآیند بیشتر می‌شوند، در نتیجه اندازه

جدول صفحه بزرگ‌تر خواهند شد و بارکردن این جداول در ثبات‌های سخت‌افزاری بیشتر طول می‌کشد.

نکته: **سربار حافظه به ازای هر فرآیند** به دو قسمت تقسیم می‌شود: یکی میزان حافظه تلف شده به ازای آخرین صفحه فرآیند (تکه تکه شدن داخلی) و دیگری جدول صفحه (که به نوعی سربار محسوب می‌شود)، که باید این حافظه‌های سربار را به حداقل رساند. این مسئله را می‌توان به صورت ریاضی تحلیل کرد: اگر فرض کنیم اندازه متوسط فرآیندها s بایت، اندازه صفحه‌ها p بایت و اندازه هر درایه در جدول صفحه e باید باشد، هر فرآیند به طور متوسط به تعداد $\frac{s}{p}$ صفحه نیاز دارد در نتیجه اندازه جدول صفحه هر فرآیند برابر است با $e \times \frac{s}{p}$ بایت. از طرفی هر فرآیند در آخرین صفحه خود $\frac{p}{۲}$ بایت فضای تلف شده دارد (تکه تکه شدن داخلی). بنابراین هر فرآیند به طور متوسط سرباری معادل $\frac{es}{p} + \frac{p}{۲}$ خواهد داشت که باید این مقدار را به حداقل رساند. البته مشاهده می‌کنید در عبارت $e \times \frac{s}{p}$، اندازه صفحه (p) در مخرج کسر قرار دارد که با کاهش اندازه صفحه، این مقدار افزایش می‌یابد، اما در عبارت $\frac{p}{۲}$ اندازه صفحه در صورت کسر قرار دارد که با کاهش اندازه صفحه این مقدار کاهش می‌یابد. بنابراین اندازه بهینه p باید مقداری بین این دو مقدار باشد. برای به دست آوردن این نقطه، از عبارت $\frac{es}{p} + \frac{p}{۲}$ نسبت به p مشتق گرفته و مقدار آن را برابر صفر قرار می‌دهیم:

$$-\frac{se}{p^۲} + \frac{۱}{۲} = ۰ \quad \Rightarrow \quad p = \sqrt{۲se}$$

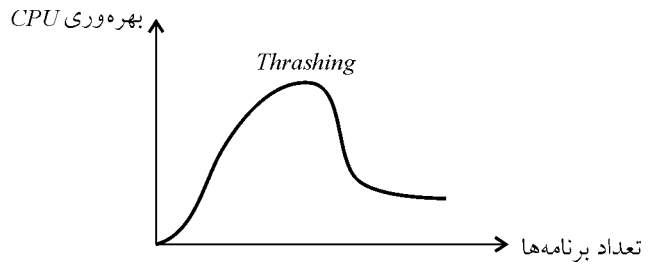
پس **اندازه بهینه صفحه** برای حداقل شدن سربار برابر است با $\sqrt{۲se}$.

۴ـ۱۳ـ کوبیدگی (له شدگی) (Thrashing)

اگر حافظه تخصیص داده شده به یک فرآیند، آن‌قدر کوچک باشد که نتواند صفحاتی که فرآیند، زیاد با آنها سر و کار دارد را در خود جای دهد، سرعت اجرای فرآیند کاهش می‌یابد، زیرا این فرآیند خطاهای نقص صفحه زیادی تولید می‌کند. در این حالت مدت زمان اجرای یک فرآیند به چندین و چند برابر حالت عادی افزایش می‌یابد. اصطلاحاً به برنامه‌ای که در هر ۲ یا ۳ دستور خود یک خطای نقص صفحه تولید کند، **کوبیده شده (لهیده)** گویند.

نکته: فرآیندی که در حالت Thrashing واقع است، به جای آن که زمان CPU را به اجرا اختصاص دهد، زمان زیادی را صرف انجام عملیات صفحه‌بندی می‌کند.

نکته: نسبت میزان بهره مفید CPU با افزایش تعداد فرآیندها به صورت زیر است:



پدیده کوبیدگی

در واقع تا یک نقطه، افزایش تعداد فرآیندها (افزایش سطح چندبرنامگی)، بهره‌وری CPU را افزایش می‌دهد، اما از یک نقطه به بعد، افزایش تعداد برنامه‌ها بهره‌وری پردازنده را کاهش می‌دهد. به عنوان مثال دو فرآیند A و B را در نظر بگیرید. فرض کنید برنامه A در حال اجرا به یک صفحه نیاز دارد، بنابراین بعد از یک خطای نقص صفحه، صفحه مورد تقاضایش به حافظه آورده می‌شود و به جای یک صفحه از فرآیند B در یک قاب حافظه قرار می‌گیرد، در این لحظه نوبت به اجرای برنامه B می‌رسد و فرآیند B به همان صفحه قدیمی خودش نیاز دارد، بنابراین با یک خطای نقص صفحه آن صفحه را به حافظه برمی‌گرداند و این بار صفحه فرآیند A باید حافظه را ترک کند و الی آخر ... در واقع در این حالت صفحاتی که فرآیندها زیاد با آنها سر و کار دارند به طور کامل در حافظه قرار ندارد و کارآیی سیستم به شدت کاهش می‌یابد.

نکته: اگر عملکرد سیستم عامل را در قبال درجه چندبرنامگی سیستم بررسی کنیم، متوجه می‌شویم تحت شرایطی خاص، پدیده Thrashing به شدت تشدید می‌شود.

سناریوی زیر را در نظر بگیرید:

می‌دانیم سیستم عامل بر بهره‌وری CPU نظارت دارد و اگر بهره‌وری CPU بسیار کم بشاد، درجه چندبرنامگی را با افزودن یک فرآیند جدید به سیستم افزایش می‌دهد تا بهره CPU افزایش یابد. فرض کنید در این سیستم از یک الگوریتم جایگزینی سراسری برای صفحات استفاده می‌شود که صفحات را بدون توجه به این که مربوط به کدام فرآیند هستند، جایگزین می‌کند. حال فرض کنید فرآیندی وارد یک مرحله جدید از اجرا شده و به چند صفحه جدید نیاز دارد. بنابراین وقفه‌های نقص صفحه برای این فرآیند آغاز می‌شوند و این فرآیند قاب‌های فرآیندهای دیگر را در اختیار می‌گیرد. از طرفی فرآیندهایی که تعدادی از صفحات آنها از حافظه خارج شده‌اند، به آن صفحات نیاز دارند، بنابراین وقفه‌های نقص صفحه برای این فرآیند آغاز می‌شوند و این فرآیند قاب‌های فرآیندهای دیگر را در اختیار می‌گیرد. از طرفی فرآیندهایی که تعدادی از صفحات آنها از حافظه خارج شده‌اند، به آن صفحات نیاز دارند، بنابراین وقفه‌های نقص صفحه برای آنها نیز شروع می‌شود و به طور مشابه این فرآیندها قاب‌های فرآیندهای دیگر را در اختیار می‌گیرند و این مسئله برای فرآیندهای دیگر تکرار

می‌شود. بنابراین فرآیندها پی در پی خطای نقص صفحه را تجربه می‌کنند و صفحات درگیر، مرتباً به داخل و خارج حافظه مبادله می‌شوند. به این ترتیب بهره‌وری CPU کاهش می‌یابد و زمانبند پردازنده متوجه این کاهش بهره‌وری می‌گردد و جهت افزایش بهره‌وری CPU درجه چندبرنامگی را افزایش می‌دهد.

فرآیندهای جدید نیز برای شروع سعی می‌کنند قاب‌هایی از سایر فرآیندها دریافت کنند که در نتیجه نقص صفحه‌های بیشتری رخ می‌دهد.

نکته: پدیده Thrashing را می‌توان با استفاده از الگوریتم‌های محلی جایگزینی صفحه به جای الگوریتم‌های سراسری، کنترل کرد. در این حالت یک فرآیند که دچار کوبیدگی شده است نمی‌تواند فرآیندهای دیگر را نیز مبتلا کند!

نکته: یک ایده دیگر جهت مقابله با Thrashing این است که فرآیندها اولویت بندی شوند. در این صورت برنامه‌های با اولویت پایین‌تر اجازه ندارند صفحات مربوط به فرآیندهای با اولویت بالاتر را از حافظه خارج کنند.

نکته: یک راه دیگر مقابله با Thrashing کنترل تعداد وقفه‌های نقص صفحه می‌باشد. به این ترتیب که اگر تعداد نقص‌های صفحه برای یک فرآیند افزایش یافت، باید تعدادی قاب حافظه به آن اختصاص یابد و اگر تعداد نقص‌های صفحه برای یک فرآیند از یک حد پایین کمتر شد، باید تعدادی قاب از آن فرآیند پس گرفته شود. نکته مهم اینجاست که اگر تعداد نقص‌های صفحه یک فرآیند بالا رفت، ولی قاب آزاد در حافظه وجود نداشت، باید درجه چندبرنامگی سیستم را کاهش داد و یک یا چند فرآیند را به حالت معلق درآورد تا قاب‌هایی که در اختیار دارد، آزاد شود. به این تکنیک **فرکانس خطای صفحه یا PFF** (Page Fault Frequency) گویند.

مدل مجموعه کاری (Working Set Model)

مجموعه کاری هر فرآیند، عبارت است از صفحاتی از آن فرآیند که اگر در حافظه قرار داشته باشند، فرآیند موردنظر کارآیی و سرعت قابل قبولی دارد. در واقع اگر مجموعه کاری یک فرآیند در حافظه باشد، فرآیند با تعداد معقول و مناسبی وقفه نقص صفحه به کار خود ادامه می‌دهد. مجموعه کاری یک فرآیند در طول اجرای آن ممکن است تغییر کند. به عنوان مثال در یک بازه زمانی خاص، یک فرآیند فقط به ۷ صفحه خود نیاز دارد، بنابراین مجموعه کاری این فرآیند در آن بازه، آن ۷ صفحه هستند، اما همین فرآیند در یک بازه دیگر فقط، به ۵ صفحه دیگر خود نیاز دارد.

ایده مدل مجموعه کاری در واقع یک رهیافت جهت مقابله با پدیده Thrashing است. در این مدل (راهکار) از مفهوم مجموعه کاری به خوبی استفاده می‌شود. به طور خلاصه مدل مجموعه کاری بیان می‌کند که قبل از دادن نوبت اجرا به یک فرآیند، باید مجموعه کاری آن فرآیند به درون حافظه بار شود.

می‌توان گفت با انتقال مجموعه کاری یک فرآیند قبل از اجرای آن به درون حافظه، نرخ خطاهای نقص صفحه کاهش می‌یابد.

نکته: به بار کردن صفحات یک فرآیند قبل از اجرای آن، **پیش صفحه‌بندی (Prepaing)** گویند.

نکته: ایده **مدل مجموعه کاری** از صادق بودن **اصل محلی بودن مراجعات** استفاده می‌کند. اصل محلی بودن مراجعات به بیان ساده چنین است: به طور معمول هنگام اجرای یک فرآیند معقول، ارجاعات به حافظه یک گستره خاص محدود هستند. البته این گستره در طول اجرای فرآیند، از مکانی به مکان دیگر منتقل می‌شود.

ایده حافظه مجازی اغلب با تکنیک صفحه‌بندی پیاده‌سازی می‌شود اما حافظه مجازی را می‌توان با قطعه‌بندی نیز ترکیب کرد. اما چون اندازه قطعات در قطعه‌بندی مساوی نیستند، این روش قدری پیچیده است.

نکته: رعایت اصول ساده برنامه‌نویسی می‌تواند در افزایش سرعت اجرای فرآیندها و کاهش تعداد نقص صفحه تأثیر مستقیم داشته باشد. به عنوان مثال قطعه برنامه زیر را در نظر بگیرید که سعی دارد همه عناصر یک آرایه دو بعدی ۱۰۰×۱۰۰ را با صفر پر کند:

```
for i:=1 to 100 do
    for j:=1 to 100 do
        a[j,i]=0;
```

فرض کنید هر یک از عناصر این آرایه، ۲ بایتی هستند و اندازه هر صفحه در حافظه نیز ۲۰۰ بایت باشد. در این صورت این آرایه ۱۰۰۰۰ عنصری، جمعاً ۱۰۰ صفحه را اشغال می‌کند و از آنجا که عناصر آرایه در زبان‌های برنامه‌نویسی C و پاسکال به صورت سطری در حافظه ذخیره می‌شوند، در واقع هر سطر این آرایه در یک صفحه قرار می‌گیرد. یعنی در صفحه اول، عناصر a[1,1] تا a[1,100]، در صفحه دوم عناصر a[2,1] تا a[2,100]، تا صفحه صدم که عناصر a[100,1] تا a[100,100] قرار می‌گیرند.

با دقت در قطعه برنامه نوشته شده، مشاهده می‌شود که پردازش به صورت ستونی انجام می‌شود زیرا از اندیس حلقه بیرونی به عنوان اندیس ستون در آرایه استفاده کرده است (اندیس i). به این ترتیب اگر فط یک صفحه برای داده‌ها در اختیار این برنامه باشد، برای انجام عملیات خود، دقیقاً ۱۰۰۰۰ خطای نقص صفحه رخ می‌دهد. زیرا هنگامی که یک صفحه به حافظه آورده شد، فقط یکی از عناصر آن پردازش می‌شود و برای عنصر بعدی یک خطای نقص صفحه رخ می‌دهد، زیرا این عنصر در صفحه بعدی قرار دارد.

به این ترتیب به ازای هر عنصر، یک خطای نقص صفحه رخ می‌دهد.

اما اگر برنامه به صورت زیر نوشته شود، شرایط تغییر می‌کند:

```
for i:=1 to 100 do
        for j:=1 to 100 do
            a[i,j]=0;
```

در این حالت هنگامی که یک صفحه به حافظه آورده می‌شود، هر ۱۰۰ عنصر آن به ترتیب پردازش می‌شوند و نیازی به نقص صفحه نیست. در واقع جمعاً ۱۰۰ نقص صفحه رخ می‌دهد.

کنکور کارشناسی ارشد

سیستم عامل

مؤلف:

ارسطو خلیلی‌فر

انتشارات آزاده

تقدیم به:

تمامی آنانی که برای پیشرفت و سعادت خود و بشریت

تلاش می کنند.

ارسطو خلیلی فر

به نام خدا

مقدمه مولف

به نام خداوند جان و خرد کزین برتر اندیشه برنگذرد

کتاب حاضر، کامل‌ترین مرجع حل تشریحی سوالات درس سیستم عامل، ویژه کنکور کارشناسی ارشد مهندسی کامپیوتر و مهندسی فناوری اطلاعات می‌باشد. با توجه به اهمیت خاصی که درس سیستم عامل برای موفقیت در آزمون کارشناسی ارشد دارد، بعد از سال‌ها تدریس و تحقیق، تصمیم گرفتم، این مجموعه خاص را به شیوه‌ای منحصربه‌فرد جمع‌آوری نمایم. در حال حاضر اکثر اساتید دانشگاه‌های معتبر کشور از کتاب‌های سه نویسنده مشهور این درس یعنی آبراهام سیلبرشاتز، ویلیام استالینگز و اندرو اس تنن‌بام استفاده می‌کنند و عموم تست‌های مطرح شده در چند سال اخیر از مفاهیم همین کتاب‌ها بوده است. لذا بر آن شدم تا با بهره‌گیری از نوشته‌های این کتب مرجع و برخی منابع معتبر دیگر، کتابی را به صورت جامع به رشته تحریر درآورم. این کتاب تلاش نموده است تا مباحث مطرح شده را برمبنای تدریس دانشگاهی و براساس سرفصل‌های مصوب وزارت علوم، تحقیقات و فناوری و منطبق با مفاهیم و مباحث تدریس شده در دانشگاه‌های معتبر ایران و جهان بررسی و تبیین نماید.

این کتاب در هشت فصل و براساس جدیدترین تغییرات منابع و آزمون‌های کارشناسی ارشد تنظیم شده است.

این کتاب علاوه براینکه برای داوطلبان آزمون کارشناسی ارشد، قابل استفاده است، مرجع ارزنده و جامعی برای آموزش این درس در دانشگاه‌ها می‌باشد.

با کمال میل خرسندم که مراتب سپاسگزاری و قدردانی خود را از همه عزیزانی که در تهیه این کتاب نقش داشته‌اند به واسطه حمایت‌های بی‌دریغشان تقدیم نمایم.

از جناب آقای جعفر بدوستانی ریاست محترم **انتشارات آزاده (راهیان ارشد)** که در به ثمر رساندن این اثر بسیار محبت کرده‌اند، سپاسگزارم.

جناب آقای امیر بدوستانی مدیریت محترم بخش نشر و چاپ انتشارات به پاس سخت‌کوشی و تلاش فراوانشان، کمال تشکر و قدردانی را می‌نمایم.

هم‌چنین سرکار خانم فرزانه محمدلو که انجام امور اجرایی کتاب را برعهده داشتند، کمال تشکر و قدردانی را می‌نمایم.

در پایان دوست دارم در یک بیان صمیمانه، به خوانندگان محترم، ابراز کنم که در نوشتن این کتاب چیزی به جز عشق و خدمت را لحاظ نکرده‌ام. تمام فکرم آن بوده که هرآنچه در توان دارم را در این مسیر به کار گیرم.

با وجود دقت فراوانی که در تهیه این اثر به‌کار رفته است، وجود اشتباه در آن اجتناب‌ناپذیر است. لذا از تمامی دوستان، اساتید و دانشجویان عزیز خواهشمندم هرگونه نظر و پیشنهاد در زمینه اصلاح یا بهبود این کتاب را از طریق سایت:

khalilifar.ir

و یا صفحه اینستاگرام:

arastoo.khalilifar

و یا صفحه تلگرام:

@arastookhalilifar

و یا به طور مستقیم با شماره تلفن ۰۹۱۲۲۳۰۶۶۰۲ با من در میان بگذارند.

امید است که این خدمت ناچیز مورد قبول خداوند متعال قرار گرفته و قابل استفاده شما عزیزان باشد.

ارسطو خلیلی‌فر

فهرست مطالب

تست‌های فصل اول: مفاهیم اولیه ۹
پاسخ تست‌های فصل اول: مفاهیم اولیه ۱۰

تست‌های فصل دوم: مدیریت فرآیندها و زمان‌بندی پردازنده ۱۴
پاسخ تست‌های فصل دوم: مدیریت فرآیندها و زمان‌بندی پردازنده ۱۸

تست‌های فصل سوم: مدیریت نخ ۴۰
پاسخ تست‌های فصل سوم: مدیریت نخ ۴۳

تست‌های فصل چهارم: مدیریت حافظه اصلی ۵۵
پاسخ تست‌های فصل چهارم: مدیریت حافظه اصلی ۶۰

تست‌های فصل پنجم: مدیریت حافظه مجازی ۱۱۱
پاسخ تست‌های فصل پنجم: مدیریت حافظه مجازی ۱۱۷

تست‌های فصل ششم: مدیریت فرآیندها و نخ‌های هم‌روند ۱۵۱
پاسخ تست‌های فصل ششم: مدیریت فرآیندها و نخ‌های هم‌روند ۱۵۹

تست‌های فصل هفتم: مدیریت بن‌بست ۲۵۳
پاسخ تست‌های فصل هفتم: مدیریت بن‌بست ۲۵۷

تست‌های فصل هشتم: مدیریت دیسک ۲۷۴
پاسخ تست‌های فصل هشتم: مدیریت دیسک ۲۷۶

جهت کسب اطلاعات بیشتر درباره‌ی تازه‌ترین خبرهای آزمون کارشناسی ارشد در گرایش‌های مختلف رشته‌ی مهندسی کامپیوتر و مهندسی فناوری اطلاعات به سایت ارسطو خلیلی‌فر مراجعه نمایید:

www.khalilifar.ir

همچنین از شما درخواست می‌کنیم که سؤالات، طرح‌ها و پیشنهادات خود را برای بهبود، تکمیل و تصحیح این کتاب با ما در میان بگذارید.

ایمیل‌های خود را به آدرس arastoo.khalilifar@gmail.com ارسال نمایید.

انتشارات آزاده (راهیان ارشد)

تست‌های فصل پنجم: مدیریت حافظه مجازی

۱- **با توجه به بحث Copy-On-Write بین فرآیندهای پدر (Parent) و فرزند (Child) در فراخوان سیستمی fork، در راستای افزایش کارایی، کدام جمله در مورد تقدم و تأخر اجرای این فرآیندها در لحظه ایجاد فرآیند فرزند صحیح است؟** (مهندسی کامپیوتر – دولتی ۸۹)

۱) با توجه به آگاهی زمان‌بند از محتوای (برنامه) فرآیند پدر، بهتر است فرآیند فرزند زودتر اجرا شود.

۲) با توجه به آگاهی زمان‌بند از محتوای (برنامه) فرآیند پدر، بهتر است فرآیند پدر زودتر اجرا شود.

۳) با توجه به عدم آگاهی زمان‌بند از محتوای (برنامه) فرآیند فرزند، بهتر است فرآیند پدر زودتر اجرا شود.

۴) با توجه به عدم آگاهی زمان‌بند از محتوای (برنامه) فرآیند فرزند، بهتر است فرآیند فرزند زودتر اجرا شود.

۲- **یک حافظه مجازی با این مشخصات در نظر بگیرید. زمان دسترسی حافظه 50ns، زمان دستیابی TLB 2ns، نسبت اصابت TLB (hit ratio)، 98% و احتمال خطای صفحه برای کل دسترسی‌ها به حافظه 2×10^{-6} است. زمان انتقال صفحه از دیسک را 10ms فرض کنید. برای سرعت بخشیدن به این حافظه از حافظه پنهان (cache) با این مشخصات استفاده شده است. زمان دسترسی حافظه پنهان 10ns، نسبت اصابت حافظه پنهان 90%، جریمه هر عدم اصابت در حافظه پنهان 100ns است. میانگین زمان دسترسی به حافظه برای هر آدرس به کدام یک از گزینه‌های زیر نزدیک‌تر است؟** (مهندسی IT- دولتی۸۹)

۱) 75ns　　۲) 45ns

۳) 73ns　　۴) 43ns

۳- **به فرآیندی 4 قاب (frame) تخصیص یافته است. (تمام اعداد دهدهی هستند و همه شماره‌گذاری‌ها از صفر شروع شده است.) زمان آخرین بارشدن یک صفحه در یک قاب، زمان آخرین دستیابی به صفحه، شماره صفحه مربوط به هر قاب، بیت‌های مراجعه (R) و تغییر (M) برای هر قاب در جدول زیر نشان داده شده‌اند (زمان‌ها برحسب ضربان ساعت و از شروع فرآیند در زمان صفر است.)** (مهندسی IT- دولتی۸۹)

شماره صفحه	شماره قاب	زمان بار شدن صفحه در حافظه	زمان آخرین مراجعه	بیت مراجعه (R)	بیت تغییر (M)
2	0	60	160	0	1
1	1	130	161	0	0
0	2	26	163	1	0
3	3	20	162	1	1

یک خطای صفحه برای صفحه 4 رخ داده است. برای هر یک از سیاست‌های مدیریت حافظه FIFO و LRU و Clock (اشاره‌گر روی صفحه صفر است و اولویت با انتخاب صفحه تغییر نیافته است) به ترتیب چه صفحه‌ای برای جایگزینی انتخاب می‌شود؟

۱) 0 و 1 و 1 ۲) 1 و 0 و 2 ۳) 3 و 2 و 1 ۴) 1 و 1 و 2

۴- یک سیستم حافظه مجازی صفحه‌بندی را در نظر بگیرید که از دیسک با زمان دسترسی و انتقال 3 میلی‌ثانیه برای هر صفحه استفاده می‌کند. هر دسترسی به حافظه دارای زمان 50 نانوثانیه است. برای بهبود کارآیی میانگیر دم‌دستی ترجمه (TLB) اضافه شده است که دارای زمان دسترسی 2 نانوثانیه می‌باشد. فرض کنید 98% دسترسی‌ها از طریق TLB انجام می‌گیرد. اگرچه می‌توان میانگین زمان دسترسی به یک آدرس حافظه مجازی را محاسبه نمود، ولی زمان‌های واقعی دسترسی‌ها متعدد هستند و با مقدار میانگین تفاوت دارند. چند زمان واقعی دسترسی عبارتند از: (از اعداد بسیار کوچک در مقابل اعداد بسیار بزرگ صرف‌نظر کنید.) (مهندسی IT- دولتی۹۰)

۱) 52 نانوثانیه، 6 میلی‌ثانیه، 3 میلی‌ثانیه ۲) 50 نانوثانیه، 3.5 میلی‌ثانیه، 100 نانوثانیه

۳) 50 نانوثانیه، 100 نانوثانیه، 7 میلی‌ثانیه ۴) 102 میلی‌ثانیه، 3.5 میلی‌ثانیه، 7 میلی‌ثانیه

۵- راه‌حل‌های سخت‌افزاری متعددی به همراه سیستم عامل به اجرای برنامه‌ها کمک می‌کنند. در اینجا به چهار مورد از این راه‌حل‌ها اشاره شده است. در کدام گزینه هر چهار مورد راه‌حل سخت‌افزاری است؟ (مهندسی IT- دولتی۹۰)

۱) ۱- کنترل جدول TLB، ۲- تبدیل آدرس منطقی به فیزیکی، ۳- ثبت تغییر یافتن (modify) یک صفحه، ۴- تشخیص دستورالعمل غیرمجاز

۲) ۱- اعلام پایان سهم زمانی، ۲- حافظه پنهان (cache)، ۳- تشخیص نبودن صفحه در حافظه اصلی، ۴- بار کردن ثبات‌های مدیریت حافظه

۳) ۱- ثبت دسترسی (access) به یک صفحه، ۲- بردن فرآیند از حالتی (state) به حالت دیگر، ۳- تشخیص سرریز در یک محاسبه، ۴- خروج فرآیند

۴) ۱- تبدیل آدرس قطعه‌بندی شده به آدرس حافظه اصلی، ۲- جلوگیری از اجرای دستورالعمل تعریف نشده، ۳- یافتن صفحه خطا خورده (fault) در دیسک، ۴- حفاظت از فضای حافظه فرآیندها

۶- تبدیل آدرس منطقی به فیزیکی در مدیریت صفحه‌بندی در یک پردازنده در شکل زیر مشاهده می‌شود (صفحه‌بندی دو سطحی). اطلاعات موجود عبارتند از:

اندازه هر درایه (entry) جدول صفحه 4 بایت است.

زمان دسترسی به TLB برابر 2ns

زمان دسترسی به حافظه برابر 50ns

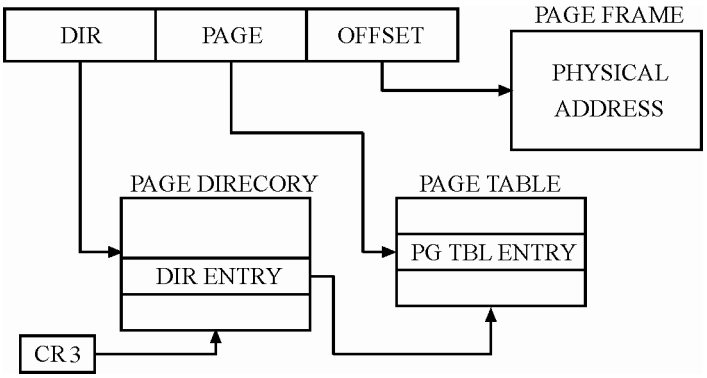
زمان دسترسی به حافظه پنهان (cache) برابر 10ns

جریمه cache miss برابر 100ns

زمان تبادل یک صفحه بین حافظه و دیسک برابر 5ms

فرمت آدرس منطقی پردازنده

تبدیل آدرس منطقی به فیزیکی در پردازنده



کدام گزینه زمان‌های ممکن برای دسترسی به یک مکان حافظه که با آدرس منطقی مشخص شده است را نشان می‌دهد؟ (مهندسی کامپیوتر – دولتی۹۱)

۱) 32ns, 162ns, 152ns, 482ns, 10ms, 15ms

۲) 32ns, 162ns, 132ns, 282ns, 382ns, 15ms, 10ms

۳) 12ns, 82ns, 162ns, 282ns, 332ns, 5ms, 10ms, 15ms

۴) 12ns, 32ns, 112ns, 132ns, 232ns, 332ns, 5ms, 10ms

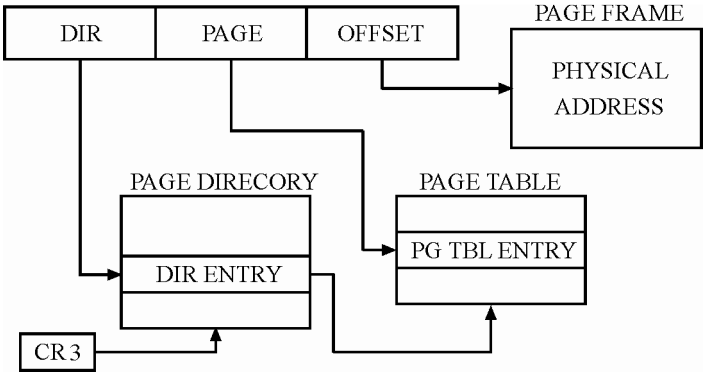
۷- تبدیل آدرس منطقی به فیزیکی در مدیریت صفحه‌بندی (دو سطحی) برای یک پردازنده مفروض در شکل زیر مشاهده می‌شود. اطلاعات زیر موجود است: (مهندسی IT- دولتی۹۱)

زمان دسترسی به TLB برابر 2ns

زمان دسترسی به حافظه برابر 50ns

زمان دسترسی به حافظه پنهان (cache) برابر 10ns
جریمه cache miss برابر 100ns
میانگین زمان تبادل صفحه بین حافظه و دیسک برابر 5ms
نسبت اصابت TLB (hit ratio) برابر 98%
نسبت اصابت (hit ratio) حافظه پنهان برابر 90%
احتمال روی دادن خطای فقدان صفحه برای هر دسترسی به حافظه $p = 10^{-6}$ است.
فرمت آدرس منطقی پردازنده مفروض

تبدیل آدرس منطقی به فیزیکی در پردازنده مفروض



میانگین زمان دسترسی به یک مکان حافظه که با آدرس منطقی مشخص شده به کدام یک از اعداد زیر نزدیک‌تر است؟

۱) 128 نانوثانیه ۲) 57 نانوثانیه ۳) 107 نانوثانیه ۴) 28 نانوثانیه

۸- کدام یک از فن‌های برنامه‌نویسی و ساختارها «مناسب» محیط صفحه‌بندی بر پایه تقاضا هستند؟ و کدام یک مناسب نیستند؟ **(مهندسی IT- دولتی۹۱)**

۱- پشته، ۲- جدول سمبل hashed، ۳- جستجوی ترتیبی، ۴- جستجوی دودویی، ۵- کُد خالص، ۶- عملیات برداری، ۷- کارکرد غیرمستقیم

۱) ۱- نامناسب، ۲- نامناسب، ۳- مناسب، ۴- نامناسب، ۵- مناسب، ۶- نامناسب، ۷- نامناسب
۲) ۱- مناسب، ۲- نامناسب، ۳- مناسب، ۴- نامناسب، ۵- مناسب، ۶- مناسب، ۷- نامناسب
۳) ۱- نامناسب، ۲- نامناسب، ۳- مناسب، ۴- مناسب، ۵- مناسب، ۶- نامناسب، ۷- نامناسب
۴) ۱- مناسب، ۲- مناسب، ۳- مناسب، ۴- مناسب، ۵- مناسب، ۶- نامناسب، ۷- مناسب

۹- کدام گزینه زیر درباره جدول صفحه معکوس (Inverted Page Table) درست نیست؟

(مهندسی کامپیوتر – دولتی۹۲)

۱) این نوع جدول زمان نگاشت آدرس منطقی به آدرس فیزیکی را کاهش می‌دهد.
۲) این نوع جدول صفحه سبب کاهش اندازه حافظه فیزیکی جهت ذخیره‌سازی آن می‌شود.
۳) در این نوع جدول صفحه زمان سرویس نقص صفحه (page fault) به دلیل ایجاد یک نقص صفحه دیگر افزایش می‌یابد.
۴) برای این نوع جدول صفحه می‌بایست یک جدول صفحه خارجی نیز ذخیره شود.

۱۰- کدام یک از گزینه‌های زیر درباره مدیریت حافظه با روش صفحه‌بندی درست نیست؟

(مهندسی IT- دولتی۹۲)

۱) کاهش اندازه صفحه سبب کاهش زمان سرویس نقص صفحه می‌شود.
۲) کاهش اندازه صفحه سبب افزایش بهره‌وری حافظه و افزایش زمان I/O می‌شود.
۳) کاهش اندازه صفحه سبب کاهش تکه تکه شدن خارجی (external fragmentation) حافظه می‌شود.
۴) کاهش اندازه صفحه‌ها سبب کاهش تکه تکه شدن داخلی (internal fragmentation) حافظه می‌شود.

۱۱- در یک سیستم مدیریت حافظه که به فناوری copy-on-write مجهز است پس از تولید تعدادی فرزند توسط یک فرآیند، کدام گزینه زیر درست است؟ **(مهندسی IT- دولتی۹۲)**

۱) هر یک از فرزندان و یا فرآیند اصلی در هر لحظه می‌توانند به کار خود پایان دهند.
۲) فرآیند پدر در صورتی می‌تواند به کار خود پایان دهد که فرزندان همه صفحه‌های مربوط به خود را تغییر داده باشند.
۳) فرآیند پدر در صورتی می‌تواند به کار خود پایان دهد که دست کم، یک فرزند، یکی از صفحه‌های خود را تغییر داده باشد.
۴) هر سه پاسخ درست می‌باشد.

۱۲- در یک سیستم صفحه‌بندی برحسب تقاضا، اگر احتمال نقص صفحه برابر p باشد و زمان انتقال یک صفحه بین حافظه جانبی و حافظه اصلی برابر با d باشد و به طور میانگین نیمی از صفحات در حافظه اصلی تغییر پیدا کرده باشند. اگر از یک حافظه جانبی با سرعت ۲ برابر استفاده شود، آنگاه متوسط زمان دسترسی موثر به حافظه، چقدر کاهش خواهد یافت؟ **(مهندسی کامپیوتر – دولتی۹۳)**

۱) $\frac{1}{2}pd$ ۲) $\frac{2}{3}pd$ ۳) $\frac{3}{4}pd$ ۴) pd

---

۱۳- اگر یک پردازه با دستور ( )fork پردازه جدیدی را ایجاد نماید، کدام یک از داده‌های زیر بین پدر و فرزند به اشتراک گذاشته نمی‌شود؟ **(مهندسی IT- دولتی۹۳)**

۱) Process id ۲) Heap ۳) Stack ۴) Code

۱۴- سیستم عامل می‌تواند بر کوبیدگی (Thrashing) غلبه نماید اگر .....

(مهندسی IT- دولتی۹۳)

۱) اندازه صفحه افزایش یابد.
۲) درجه چند برنامگی را افزایش دهد.
۳) سرعت ورودی و خروجی افزایش یابد.
۴) تخصیص حافظه به پردازه‌ها با توجه به اندازه پنجره‌کاری آنها تنظیم گردد.

۱۵- چگونه سیستم عامل می‌تواند بر کوبیدگی (thrashing) غلبه کند؟ (مهندسی کامپیوتر-دولتی۹۴)

۱) اندازه صفحه افزایش یابد.
۲) درجه چند برنامگی افزایش یابد.
۳) سرعت ورودی و خروجی افزایش یابد.
۴) تخصیص حافظه به پردازه‌ها با توجه به اندازه پنجره کاری آنها تنظیم گردد.

۱۶- در رشته مرجع صفحه زیر اگر از الگوریتم LRU(Least Recently Used) برای جایگزینی صفحات استفاده شود چند خطای صفحه رخ می‌دهد؟ (مهندسی IT - دولتی۹۴)

7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1

۱) 14 ۲) 13 ۳) 12 ۴) 11

۱۷- فرض کنید در یک سیستم حافظه، زمان دسترسی به کش (حافظه نهان) 10 نانو ثانیه است و زمان دسترسی به حافظه اصلی 200 نانو ثانیه است. زمان دسترسی موثر در این سیستم، 10 درصد بیشتر از زمان دسترسی به کش است. نرخ اصابت (Hit Ratio) در این سیستم چقدر است؟

(مهندسی IT -دولتی۹۴)

۱) $\frac{200}{210}$ ۲) $\frac{199}{200}$ ۳) $\frac{190}{200}$ ۴) $\frac{110}{200}$

پاسخ تست‌های فصل پنجم: مدیریت حافظه مجازی

۱- گزینه (۴) صحیح است.

اصولاً سیستم عامل و بخش‌های مختلف آن از جمله زمان‌بند، از محتوای برنامه‌های کاربر مطلع نیست. به بیان دیگر، زمان‌بند، برنامه‌های کاربر را به صورت یک Black Box (جعبه سیاه) می‌بیند و از محتوای آن اطلاعی ندارد. بنابراین گزینه‌های اول و دوم نادرست است.

سیستم عامل Unix برای ایجاد یک فرآیند جدید از فراخوان سیستمی Fork استفاده می‌کند. در این سیستم عامل جهت پیاده‌سازی مفهوم حافظه مجازی از تکنیک Copy-On-Write استفاده می‌گردد. Copy-On-Write یکی از فیلدهای جدول صفحه است و هنگامی که بیش از یک فرآیند در یک صفحه باشد، این فیلد براساس تعداد فرآیندهای موجود در یک صفحه مقدار می‌گیرد.

در سیستم عامل Unix، هرگاه فرآیندی (فرآیند پدر) فراخوانی سیستمی Fork را انجام دهد فرآیند جدیدی (فرآیند فرزند) ایجاد می‌شود که کدها و داده‌های آن عیناً همان کدها و داده‌های فرآیند پدر است. در واقع جهت صرفه‌جویی در حافظه، فرآیندهای پدر و فرزند در ابتدای کار از صفحات کد و داده به صورت مشترک استفاده می‌کنند، البته تا زمانی که فقط عمل خواندن (Read) بین فرآیندها مدنظر باشد، این صفحات به صورت مشترک استفاده می‌گردد.

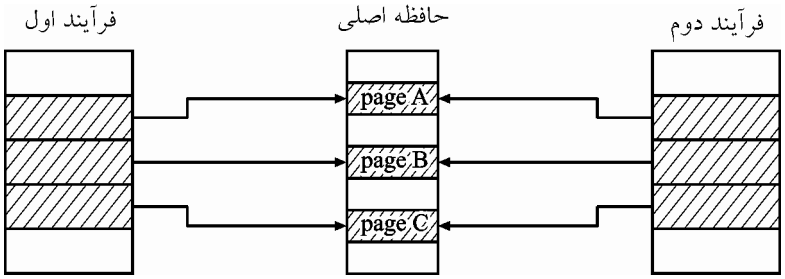
اما بر طبق تکنیک Copy-On-Write هرگاه یکی از دو فرآیند پدر یا فرزند بخواهد صفحه‌ای از صفحات مشترک را تغییر دهد (مثلاً چیزی بنویسد) یک کپی جداگانه از آن فرآیند ساخته می‌شود و فرآیند از آن به بعد از آن استفاده می‌کند (به جای صفحه مشترک) و فرآیندهای دیگر از صفحات اصلی (مشترک) استفاده می‌کنند. بدین ترتیب محرمانگی داده‌ها حفظ شده و تغییرات صورت گرفته توسط یک فرآیند بر روی سایر فرآیندها اثر نخواهد داشت.

در سیستم عامل Unix یک فرآیند فرزند پس از ایجاد توسط فراخوانی Fork، می‌تواند بلافاصله از فراخوانی سیستمی ()exec استفاده کند و کل فضای حافظه‌ی خود را جایگزین کند و از ادامه شراکت با پدر صرفه نظر کند. بنابراین، با این کار، حافظه‌ی جداگانه‌ای برای فرآیند فرزند ایجاد می‌شود. با توجه به این مطلب اگر بعد از اجرای Fork و ایجاد فرآیند فرزند، ابتدا فرآیند پدر اجرا گردد، ممکن است فرآیند پدر بخواهد بطور خصوصی در یکی از صفحات چیزی بنویسد و باعث ایجاد یک کپی از صفحه بر اساس تکنیک Copy-On-Write شود. حال اگر در ادامه فرآیند فرزند اجرا گردد و در همان ابتدا از فراخوان سیستمی ()exec استفاده کند و راه خود را از پدر جدا کند و شراکت را برهم زند، صفحاتی که پدر به دلیل تغییرات خود ایجاد کرده بود، سربار به حساب می‌آیند و کار بیهوده تلقی می‌گردد، مانند پدری که پس از فرزنددار شدن برای صرفه‌جویی در هزینه‌ها، از خانه‌ی خود به شکل اشتراکی استفاده می‌کند. اما این پدر بعدها به دلیل کارهای شخصی خود خانه‌ی دیگری را نیز تهیه می‌کند و بعد از تهیه خانه‌ی دوم متوجه می‌شود، که فرزند راه خود را جدا کرده است، و شراکت را بر هم زده است، و فرزند نیز خانه‌ای برای خود تهیه

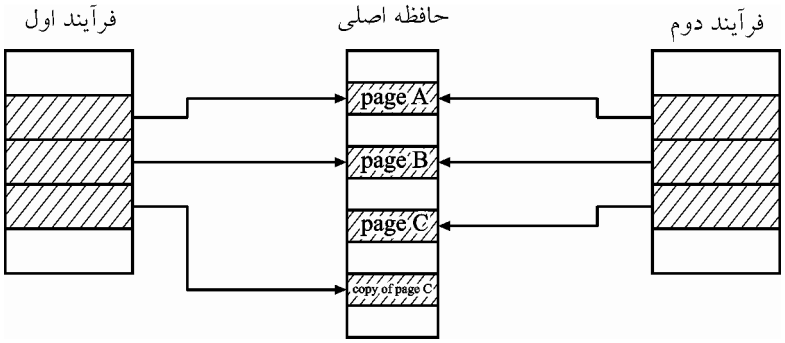
کرده است، حال خانه‌ی دوم پدر برای رسیدگی به امور شخصی بلااستفاده می‌ماند و این سربار است، زیرا دیگر شراکتی در کار نیست، فرزندی نیست، همان خانه‌ی اول برای پدر کافی بود. بهتر بود پدر صبر می‌کرد، تا اول فرزند تصمیم بگیرد، سپس بر اساس تصمیم فرزند، پدر نیز تصمیم خود را می‌گرفت.

حال مجدداً برگردید به وادی کامپیوتر، در صورتی که اگر بعد از اجرای فراخوانی سیستمی Fork، ابتدا فرآیند فرزند فراخوانی و اجرا شود، ممکن است در همان ابتدای اجرایش دستور ()exec را اجرا کند و در نتیجه از آن به بعد، از فضای حافظه‌ی شخصی خود استفاده کند، که در این صورت اگر فرآیند پدر بخواهد چیزی بر روی صفحات مشترک شده بنویسد، دیگر نیازی به کپی‌کردن آن صفحه نخواهد بود و این یعنی حذف سربار و افزایش کارایی.

با توجه به مطالب مطرح شده، گزینه چهارم درست و گزینه سوم نادرست خواهد بود.



قبل از اینکه فرآیند اول صفحه C را تغییر دهد.



بعد از اینکه فرآیند اول صفحه C را تغییر دهد.

۲- گزینه (۴) صحیح است.

به طور کلی میانگین زمان دسترسی سیستم به مقصد نهایی برای هر آدرس سیستم از رابطه زیر محاسبه می‌گردد:

$$T_{Access} = T_{Translation} + T_{Destination}$$

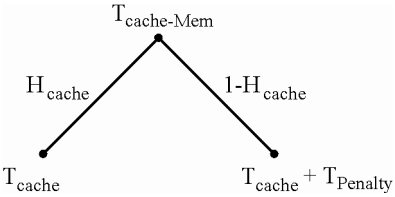
که پارامترهای آن شامل $T_{Translation}$ یعنی **بخش ترجمه** و $T_{Destination}$ یعنی **بخش مقصد** می‌باشد.

$T_{Translation}$: میانگین زمان ترجمه

توجه: مطابق فرض سوال در **بخش ترجمه** یعنی $T_{Translation}$، سه عنصر T_{TLB}، T_{Cache} و T_{Mem} برای ترجمه وجود دارد، بنابراین باید میانگین آنها محاسبه گردد.

برای محاسبه $T_{Translation}$ ابتدا میانگین T_{Cache} و T_{Mem} را محاسبه می‌کنیم به شکل $T_{Cache-Mem}$، در نهایت هم، میانگین $T_{Cache-Mem}$ و T_{TLB} را محاسبه می‌کنیم که همان $T_{Translation}$ است. به صورت زیر:

برای بدست آوردن $T_{Cache-Mem}$ درخت زیر را در نظر بگیرید:



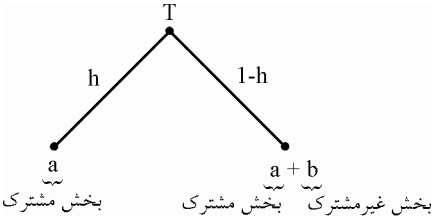
توجه: PAGE FRAME یا در Cache قرار دارد یا ندارد، اگر PAGE FRAME در Cache نباشد، پس باید به Cache آورده شود، که این زمان به Cache آوردن برابر $T_{Penalty}$ است. در واقع سیستم از Cache می‌خواند. بنابراین در درخت فوق به جای استفاده از T_{Mem} از $T_{Penalty}$ استفاده کردیم که شامل رفتن به Memory و آوردن به Cache است.

پس از ساده‌سازی درخت فوق رابطه زیر را برای محاسبه $T_{Cache-Mem}$ خواهیم داشت:

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

$$T_{Cache-Mem} = 10 + (1\text{-}0.9) \times 100 = 20 \text{ ns}$$

توجه: جهت ساده‌سازی رابطه درخت فوق، همواره می‌توان از **اتحاد درخت احتمال** به صورت زیر، استفاده نمود:



$$T = \text{بخش مشترک} + (1-h) \times \text{بخش غیرمشترک}$$

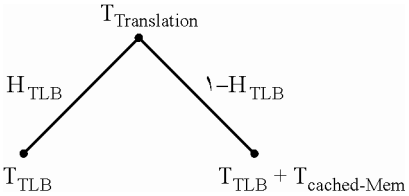
$$T = a + (1-h) \times b$$

پس از ساده‌سازی درخت مطرح شده براساس **اتحاد درخت احتمال**، رابطه زیر را بـرای محاسـبه $T_{Cache-Mem}$ خواهیم داشت:

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

$$T_{Cache-Mem} = 10 + (1-0.9) \times 100 = 20 \text{ ns}$$

در انتها برای بدست آوردن $T_{Translation}$ درخت زیر را در نظر بگیرید:



پس از ساده‌سازی درخت فوق رابطه زیر را برای محاسبه $T_{Translation}$ خواهیم داشت:

$$T_{Translation} = T_{TLB} + (1-H_{TLB}) \times T_{Cache-Mem}$$

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

$T_{Destination}$: میانگین زمان دسترسی به مقصد موردنظر

توجه: مطابق فرض سوال در **بخش مقصد** یعنی $T_{Destination}$، سه عنصر T_{Cache}، T_{Mem} و T_{Disk} برای دسترسی به مقصد مورد نظر وجود دارد، بنابراین باید میانگین آنها محاسبه گردد.

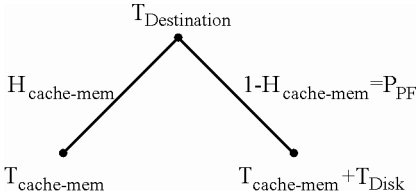
بـرای محاسـبه $T_{Destination}$ ابتـدا میـانگین T_{Cache} و T_{Mem} را محاسـبه مـی‌کنـیم بـه شـکل $T_{Cache-Mem}$، در نهایـت هـم، میـانگین $T_{Cache-Mem}$ و T_{Disk} را محاسـبه مـی‌کنـیم کـه همـان $T_{Destination}$ است. به صورت زیر:

مطابق آنچه پیشتر گفتیم، $T_{Cache-Mem}$ به صورت زیر محاسبه شد:

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

$$T_{Cache-Mem} = 10 + (1-0.9) \times 100 = 20 \text{ ns}$$

در انتها برای بدست آوردن $T_{Destination}$ درخت زیر را در نظر بگیرید:



پس از ساده‌سازی درخت فوق رابطه زیر را برای محاسبه $T_{Destination}$ خواهیم داشت:

$$T_{Destination} = T_{Cache-Mem} + (1-H_{Cache-Mem}) \times T_{Disk}$$

$T_{Destination} = T_{Cache\text{-}Mem} + (P_{PF}) \times T_{Disk}$

$T_{Destination} = 20 + (2\times10^{-6}) \times (10\times10^{6}) = 20+20 = 40\ ns$

و در نهایت براساس رابطه کل زمان دسترسی سیستم خواهیم داشت:

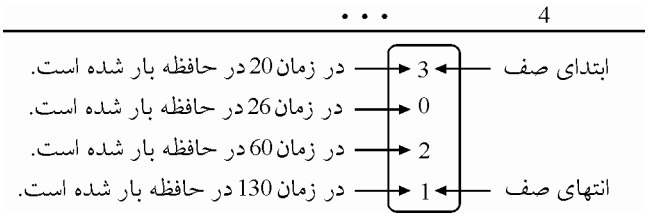
$T_{Access} = T_{Translation} + T_{Destination}$

$T_{Access} = 2.4+40 = 42.4ns \cong 43\ ns$

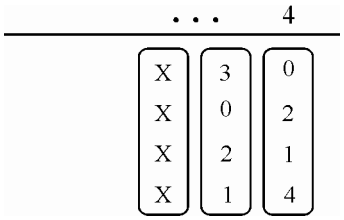
۳- گزینه(۳) صحیح است.

صفحه‌ای که به دلیل خطای نقص صفحه باید خارج گردد، مطابق الگوریتم FIFO براساس خروج به ترتیب ورود می‌باشد. بنابراین در این الگوریتم، زمان بارشدن صفحه در حافظه اهمیت دارد.

الگوی زیر را در نظر بگیرید:



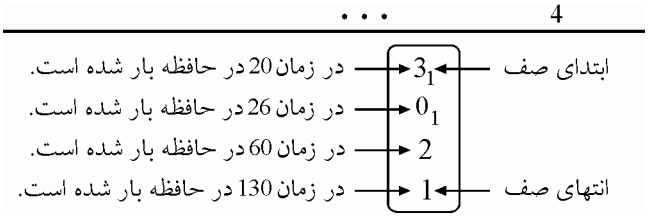
بنابراین صفحه شماره 3 برای خروج انتخاب می‌گردد، زیرا در ابتدای صف قرار دارد، پس از جایگزینی صفحه 4 با صفحه شماره 3، شکل حافظه به صورت زیر خواهد بود:



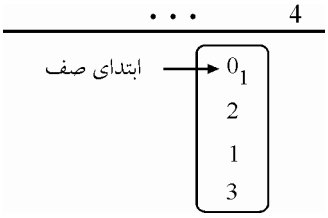
بنابراین گزینه سوم درست است.

صفحه‌ای که به دلیل خطای نقص صفحه باید خارج گردد، مطابق الگوریتم Clock همانند الگوریتم FIFO براساس خروج به ترتیب ورود می‌باشد. با این تفاوت که صفحه‌ای که برای خروج انتخاب می‌گردد، ابتدا بیت R آن بررسی می‌شود، اگر بیت R آن صفر بود، صفحه کاندید، از حافظه خارج می‌گردد، اما اگر صفحه‌ای که برای خروج انتخاب شده است، بیت R آن برابر یک باشد، ابتدا بیت R این صفحه کاندید برابر صفر شده و سپس به انتهای صف منتقل می‌گردد، در واقع شانس می‌آورد که خارج نمی‌شود چون یک شانس حضور دارد، سپس توسط جلو رفتن اشاره‌گر، نوبت به صفحه بعدی داده می‌شود که بیت R آن برای خروج بررسی می‌گردد و این روال تا پیدا کردن یک صفحه با بیت R برابر با صفر ادامه پیدا می‌کند.

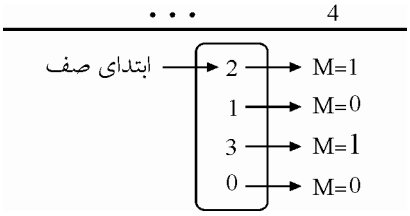
الگوی زیر را در نظر بگیرید:



در صورت سوال مطرح شده است که در حال حاضر اشاره‌گر روی صفحه شماره صفر قرار دارد، یعنی صفحه‌ی شماره 3 یک شانس دارد و به انتهای صف منتقل می‌گردد. حال مطابق فرض مسأله اشاره‌گر به صفحه‌ی شماره صفر در حال اشاره است.

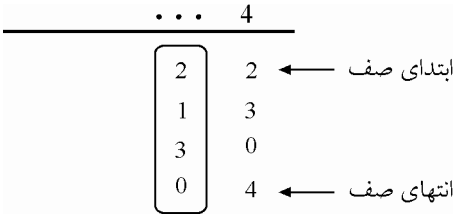


اما صفحه شماره‌ی صفر مقدار بیت R آن برابر یک می‌باشد، بنابراین مقدار آن صفر می‌شود و مجدداً اشاره‌گر یکی رو به جلو می‌رود. بنابراین داریم:



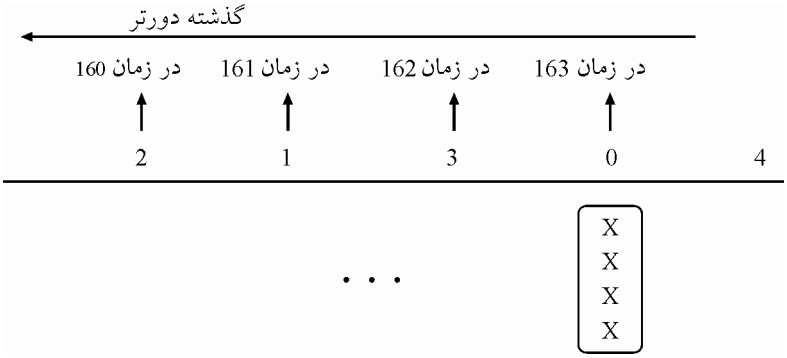
اگرچه برای صفحه شماره 2 بیت R برابر صفر است و طبق الگوریتم ساعت معمولی، این صفحه باید خارج گردد، اما دقت کنید که در صورت سوال، مطرح شده است که «الویت با انتخاب صفحه تغییر نیافته است (M=0)». در واقع با یک الگوریتم ساعت توسعه‌یافته مواجه هستیم که علاوه بر بیت R، بیت M را نیز برای خروج صفحه در نظر می‌گیرد. در واقع در این الگوریتم ساعت توسعه یافته از بین صفحات کاندید برای خروج با بیت R برابر با صفر، آن صفحه‌ای خارج می‌گردد که هم بیت R آن برابر صفر باشد و هم بیت M آن برابر صفر باشد. بنابراین مطابق مطالب بیان شده، صفحه‌ی شماره 1 نسبت به صفحه شماره 2 از اولویت بالاتری برای خروج برخوردار است. بنابراین صفحه شماره 1 به جای صفحه شماره 2 خارج می‌گردد. اگر یک دور می‌زدیم و هیچ

صفحه‌ای هر دو بیت R و M آن هر دو صفر نبود، همان صفحه شماره 2 خارج می‌گردید. پس از جایگزینی صفحه شماره 4 با صفحه شماره 1، شکل حافظه به صورت زیر خواهد بود:



صفحه‌ای که به دلیل خطای نقص صفحه باید خارج گردد، مطابق الگوریتم LRU، صفحه‌ای است که در گذشته‌ی دورتری مورد مراجعه قرار گرفته است. بنابراین در این الگوریتم، زمان آخرین مراجعه به صفحه اهمیت دارد.

الگوی زیر را در نظر بگیرید:



توجه: کاراکتر X در شکل فوق به این معنی است که شماره صفحات 0، 1، 2 و 3 به هر ترتیبی ممکن است وجود داشته باشد، اما به هر ترتیبی که باشند، به هنگام نقص صفحه، مطابق الگوریتم LRU، صفحه‌ای برای جایگزینی انتخاب می‌گردد، که در گذشته‌ی دورتری، مورد استفاده قرار گرفته باشد. بنابراین از آنجایی که صفحه‌ی شماره 2، در گذشته‌ی دورتری مورد استفاده قرار گرفته است، پس صفحه‌ی شماره 2 برای خروج انتخاب می‌گردد.

۴- گزینه (۱) صحیح است.

به طور کلی میانگین زمان دسترسی سیستم به مقصد نهایی برای هر آدرس سیستم از رابطه زیر محاسبه می‌گردد:

$$T_{Access} = T_{Translation} + T_{Destination}$$

که پارامترهای آن شامل $T_{Translation}$ یعنی **بخش ترجمه** و $T_{Destination}$ یعنی **بخش مقصد** می‌باشد.

توجه: در صورت سؤال مطرح شده است که، اگرچه می‌توان میانگین زمان دسترسی به یک آدرس حافظه مجازی را محاسبه نمود، ولی زمان‌های واقعی دسترسی‌ها متعدد هستند و با مقدار میانگین تفاوت دارند، در واقع خواسته سؤال، محاسبه تک به تک زمان دسترسی به یک آدرس حافظه مجازی است و نه میانگین زمان دسترسی به یک آدرس حافظه مجازی.

توجه: مطابق فرض سؤال در **بخش ترجمه** یعنی $T_{Translation}$، دو عنصر T_{TLB} و T_{Mem} برای ترجمه وجود دارد.

توجه: مطابق فرض سؤال در **بخش مقصد** یعنی $T_{Destination}$، دو عنصر T_{Mem} و T_{Disk} وجود دارد.

برخی از انواع زمان‌های دسترسی به یک آدرس حافظه مجازی به صورت زیر است:

حالت اول: صفحه مورد درخواست در حافظه است و آدرس آن در TLB موجود است.

$$T = T_{Translation} + T_{Destination}$$

$$T = T_{TLB} + T_{Mem} = 2ns + 50ns = 52ns$$

حالت دوم: صفحه مورد درخواست در حافظه است، اما آدرس آن در TLB نیست، در واقع تمام جدول صفحه در حافظه قرار دارد، بنابراین برای ترجمه نیز باید سراغ حافظه رفت، زیرا آدرس در جدول صفحه درون حافظه قرار دارد.

$$T = T_{Translation} + T_{Destination}$$

$$T = (T_{TLB}+T_{Mem}) + T_{Mem} = (2ns+50ns) + 50ns = 102ns$$

حالت سوم: صفحه مورد درخواست در حافظه نیست، یعنی نقص صفحه رخ می‌دهد، بنابراین صفحه مورد درخواست در دیسک قرار دارد که باید به حافظه منتقل شود، اما قاب خالی در حافظه موجود نیست، بنابراین یک صفحه باید از حافظه خارج گردد تا فضای کافی برای ورود صفحه مورد درخواست فراهم گردد. ولی صفحه‌ای که باید از حافظه خارج گردد، **تمیز** است، یعنی کپی صفحه موجود در حافظه با کپی صفحه موجود در دیسک یکی است، یعنی صفحه تغییر نکرده است. بنابراین نیاز به دوباره نویسی ندارد. یعنی صفحه جدید بر روی صفحه کاندید برای خروج جایگزین می‌گردد، بدون آنکه نیاز باشد صفحه کاندید برای خروج، اطلاعاتش بر روی دیسک ذخیره گردد.

$$T = T_{Translation} + T_{Destination}$$

$$T = (T_{TLB} + T_{Mem}) + (T_{Mem} + T_{Disk}) = (2ns + 50ns) + (50ns + 3ms) \cong 3ms$$

توجه: مطابق فرض مسأله از اعداد بسیار کوچک در مقابل اعداد بسیار بزرگ صرف‌نظر شده است.

حالت چهارم: صفحه مورد درخواست در حافظه نیست، یعنی نقص صفحه رخ می‌دهد، بنابراین صفحه مورد درخواست در دیسک قرار دارد که باید به حافظه منتقل شود، اما قاب خالی در حافظه هم موجود است. بنابراین صفحه مورد درخواست در قاب خالی قرار می‌گیرد.

$T = T_{Translation} + T_{Destination}$

$T = (T_{TLB}+T_{Mem}) + (T_{Mem}+T_{Disk}) = (2ns+50ns) + (50ns+3ms) \cong 3ms$

حالت پنجم: صفحه مورد درخواست در حافظه نیست، یعنی نقص صفحه رخ می‌دهد، بنابراین صفحه مورد درخواست در دیسک قرار دارد که باید به حافظه منتقل شود، اما قاب خالی در حافظه موجود نیست، بنابراین یک صفحه باید از حافظه خارج گردد تا فضای کافی برای ورود صفحه مورد درخواست فراهم گردد. ولی صفحه‌ای که باید از حافظه خارج گردد، **کثیف** است، یعنی کپی صفحه موجود در حافظه با کپی صفحه موجود در دیسک یکی نیست، یعنی صفحه تغییر کرده است. بنابراین نیاز به دوباره نویسی دارد. یعنی قبل از آنکه صفحه جدید بر روی صفحه کاندید برای خروج جایگزین گردد، ابتدا اطلاعات صفحه کاندید برای خروج بر روی دیسک ذخیره می‌گردد که یک $T_{RW} = T_{Disk}$ زمان نیاز دارد، سپس صفحه جدید بر روی صفحه کاندید برای خروج جایگزین می‌گردد، که مجدداً یک T_{Disk} زمان نیاز دارد.

$T = T_{Translation} + T_{Destination}$

$T = (T_{TLB}+T_{Mem}) + (T_{Mem}+T_{RW}+T_{Disk}) = (2ns+50ns) + (50ns+ 3ms+3ms) \cong 6ms$

۵- گزینه (۱) صحیح است.

در کامپیوترها فقط صرف اجرای برنامه‌ها در دستور کار قرار ندارد، بلکه اجرای امن برنامه‌ها در دستور کار قرار دارد، امن بودن به این معنی است که اجرای برنامه‌ها باعث مختل شدن بخشی از سیستم کامپیوتری یا تمام سیستم نگردد. بهتر بود در صورت سؤال جمله اول به شکل زیر مطرح می‌شد:

راه‌حل‌های سخت‌افزاری متعددی به همراه سیستم عامل به اجرای **امن** برنامه‌ها کمک می‌کند. اما اینکه وجود سخت‌افزار برای اجرا لازم است، کاملاً بدیهی است. ولی آیا سخت‌افزار برای برقراری امنیت هم کاربرد دارد؟ مگر سیستم عامل به تنهایی و بدون پشتیبانی سخت‌افزار قادر به برقراری امنیت نیست؟ بدیهی است که خیر، زیرا وقتی سیستم عامل پردازنده را به یک برنامه کاربر می‌سپارد تا آن را اجرا کند، برای مدتی هیچ برنامه‌ای جز برنامه مورد نظر اجرا نمی‌شود. بنابراین سیستم عامل که خودش یک برنامه است و تا اجرا نشود کاری نمی‌تواند بکند، چگونه ممکن است در لحظه وقوع نقص حفاظتی از این امر مطلع شود و برای جلوگیری از آن کاری بکند، مگر آنکه سخت‌افزار حفاظت این نقص‌ها را کشف کرده و سیستم عامل را وارد صحنه کند.

نتیجه اینکه سخت‌افزار هم برای اجرای برنامه‌ها، به برنامه‌ها کمک می‌کند و هم برای برقراری امنیت اجرای برنامه‌ها، به برنامه‌ها کمک می‌کند، در این صورت یک برنامه، فقط اجرا نمی‌شود که هر کاری که دلش می‌خواهد انجام دهد و به تمام بخش‌های کامپیوتر دسترسی داشته باشد، بلکه با این تفاسیر، برنامه می‌تواند اجرا شود، اما باید اجرای **امن** شود، یعنی در یک چارچوب از قبل مشخص شده و مبتنی بر قوانین سیستم اجرا شود، در واقع سخت‌افزار کمک می‌کند تا برنامه‌ها

اجرا شوند، اما هرجا که لازم باشد و هرجا که برنامه پا را از گلیم بیشتر دراز کند، آنگاه سخت‌افزار جلوی اجرای برنامه خاطی را می‌گیرد و پردازنده را هم از او می‌گیرد و سیستم عامل را وارد صحنه می‌کند تا به امور برنامه خاطی رسیدگی کند.

انسان حرف‌های خود را توسط برنامه و به واسطه زبان‌های برنامه‌نویسی، برای اجرا به کامپیوتر می‌گوید. اما این کامپیوتر ننشسته است که هرچه برنامه بگوید، چشم بسته، اجرا کند. عملیات کامپیوتر حساب دارد، کتاب دارد، در واقع هر سیستمی نیاز به رسیدگی و کنترل دارد، وگرنه هرج و مرج می‌شود و کارکرد سیستم مختل می‌گردد.

در سیستم‌های کامپیوتری، مسئولیت خواندن و اجرای دستورات برنامه کاربر را مجموعه پردازنده مرکزی برعهده گرفته است. حال این پردازنده خط به خط برنامه‌های کاربر را می‌خواند، اگر حرف خوب بود اجرا می‌کند و اگر حرف بد بود اجرا نمی‌کند. حرف خوب از نگاه پردازنده، کار مجاز (کارغیرممتاز) است و حرف بد از نگاه پردازنده کار غیرمجاز (کار ممتاز) است. اگر کار غیرمجاز دید، بزرگتر خود، سیستم عامل را وارد صحنه می‌کند.

توجه: از آنجا که در هنگام اجرای برنامه کاربر، سیستم عامل حضور ندارد، بنابراین کشف و تشخیص هرگونه کار غیرمجاز (انجام کار ممتاز) به واسطه برنامه کاربر، توسط پردازنده و سخت‌افزار انجام می‌شود.

توجه: به طور کلی، تمامی کارهای حمایتی که در هنگام اجرای برنامه‌های کاربر انجام می‌گردد، همه سخت‌افزاری هستند، چون در لحظه اجرای برنامه کاربر، سیستم عامل اصلاً حضور ندارد.

توجه: عملیات ورودی و خروجی، عمل تقسیم بر صفر، تغییر تایمر سیستم، تغییر ثبات‌های حفاظتی، از کار انداختن وقفه، دسترسی به حافظه سیستم عامل، دسترسی بدون اجازه به حافظه فرآیندهای دیگر نمونه‌هایی ازکارهای ممتاز محسوب می‌گردند. در یک بیان کلی‌تر، هرکاری که استفاده نادرست ازآن باعث مختل شدن بخشی از سیستم کامپیوتری یا کل سیستم کامپیوتری گردد، کار ممتاز به حساب می‌آید.

توجه: انجام کار ممتاز برای یک فرآیند کاربر، کار غیرمجاز است و فقط انجام کار غیرممتاز است که برای یک فرآیند کاربر، مجاز محسوب می‌شود.

توجه: هرگاه یک فرآیند کاربر قصد اجرای یک کار ممتاز را توسط پردازنده داشته باشد، پردازنده فوراً توسط یک وقفه، سیستم عامل را باخبر می‌کند و کنترل به سیستم عامل واگذار می‌گردد تا سیستم عامل در پاسخ به آن وقفه یک روال پاسخ به وقفه را اجرا کند.

توجه: دقت کنید که در هر لحظه فقط یک برنامه قادر به اجرا توسط پردازنده است، در واقع در لحظه‌ای که برنامه کاربر در حال اجرا است و تصمیم به انجام یک کار ممتاز می‌گیرد، سیستم عامل اصلاً در حال اجرا نیست که بخواهد جلوی این کار غیرمجاز فرآیند در حال اجرا را بگیرد، اما سیستم عامل این مسأله را قبلاً با پردازنده و سخت‌افزار هماهنگ کرده است، پردازنده و سخت‌افزار در غیاب سیستم عامل، به محض مشاهده یک کار غیرمجاز از فرآیند کاربر در حال

اجرا، فوراً توسط صدور یک وقفه، سیستم عامل را وارد صحنه می‌کند. در واقع بـه واسـطه وقفـه، پردازنده از فرآیند خاطی گرفته شده و در اختیار سیستم عامل قرار داده می‌شود تـا سیسـتم عامـل بتواند به امور فرآیند خاطی رسیدگی کند.

توجه: در سطح پردازنده دو نوع کار انجام می‌گردد، (۱) کارهای ممتـاز، (۲) کارهـای غیـر ممتـاز، کارهای ممتاز در مود هسته پردازنده و کارهای غیرممتـاز در مـود کـاربر پردازنـده اجـازه‌ی اجـرا دارند. بنابراین هیچگاه یک کار ممتاز اجازه اجرا در مود کاربر را نخواهد داشت.

کارهای حمایتی سخت‌افزار در هنگام اجرای برنامه کاربر (حینا اجرا) عبارتند از:

- در هنگام اجرای برنامه کاربر از TBL استفاده می‌گردد، کـه در آن لحظـه سیسـتم عامـل حضور ندارد، اما سخت‌افزار (بخش MMU) حمایت و کنترل مـی‌کنـد. بنـابراین کنتـرل جدول TLB در هنگام اجرای یک برنامه کاربر بر عهده‌ی سخت‌افزار است.
- در هنگام اجرای برنامه کاربر نیاز به نگاشت آدرس منطقی به فیزیکـی اسـت، کـه در آن لحظه سیستم عامل حضور نـدارد، امـا سـخت‌افـزار (بخـش MMU) حمایـت و کنتـرل می‌کند. بنابراین تبدیل آدرس منطقی به فیزیکی در هنگام اجرای یـک برنامـه کـاربر، بـر عهده‌ی سخت‌افزار است.
- در هنگام اجرای برنامه کاربر ممکن است، نیاز به ثبت مقدار 1 در بیـت Modified یـک صفحه باشد، که در آن لحظه سیستم عامل حضور ندارد، اما سخت‌افزار (بخـش MMU) حمایت و کنترل می‌کند. بنابراین ثبت تغییر یافتن (Modify) یک صفحه در هنگام اجرای یک برنامه کاربر بر عهده سخت‌افزار است.
- در هنگام اجرای برنامه کاربر ممکن است یک کار غیرمجاز از سوی برنامه انجـام شـود، که در آن لحظه سیستم عامل حضور ندارد، اما سخت‌افـزار (مـود حفاظـت شـده CPU) حمایت و کنتـرل مـی‌کنـد. بنـابراین کشـف و تشـخیص کـار غیرمجـاز (دسـتورالعمل غیرمجاز) یک برنامه کاربر در حال اجرا، برعهده‌ی سخت‌افزار است.
- در صورت وجود برش زمانی در سیستم، در هنگام اجرای برنامه کاربر، لازم است پایـان برش زمانی به برنامه اعـلام شـود، کـه در آن لحظـه سیسـتم عامـل حضـور نـدارد، امـا سخت‌افزار حمایت و کنترل می‌کند (توسط Timer). بنابراین اعلام پایان برش زمـانی در هنگام اجرای یک برنامه کاربر، بر عهده سخت‌افزار است.
- در هنگام اجرای برنامه کاربر از حافظه پنهان (Cache) استفاده می‌گردد، که در آن لحظـه سیستم عامل حضور ندارد، اما سخت‌افـزار (بخـش CCU) حمایـت و کنتـرل مـی‌کنـد. بنابراین کنترل حافظه‌ی پنهان (Cache) در هنگام اجرای یک برنامـه کـاربر، بـر عهـده‌ی سخت‌افزار است.
- در هنگام اجرای برنامه‌ی کاربر ممکن اسـت نقـص صـفحه رخ دهـد، کـه در آن لحظـه سیستم عامل حضور ندارد، اما سخت‌افزار (بخـش MMU) حمایـت و کنتـرل مـی‌کنـد.

بنابراین کشف و تشخیص نقص صفحه در هنگام اجـرای یـک برنامـه کـاربر برعهـده‌ی سخت‌افزار است.

- در هنگام اجرای برنامه کاربر ممکـن اسـت، نیـاز بـه ثبـت مقـدار 1 در بیـت دسترسـی (Referenced Access) یک صفحه باشد، که در آن لحظه سیستم عامل حضور ندارد، اما سخت‌افزار (بخش MMU) حمایت و کنترل می‌کند، بنـابراین ثبـت دسترسـی (Access) یک صفحه در هنگام اجرای یک برنامه‌ی کاربر، برعهده‌ی سخت‌افزار است.
- در هنگام اجرای برنامه کاربر ممکن است، سرریزی (Over Flow) در محاسبات پردازنده رخ دهد، که در آن لحظه سیستم عامل حضور ندارد، اما سـخت‌افـزار حمایـت و کنتـرل می‌کند، بنابراین کشف و تشخیص سرریزی (Over Flow) در هنگام اجرای یک برنامه‌ی کاربر، برعهده‌ی سخت‌افزار است.

توجه: پس از این کشف و تشخیص سرریزی (Over Flow) توسط سخت‌افزار، سخت‌افزار توسط یک وقفه سیستم عامل را باخبر و وارد صحنه می‌کند، تا به ایـن مسـأله توسـط روال پاسـخ وقفـه، پاسخ دهد. در واقع برنامه کاربر دچار یک کار غیرمجاز شده اسـت، کـه مـی‌بایسـت سـخت‌افـزار توسط یک وقفه و گرفتن پردازنده از برنامه کاربر، سیستم عامل را با خبر و وارد صحنه کند، تا بـه امور برنامه خاطی رسیدگی کند.

- در هنگام اجرای برنامه کاربر نیاز به نگاشت آدرس منطقی (آدرس قطعه‌بنـدی شـده) بـه فیزیکی (آدرس حافظه‌ی اصلی) است، که در آن لحظه سیستم عامل حضـور نـدارد، امـا سخت‌افزار (بخش MMU) حمایت و کنترل می‌کند. بنـابراین تبـدیل آدرس منطقـی بـه فیزیکی در هنگام اجرای یک برنامه کاربر، بر عهده‌ی سخت‌افزار است.
- در هنگام اجرای برنامه کاربر ممکن است یک کـار تعریـف نشـده و غیرمجـاز از سـوی برنامه کاربر انجام شود، که در آن لحظه سیستم عامل حضور ندارد، اما سخت‌افزار (مـود حفاظت شده CPU) حمایت و کنترل می‌کند. بنابراین کشف، تشـخیص و جلـوگیری از اجرای کار و دستورالعمل تعریف نشده و غیرمجـاز برنامـه‌ی کـاربر در حـال اجـرا، بـر عهده‌ی سخت‌افزار است.
- در هنگام اجرای برنامه ممکن است، برنامه بخواهد پا را از گلیم خود فراتر قـرار دهـد و به فضای حافظه‌ی سایر فرآیندها سَرَک بکشد، که یـک کـار غیرمجـاز اسـت، کـه در آن لحظه سیستم عامل حضور نـدارد، امـا سـخت‌افـزار (بخـش MMU) حمایـت و کنتـرل می‌کند. بنابراین حفاظت از فضای حافظه‌ی سایر فرآیندها در قبال درخواسـت دسترسـی غیرمجاز یک برنامه کاربر در حال اجرا، بر عهده‌ی سخت‌افزار است.

تا به اینجا به این نتیجه رسیدیم که سیستم عامل در لحظه‌ی اجرای یک برنامه کاربر حمـایتی را در قبال برنامه کاربر در حال اجرا انجام نمی‌دهد، مگر سخت‌افزار توسط تولید وقفه، درخواست ورود سیستم عامل را داشته باشد که در این صورت برنامه کاربر در حال اجرا پردازنده را واگذار می‌کنـد

و سیستم عامل وارد صحنه می‌گردد. بنابراین مسئولیت **کارهای حمایتی**، در قبال برنامه در حال اجرا (حینا اجرا) را، **سخت افزار** بر عهده می‌گیرد. اما مسئولیت **کارهای حمایتی**، قبل (پیشا اجرا) و بعد از اجرای برنامه کاربر (پسا اجرا) را، **سیستم عامل** بر عهده می‌گیرد.

کارهای حمایتی سیستم عامل قبل و بعد از اجرای برنامه کاربر (پیشا اجرا و پسا اجرا) عبارتند از:

- قبل از اجرای برنامه کاربر، ثبات‌های مدیریت حافظه (بخش MMU) و تمام ثبات‌های پردازنده بر اساس PCB فرآیند، بار می‌شوند، در این لحظه سیستم عامل حضور دارد و حمایت و کنترل می‌کند. بنابراین بار کردن ثبات‌های مدیریت حافظه و تمام ثبات‌های پردازنده قبل از اجرای برنامه کاربر، برعهده‌ی سیستم عامل است. بنابراین گزینه‌ی دوم نادرست است.

توجه: بار کردن ثبات‌های مدیریت حافظه و پردازنده در بخش Dispatcher سیستم عامل صورت می‌گیرد.

- بعد از اجرای برنامه کاربر و متوقف شدن برنامه کاربر به دلیل وقفه‌های نرم‌افزاری یا سخت‌افزاری، مانند پایان برش زمانی، پردازنده از برنامه کاربر گرفته می‌شود و به سیستم عامل واگذار می‌شود تا سیستم عامل پاسخ مناسب را ارائه کند. در این حالت سیستم عامل بر اساس شرایط، برنامه کاربر را در **حالت مسدود** (مثلاً به دلیل عملیات ورودی و خروجی) و یا در **حالت آماده** (به دلیل پایان برش زمانی) قرار می‌دهد. همچنین سیستم عامل پس از اتمام عملیات ورودی و خروجی برنامه کاربر، برنامه را در صف آماده قرار می‌دهد و یا قبل از اجرای برنامه، سیستم عامل توسط زمان‌بند کوتاه مدت یک برنامه را از صف آماده برای در اختیار گرفتن پردازنده و عملیات اجرا، انتخاب می‌کند. بنابراین گزینه سوم نادرست است.
- بعد از اجرای برنامه و توقف برنامه کاربر به دلیل درخواست خروج توسط خود برنامه (اجرای یک فراخوان سیستمی)، توسط صدور یک وقفه نرم‌افزاری به سیستم عامل، سیستم عامل فرآیند مورد نظر را از سیستم خارج می‌کند.

توجه: کشف و تشخیص این وقفه توسط سخت‌افزار است اما خروج فرآیند توسط سیستم عامل انجام می‌گردد. بنابراین گزینه سوم نادرست است.

- بعد از اجرای برنامه و توقف برنامه کاربر به دلیل نقص صفحه (Page Fault)، سیستم عامل عملیات جستجوی صفحه مورد نظر را بر روی دیسک آغاز می‌کند و سپس صفحه یافت شده را در حافظه قرار می‌دهد.

توجه: کشف و تشخیص نقص صفحه توسط سخت‌افزار است، اما جستجوی صفحه مورد نظر بر روی دیسک و انتقال از دیسک به حافظه توسط سیستم عامل انجام می‌گردد. بنابراین گزینه چهارم نادرست است.

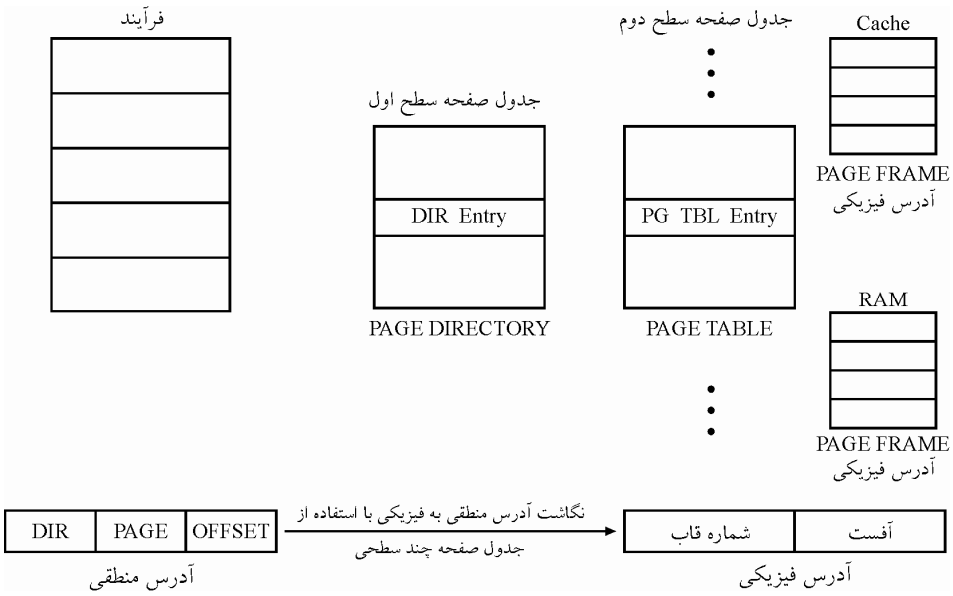
توجه: MMU سرواژه عبارت Memory Mangement Unit و به معنی واحد مدیریت حافظه است.

توجه: CCU سرواژه عبارت Cache Control Unit و به معنی واحد کنترل حافظه نهان است.

۶- گزینه (۴) صحیح است.

آدرس منطقی به صورت زیر است:

مطابق فرض مسأله، برای نگاشت آدرس منطقی به فیزیکی از جدول صفحه چند سطحی (دو سطحی) استفاده شده است.



به طور کلی میانگین زمان دسترسی سیستم به مقصد نهایی برای هر آدرس سیستم از رابطه زیر محاسبه می‌گردد:

$$T_{Access} = T_{Translation} + T_{Destination}$$

که پارامترهای آن شامل $T_{Translation}$ یعنی **بخش ترجمه** و $T_{Destination}$ یعنی **بخش مقصد** می‌باشد.

توجه: در صورت سؤال مطرح شده است که، مقدار زمان‌های ممکن برای دسترسی به یک مکان حافظه که با آدرس منطقی مشخص شده است، چقدر است؟ در واقع خواسته سؤال، محاسبه تک به تک زمان دسترسی به یک آدرس حافظه منطقی (مجازی) است و نه میانگین زمان دسترسی به یک آدرس حافظه منطقی.

۱- اگر تعدادی از درایه‌های **مرتبط** دو جدول PAGE DIRECTORY و PAGE TABLE داخل TLB باشد.

توجه: در سیستم جداول صفحه چندسطحی درایه‌های موجودی که در TLB قرار دارند، شامل اطلاعات مرتبط همه سطوح است، نمی‌شود سطحی باشد و سطحی نباشد، در سؤال مذکور یا تعدادی از درایه‌های مرتبط دو جدول PAGE DIRECTORY و PAGE TABLE به طور همزمان داخل TLB هستند و یا به طور همزمان داخل TLB نیستند.

الف) مقصد یعنی PAGE FRAME در Cache باشد:

$$T=T_{Translation} + T_{Destination}$$

$$T= T_{TLB} + T_{Cache} = 2+10=12ns$$

ب) مقصد یعنی PAGE FRAME در Cache نباشد:

پس باید به Cache آورده شود، که این زمان به Cache آوردن برابر $T_{Cache\text{-}Miss\text{-}Penalty}$ است.

$$T=T_{Translation} + T_{Destination}$$

$$T= T_{TLB} + (T_{Cache} + T_{Cache\text{-}Miss\text{-}Penalty}) = 2+(10+100)=112ns$$

۲- اگر هیچ یک از درایه‌های **مرتبط** دو جدول PAGE DIRECTORY و PAGE TABLE داخل TLB نباشد.

الف) جداول PAGE DIRECTORY و PAGE TABLE و مقصد یعنی PAGE FRAME در Cache باشند:

$$T=T_{Translation} + T_{Destination}$$

$$T= (T_{TLB} + T_{Cache\text{-}DIR} + T_{Cache\text{-}PAGE\text{-}TABLE}) + T_{Cache\text{-}FRAME}$$

$$T=(2+10+10)+10=32ns$$

توجه: حال اگر در ادامه هر یک از جداول PAGE DIRECTORY و PAGE TABLE و یا مقصد PAGE FRAME در Cache نباشند، باید به Cache آورده شوند، که این زمان به Cache آوردن برابر $T_{Cache\text{-}Miss\text{-}Penalty}$ است.

ب) یک مورد از سه مورد فوق در cache نباشد:

$$T=32+1\times T_{Cache\text{-}Miss\text{-}Penalty}=32+100=132ns$$

ج) دو مورد از سه مورد فوق در Cache نباشد:

$$T=32+2\times T_{Cache\text{-}Miss\text{-}Penalty}=32+2\times100=232ns$$

د) سه مورد از سه مورد فوق در Cache نباشد:

$T=32+3\times T_{Cache\text{-}Miss\text{-}Penalty}=32+3\times100=332ns$

۳- اگر نقص صفحه رخ دهد و مقصد یعنی PAGE FRAME در دیسک باشد.

توجه: مطابق گزینه‌ها از اعداد بسیار کوچک در مقابل اعداد بسیار بزرگ صرفه نظر شده است.

الف) صفحه مورد درخواست در دیسک قرار دارد که باید به حافظه و در نهایت به Cache منتقل شود، اما قاب خالی در حافظه موجود نیست، بنابراین یک صفحه باید از حافظه خارج گردد، تا فضای کافی برای ورود صفحه مورد درخواست فراهم گردد. ولی صفحه‌ای که باید از حافظه خارج گردد، **تمیز** است، یعنی کپی صفحه موجود در حافظه با کپی موجود در دیسک یکی است، یعنی صفحه تغییر نکرده است، بنابراین نیاز به دوباره نویسی ندارد. یعنی صفحه جدید بر روی صفحه کاندید برای خروج، جایگزین می‌گردد، بدون آنکه صفحه کاندید برای خروج، اطلاعاتش بر روی دیسک ذخیره گردد.

$T=T_{Translation}+T_{Destination}\cong T_{Disk}=5ms$

ب) صفحه مورد درخواست در دیسک قرار دارد که باید به حافظه و در نهایت به Cache منتقل شود، اما قاب خالی در حافظه موجود است. بنابراین صفحه مورد درخواست در قاب خالی قرار می‌گیرد.

$T=T_{Translation}+T_{Destination}\cong T_{Disk}=5ms$

ج) صفحه مورد درخواست در دیسک قرار دارد که باید به حافظه و در نهایت به Cache منتقل شود، اما قاب خالی در حافظه موجود نیست، بنابراین یک صفحه باید از حافظه خارج گردد تا فضای کافی برای ورود صفحه مورد درخواست فراهم گردد. ولی صفحه‌ای که باید از حافظه خارج گردد، **کثیف** است، یعنی کپی صفحه‌ی موجود در حافظه با کپی صفحه موجود در دیسک یکی نیست، یعنی صفحه تغییر کرده است، بنابراین نیاز به دوباره‌نویسی دارد. یعنی قبل از آنکه صفحه جدید بر روی صفحه‌ی کاندید برای خروج جایگزین گردد، ابتدا اطلاعات صفحه کاندید برای خروج بر روی دیسک ذخیره می‌گردد که یک $T_{RW}=T_{Disk}$ زمان نیاز دارد، سپس صفحه جدید بر روی صفحه کاندید برای خروج، جایگزین می‌گردد، که مجددا یک T_{Disk} زمان نیاز دارد.

$T=T_{Translation}+T_{Destination}\cong T_{RW}+T_{Disk}=10ms$

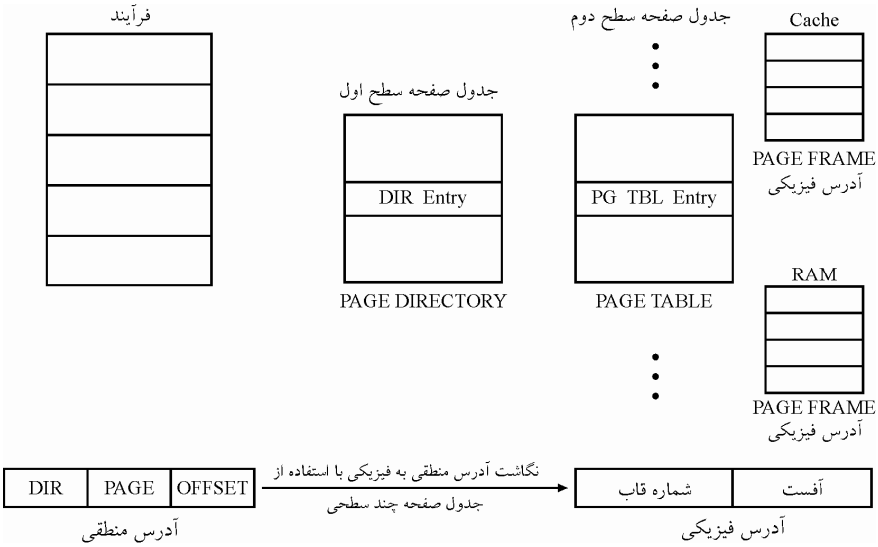
۷- گزینه(۴) صحیح است.

آدرس منطقی به صورت زیر است:

DIR	PAGE	OFFSET

آدرس منطقی

مطابق فرض مساله، برای نگاشت آدرس منطقی به فیزیکی از جداول صفحه چند سطحی (دوسطحی) استفاده شده است.



به طور کلی میانگین زمان دسترسی سیستم به مقصد نهایی برای هر آدرس سیستم از رابطه زیر محاسبه می‌گردد:

$$T_{Access} = T_{Translation} + T_{Destination}$$

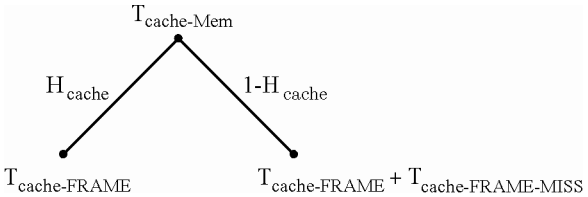
که پارامترهای آن شامل $T_{Translation}$ یعنی **بخش ترجمه** و $T_{Destination}$ یعنی **بخش مقصد** می‌باشد.

$\mathbf{T_{Translation}}$: میانگین زمان ترجمه

توجه: مطابق فرض سوال در **بخش ترجمه** یعنی $T_{Translation}$، سه عنصر T_{TLB}، T_{Cache} و T_{Mem} برای ترجمه وجود دارد، بنابراین باید میانگین آنها محاسبه گردد.

برای محاسبه $T_{Translation}$ ابتدا میانگین T_{Mem} و T_{Cache} را محاسبه می‌کنیم به شکل $T_{Cache-Mem}$، در نهایت هم، میانگین $T_{Cache-Mem}$ و T_{TLB} را محاسبه می‌کنیم که همان $T_{Translation}$ است. به صورت زیر:

برای بدست آوردن $T_{Cache-Mem}$ درخت زیر را در نظر بگیرید:



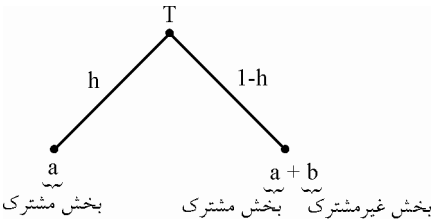
توجه: PAGE FRAME یا در Cache قرار دارد یا ندارد، اگر PAGE FRAME در Cache نباشد، پس باید به Cache آورده شود، که این زمان به Cache آوردن برابر $T_{Cache\text{-}FRAME\text{-}MISS}$ است. در واقع سیستم از Cache می‌خواند. بنابراین در درخت فوق به جای استفاده از T_{Mem} از $T_{Cache\text{-}FRAME\text{-}MISS}$ استفاده کردیم که شامل رفتن به Memory و آوردن به Cache است.

پس از ساده‌سازی درخت فوق رابطه زیر را برای محاسبه $T_{Cache\text{-}Mem}$ خواهیم داشت.

$$T_{Cache\text{-}mem} = T_{Cache\text{-}FRAME} + (1- H_{Cache}) \times T_{Cache\text{-}FRAME\text{-}MISS}$$

$$T_{Cache\text{-}mem} = 10+(1\text{-}0.9)\times 100=20\ ns$$

توجه: جهت ساده‌سازی رابطه درخت فوق، همواره می‌توان از **اتحاد درخت احتمال** به صورت زیر، استفاده نمود:



$$T = \text{بخش مشترک} + (1-h)\times \text{بخش غیرمشترک}$$

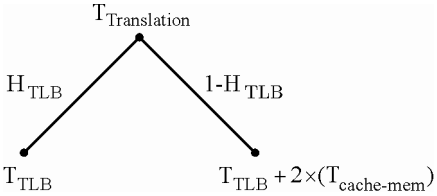
$$T = a + (1-h)\times b$$

پس از ساده‌سازی درخت مطرح شده براساس **اتحاد درخت احتمال**، رابطه زیر را برای محاسبه $T_{Cache-Mem}$ خواهیم داشت:

$$T_{Cache\text{-}mem} = T_{Cache\text{-}FRAME} + (1- H_{Cache}) \times T_{Cache\text{-}FRAME\text{-}MISS}$$

$$T_{Cache\text{-}mem} = 10+(1\text{-}0.9)\times 100=20\ ns$$

در انتها برای بدست آوردن $T_{Translation}$ درخت زیر را در نظر بگیرید:



توجه: وجود ضریب 2 در پشت $T_{Cache\text{-}mem}$ به این دلیل است که برای تولید آدرس فیزیکی، به دو آدرس DIR در جدول صفحه جزئی سطح اول، یعنی PAGE DIRECTORY و آدرس PAGE در جدول صفحه جزئی سطح دوم، یعنی PAGE TABLE نیاز داریم. بنابراین به 2 بار $T_{Cache\text{-}mem}$ و

مراجعه به Cache نیاز است. یعنی یک $T_{Cache\text{-}mem}$ برای مراجعه به جدول صفحه جزئی سطح اول و یک $T_{Cache\text{-}mem}$ دیگر برای مراجعه به جدول صفحه جزئی سطح دوم.

توجه: در سیستم جداول صفحه چندسطحی درایه‌های موجودی که در TLB قرار دارند، شامل اطلاعات مرتبط همه سطوح است، نمی‌شود سطحی باشد و سطحی نباشد، در سؤال مذکور یا تعدادی از درایه‌های مرتبط دو جدول PAGE DIRECTORY و PAGE TABLE به طور همزمان داخل TLB هستند و یا به طور همزمان داخل TLB نیستند.

پس از ساده‌سازی درخت فوق رابطه زیر را برای محاسبه $T_{Translation}$ خواهیم داشت:

$$T_{Translation} = T_{TLB} + (1 - H_{TLB}) \times 2 \times (T_{Cache\text{-}mem})$$

$$T_{Translation} = 2 + (1 - 0.98) \times 2 \times (20) = 2.8 \text{ ns}$$

$T_{Destination}$: میانگین زمان دسترسی به مقصد موردنظر

توجه: مطابق فرض سوال در **بخش مقصد** یعنی $T_{Destination}$، سه عنصر T_{Cache}، T_{Mem} و T_{Disk} برای دسترسی به مقصد مورد نظر وجود دارد، بنابراین باید میانگین آنها محاسبه گردد.

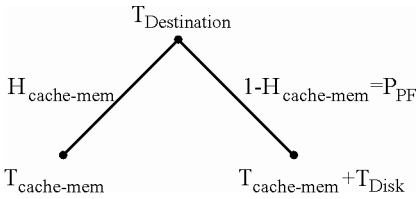
برای محاسبه $T_{Destination}$ ابتدا میانگین T_{Cache} و T_{Mem} را محاسبه می‌کنیم به شکل $T_{Cache-Mem}$، در نهایت هم، میانگین $T_{Cache-Mem}$ و T_{Disk} را محاسبه می‌کنیم که همان $T_{Destination}$ است. به صورت زیر:

مطابق آنچه پیشتر گفتیم، $T_{Cache\text{-}Mem}$ به صورت زیر محاسبه شد:

$$T_{Cache\text{-}mem} = T_{Cache\text{-}FRAME} + (1 - H_{Cache}) \times T_{Cache\text{-}FRAME\text{-}MISS}$$

$$T_{Cache\text{-}mem} = 10 + (1 - 0.9) \times 100 = 20 \text{ ns}$$

در انتها برای بدست آوردن $T_{Destination}$ درخت زیر را در نظر بگیرید:



پس از ساده‌سازی درخت فوق رابطه زیر را برای محاسبه $T_{Destination}$ خواهیم داشت:

$$T_{Destination} = T_{Cache\text{-}Mem} + (1 - H_{Cache\text{-}Mem}) \times T_{Disk}$$

$$T_{Destination} = T_{Cache\text{-}Mem} + (P_{PF}) \times T_{Disk}$$

$$T_{Destination} = 20 + (10^{-6}) \times (5 \times 10^{6}) = 20 + 5 = 25 \text{ ns}$$

و در نهایت براساس رابطه کل زمان دسترسی سیستم خواهیم داشت:

$$T_{Access} = T_{Translation} + T_{Destination}$$

$T_{Access} = 2.8+25=27.8\ ns \cong 28\ ns$

۸- گزینه (۲) صحیح است.

تکنیک‌های برنامه‌نویسی و ساختارهای داده‌ای که اصل مراجعات محلی را رعایت می‌کنند، برای صفحه‌بندی بر پایه تقاضا مناسب هستند.

مانند پشته، جستجوی ترتیبی، کد خالص (محض)، آرایه، بردار (عملیات برداری)، اما تکنیک‌های برنامه‌نویسی و ساختارهای داده‌ای که اصل مراجعات محلی را رعایت نمی‌کنند، برای صفحه بندی بر پایه تقاضا مناسب نیستند. در واقع این ساختارها به محل‌های مختلف پرش می‌کنند.

مانند جدول سمبل hashed، جستجوی دودویی و آدرس‌دهی غیرمستقیم (کارکرد غیرمستقیم)

در حافظه مجازی با صفحه‌بندی درخواستی (بر پایه تقاضا)، صفحات فقط زمانی بارگذاری می‌شوند که در حین اجرای برنامه درخواست شوند و به جای اینکه کل فرآیند به حافظه آورده شود، فقط صفحات مورد نیاز وارد حافظه می‌شوند. بر این اساس تکنیک‌های برنامه‌نویسی و ساختارهایی برای این محیط مناسب هستند که اصل مراجعات محلی در آن‌ها رعایت شود.

انتخاب دقیق تکنیک‌های برنامه‌نویسی و ساختمان داده‌ها می‌تواند، اصل محلی بودن مراجعات را افزایش و نرخ خطای صفحه و تعداد صفحات موجود در مجموعه کاری را کاهش دهد. از این رو پشته ویژگی مراجعات محلی را به خوبی دارد، زیرا دستیابی از بالای آن صورت می‌گیرد و بعدی و بعدی به هم نزدیک هستند. همچنین در جستجوی ترتیبی، کد خالص و عملیات برداری اصلی مراجعات محلی رعایت می‌شود.

اما جدول درهم‌سازی (hashed) از جمله ساختارهایی است که مراجعات را متفرق می‌کند و به محلی بودن آسیب می‌رساند. همچنین جستجوی دودویی و کارکرد غیرمستقیم باعث تضعیف محلی بودن می‌شوند. محلی بودن را مانند شعاع یک دایره در نظر بگیرید، هر چه قدر این شعاع کوچکتر شود محلی بودن بیشتر و بیشتر می‌شود و هر چه قدر این شعاع بزرگتر شود محلی بودن کمتر و کمتر می‌شود. شما با هم محله‌های خود محلی‌تر هستید، اما هر چه قدر از محله خود دور می‌شوید، اهالی محله‌های دیگر با شما کمتر محلی هستند.

۹- گزینه (۱) صحیح است.

در تکنیک صفحه‌بندی، برای هر فرآیند یک جدول صفحه معمولی تشکیل می‌شود که در آن به ازای همه صفحه‌های یک فرآیند، درایه وجود دارد و هر درایه مشخص می‌کند که کدام قاب فیزیکی به این صفحه اختصاص یافته است. در این روش وقتی فرآیندها بزرگ باشند، هزینه نگهداری جداول صفحه بسیار زیاد می‌شود، در ضمن به ازای هر فرآیند نیز باید یک جدول صفحه داشته باشیم. به عبارتی وقتی تعداد و اندازه فرآیندها بزرگ شود، این روش برای ترجمه آدرس مجازی به فیزیکی مقرون به صرفه نیست.

برای حل این مشکل **جدول صفحه چندسطحی** و **جدول صفحه معکوس** ابداع شده است. جدول صفحه معکوس از نظر سربار حافظه بهتر از روش جدول صفحه چندسطحی است و حالت حداقلی است. اغلب سیستم‌های کامپیوتری، فضای آدرس منطقی(مجازی) بزرگی را پشتیبانی می‌کنند، مانند کامپیوترهایی که آدرس‌های 32 یا 64 بیتی را پشتیبانی می‌کنند، در چنین محیط‌هایی جدول صفحه معمولی بسیار بزرگ خواهد شد. برای مثال یک سیستم را با 32 بیت فضای آدرس منطقی(مجازی) در نظر بگیرید. اگر اندازه صفحه در این سیستم برابر با 4KB (2^{12}) باشد، در اینصورت جدول صفحه شامل بیش از یک میلیون درایه خواهد بود. $\left(2^{32} / 2^{12} = 2^{20}\right)$. فرض کنید هر درایه جدول صفحه معمولی، شامل 4 بایت باشد، در اینصورت هر فرآیند به $\left(4 \times 2^{20}\right)$ 4MB فضای آدرس فیزیکی فقط برای نگهداری جدول صفحه معمولی نیاز دارد. همچنین مشخص است که با توجه به محدودیت اندازه قاب (4KB) نمی‌توان این جدول صفحه معمولی را بصورت پیوسته درون یک قاب جا داد. 4MB اندازه جدول صفحه معمولی در فضای 4KB مربوط به یک قاب جا نمی‌شود. یک راه حل تقسیم جدول صفحه معمولی به جداول صفحه جزئی و ایجاد جدول صفحه چندسطحی است، در این حالت جدول صفحه معمولی نیز همانند فرآیند صفحه‌بندی می‌شود. راه حل دیگر استفاده از جدول صفحه معکوس است. در راه حل جدول صفحه معکوس به جای اینکه در جداول صفحه مربوط به هر فرآیند، به ازای هر صفحه مجازی یک درایه داشته باشیم، به ازای هر قاب در حافظه فیزیکی یک درایه در جدول صفحه معکوس نگهداری می‌کنیم. در واقع به ازای هر قاب حافظه اصلی اینکه در حال حاضر کدام صفحه مربوط به کدام فرآیند در این قاب ذخیره شده است نگهداری می‌شود. بنابراین تعداد درایه‌های جدول صفحه معکوس برابر تعداد قاب‌های حافظه فیزیکی(اصلی) است.

مثال: در یک سیستم صفحه‌بندی که دارای 34 بیت آدرس است 23 بیت اول برای شماره صفحه و 11 بیت بعدی برای آدرس‌دهی درون صفحه است. در یک سیستم با صفحه‌بندی معکوس (Inverted Page Table) با 128 مگابایت حافظه، جدول صفحه دارای چند خانه (درایه یا مدخل) است؟

پاسخ: تعداد درایه‌های جدول صفحه معکوس، مطابق رابطه زیر محاسبه می‌گردد:

تعداد قاب‌های حافظه فیزیکی = تعداد درایه‌های جدول صفحه معکوس

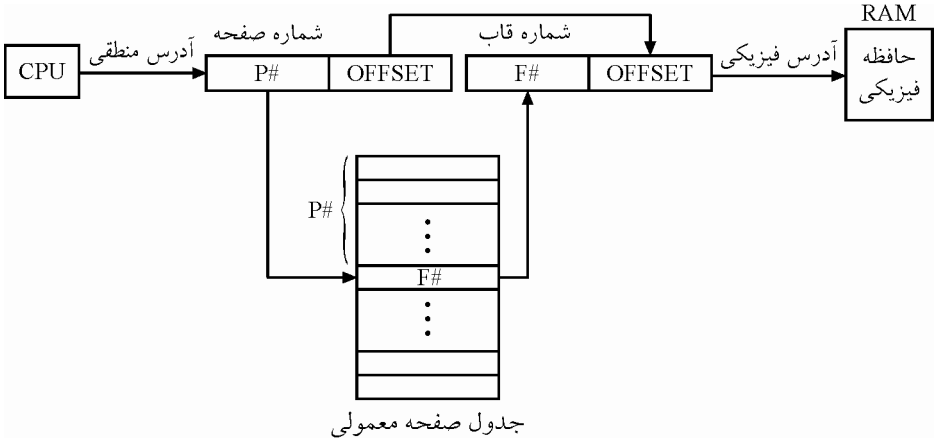
$$\text{تعداد قاب‌های حافظه فیزیکی} = \frac{\text{اندازه حافظه فیزیکی}}{\text{اندازه صفحه یا اندازه قاب}} = \frac{2^7 \times 2^{20}}{2^{11}} = 2^{16}$$

همانطور که گفتیم تعداد درایه‌های جدول صفحه معکوس برابر تعداد قاب‌های حافظه فیزیکی است، بنابراین تعداد درایه‌های جدول صفحه معکوس به صورت زیر خواهد بود:

$$\text{تعداد درایه‌های جدول صفحه معکوس} = \frac{\text{اندازه حافظه فیزیکی}}{\text{اندازه صفحه یا اندازه قاب}} = \frac{2^7 \times 2^{20}}{2^{11}} = 2^{16}$$

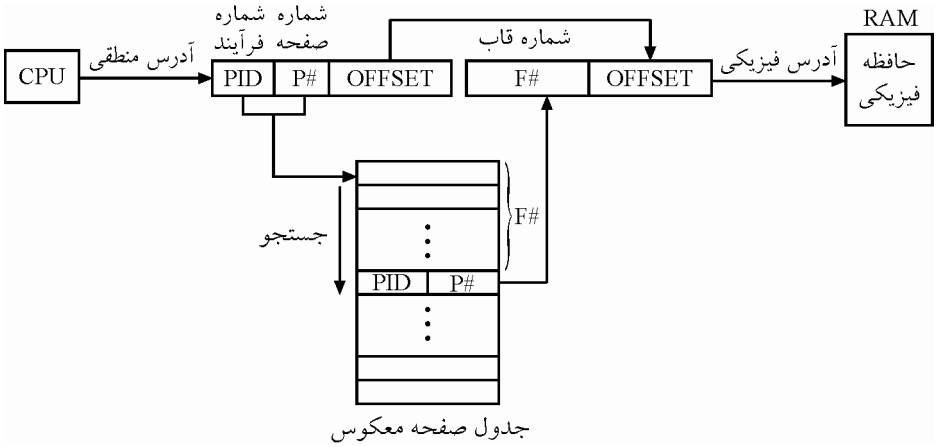
در یک عبارت ساده، هدف از استفاده از جدول صفحه معکوس، **کاهش** میزان حافظه فیزیکی مورد نیاز برای ترجمه آدرس مجازی به فیزیکی است. با استفاده از تکنیک جدول صفحه معکوس، مقدار زیادی در حافظه فیزیکی صرفه‌جویی می‌شود اما تبدیل آدرس مجازی به فیزیکی کندتر و وقت‌گیرتر می‌شود. در واقع **صرفه‌جویی** در مصرف حافظه فیزیکی را بدست می‌آوریم، اما **سرعت** تبدیل آدرس مجازی به فیزیکی را از دست می‌دهیم. زیرا جدول صفحه معکوس بر اساس #F اندیس شده است و نه مانند جدول صفحه معمولی که بر اساس #P اندیس شده است. در واقع وقتی یک فرآیند با PID مختص به خود به صفحه مجازی #P مراجعه می‌کند، سخت‌افزار دیگر نمی‌تواند از #P به عنوان اندیس جدول صفحه استفاده کند و صفحه فیزیکی را بیابد و از این جهت باید سرتاسر جدول صفحه معکوس (وارونه) را برای یافتن درایه (PID,P#) از آدرس مجازی (PID,P#,OFFSET) جستجو کند. اگر یک تطبیق پیدا شود، در اینصورت آدرس فیزیکی (F#,OFFSET) تولید خواهد شد و اگر هیچ تطبیقی یافته نشود، در اینصورت یک دسترسی آدرس غیر مجاز می‌باشد. ضمن اینکه این جستجو باید به ازای هربار دسترسی به حافظه انجام شود. همانطور که گفتیم در جدول صفحه معکوس **صرفه‌جویی** در مصرف حافظه فیزیکی را بدست می‌آوریم، اما **سرعت** تبدیل آدرس مجازی به فیزیکی را از دست می‌دهیم، یعنی سرعت نگاشت آدرس منطقی به فیزیکی **کاهش** می‌یابد. درواقع جدول صفحه معکوس **زمان** نگاشت آدرس منطقی به آدرس فیزیکی را **افزایش** می‌دهد. هرچند موجب صرفه‌جویی در مصرف حافظه فیزیکی می‌شود. **بنابراین گزینه اول نادرست و گزینه دوم درست است.**

نحوه ترجمه آدرس منطقی به فیزیکی، در سیستم جدول **صفحه معمولی** به صورت زیر است:



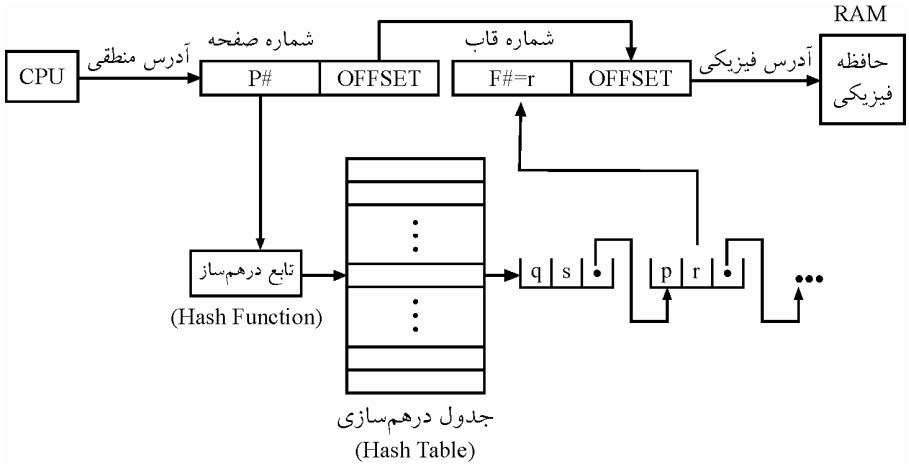
جدول صفحه معمولی

نحوه ترجمه آدرس منطقی به فیزیکی، در سیستم جدول **صفحه معکوس** به صورت زیر است:



جدول صفحه درهم‌سازی شده

یک راه حل عمومی برای نگاشت فضای آدرس بزرگ به کوچک، مثلا نگاشت فضای آدرس 64 بیتی به 32 بیتی، در سیستمی که فضای آدرس 64 بیتی را پشتیبانی نمی‌کند و فضای 32 بیتی را پشتیبانی می‌کند، استفاده از جدول صفحه درهم‌سازی شده است. تابع درهم‌ساز حجم زیادی از داده را به یک عدد طبیعی تبدیل می‌کند. عدد طبیعی حاصل از تابع درهم‌سازی معمولاً به عنوان اندیس یک آرایه مورد استفاده قرار می‌گیرد. به مقادیر حاصل از تابع درهم‌ساز مقدار درهم(Hashed Value) گفته می‌شود. به عبارت دیگر در اینجا تابع درهم‌ساز مقدار #P از یک آدرس مجازی بزرگ مربوط به یک فرآیند را می‌گیرد و پس از تقسیم بر عدد تابع درهم‌ساز، باقی‌مانده حاصل درایه مشخصی را در جدول درهم‌سازی نشان می‌دهد. هر درایه در جدول درهم‌سازی شامل یک لیست پیوندی از باقی‌مانده‌های یکسان حاصل از تابع درهم‌ساز است. به عبارت دیگر همه صفحات مجازی یک فرآیند که مقدار درهم یکسانی دارند تشکیل یک لیست پیوندی می‌دهند در جلوی یک درایه مشخص از جدول درهم‌سازی. هر گره لیست پیوندی شامل سه فیلد (۱) شماره صفحه مجازی، (۲) شماره قاب و (۳) یک اشاره‌گر به گره بعدی لیست پیوندی است. نحوه کار بدین صورت است که پس از آنکه درایه مرتبط با #P مورد نظر توسط تابع درهم‌ساز، در جدول درهم‌سازی مشخص شد، جستجو در جهت کشف شماره قاب مورد انتظار آغاز می‌شود، بدین صورت که #P مورد نظر با فید اول در گره اول لیست پیوندی مقایسه می‌شود، اگر مطابقت داشت، شماره قاب مورد نظر از فیلد دوم در گره اول جهت ساخت آدرس فیزیکی استخراج می‌شود، در غیر اینصورت گره‌های بعدی جهت کشف شماره قاب مورد انتظار در لیست پیوندی مورد جستجو قرار می‌گیرد. شکل زیر گویای مطلب است:



توجه: برای افزایش سرعت تبدیل آدرس مجازی به فیزیکی در جدول صفحه معکوس می‌توان از تکنیک TLB و یا تفکر جدول درهم‌سازی(Hashe Table) با اندکی تغییر استفاده نمود. بنابراین هر ترجمه آدرس مجازی به فیزیکی نیازمند دوبار مراجعه به حافظه فیزیکی می‌باشد، یکبار برای دسترسی به جدول درهم‌سازی و یکبار دیگر برای دسترسی به جدول صفحه معکوس.

توجه: جدول صفحه معکوس، اطلاعات کاملی در مورد فضای آدرس منطقی یک فرآیند را ندارد. در حالی که سیستم صفحه‌بندی مبتنی بر تقاضا به این اطلاعات کامل جهت پردازش نقص‌های صفحه نیازمند است. برای دسترسی به این اطلاعات کامل، یک **جدول صفحه خارجی**، یکی به ازای هر فرآیند، باید نگهداری شود. جدول صفحه خارجی شامل فیلدهای Valid، Present و آدرس صفحه مجازی دچار نقص صفحه شده بر روی رسانه ذخیره‌سازی است. **بنابراین گزینه چهارم درست است.**

توجه: هنگامی که یک نقص صفحه رخ می‌دهد، ممکن است یک **نقص صفحه دیگر** برای بارگذاری **جدول صفحه خارجی** از رسانه ذخیره‌سازی به حافظه فیزیکی رخ دهد. بنابراین در جدول صفحه معکوس، زمان سرویس نقص صفحه(page fault) به دلیل ایجاد یک نقص صفحه دیگر، **افزایش** می‌یابد. **بنابراین گزینه سوم درست است.**

توجه: در سیستم‌هایی که از جدول صفحه معکوس‌شده استفاده می‌کنند، استفاده اشتراکی از صفحات میان فرآیندها در حالت عادی امکان‌پذیر نیست، در حالت کلی اشتراک به شکل چند به یک است، یعنی صفحه مشترک میان فرآیندها در یک قاب مشخص قرار می‌گیرد. که پیاده‌سازی حالت چند به یک در جدول صفحه معکوس درحالت ابتدایی امکان‌پذیر نیست. زیرا در جدول صفحه معکوس برای هر درایه آن فقط می‌توان شماره یک صفحه از یک فرآیند را قرار داد. البته توسط مکانیزم‌هایی می‌توان اشتراک‌گذاری صفحات میان فرآیندها را در جدول صفحه معکوس نیز پیاده‌سازی نمود. برای مثال به جدول صفحه معکوس اجازه دهیم برای هر درایه، بیش از یک شماره صفحه را آدرس‌دهی کند.

۱۰- گزینه (۳) صحیح است.

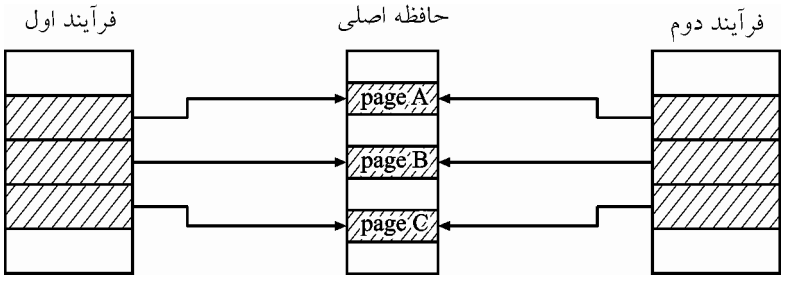
تکنیک صفحه‌بندی اصلاً تکه‌تکه شدن خارجی ندارد. بنابراین گزینه سوم نادرست است. در این تکنیک کاهش اندازه‌ی صفحه، سبب کاهش تکه‌تکه شدن داخلی (در صفحه آخر هر فرآیند) می‌شود، بنابراین گزینه چهارم درست است. کاهش اندازه صفحه سبب کاهش زمان سرویس نقص صفحه می‌شود، چون کاهش اندازه صفحه منجر به کاهش عملیات مبادله صفحه موردنظر مابین حافظه اصلی و هارد دیسک می‌شود. بنابراین گزینه اول درست است. کاهش اندازه صفحه سبب افزایش بهره‌وری حافظه می‌گردد، چون کاهش اندازه صفحه سبب کاهش تکه تکه شدن داخلی می‌شود که این امر منجر به افزایش بهره‌وری حافظه می‌گردد. همچنین کاهش اندازه صفحه سبب افزایش زمان I/O می‌شود چون هر چند کاهش اندازه صفحه منجر به کاهش عملیات مبادله صفحه موردنظر مابین حافظه اصلی و هارد دیسک می‌شود، اما در مجموع تعداد و تعدد این مبادله‌ها مابین حافظه اصلی و هارد دیسک زیاد می‌شود به دلیل کوچک بودن اندازه صفحات و به تبع ازدیاد تعداد صفحات و این یعنی افزایش زمان I/O برای کل صفحات مبادله شده. بنابراین گزینه دوم درست است.

۱۱- گزینه (۲) صحیح است.

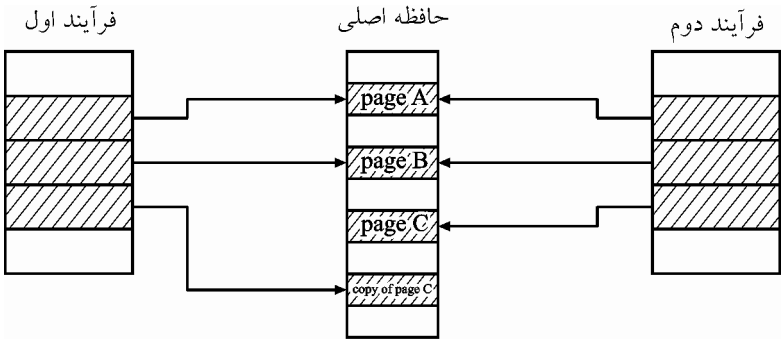
در تکنیک copy-on-write با ایجاد فرآیند جدید توسط دستور fork، صفحات فرآیند پدر برای فرزند کپی نمی‌شوند، بلکه این صفحات بین فرآیند پدر و فرزند به اشتراک گذاشته می‌شوند. هر زمان که یکی از دو فرآیند پدر یا فرزند بخواهد صفحه‌ای را تغییر دهد، یک کپی جداگانه از آن صفحه برای آن فرآیند ساخته می‌شود و فرآیند از آن به بعد از آن صفحه استفاده می‌کند (به جای صفحه مشترک) و فرآیندهای دیگر از صفحه اصلی (مشترک) استفاده می‌کنند.

در این تکنیک زمانی فرآیند پدر می‌تواند خاتمه یابد که یک کپی از صفحات آن برای هر یک از فرزندانش ایجاد شده باشد (یعنی صفحات تمامی فرآیندهای فرزند تغییر کرده باشند، به عبارت دیگر همه فرزندان همه صفحه‌های مربوط به خود را تغییر داده باشند.) و دیگر نیازی به صفحات فرآیند پدر نباشد.

شکل زیر گویای مطالب می‌باشد:



قبل از اینکه فرآیند اول صفحه C را تغییر دهد.



بعد از اینکه فرآیند اول صفحه C را تغییر دهد.

۱۲- گزینه (۳) صحیح است.

به طور کلی میانگین زمان دسترسی سیستم به مقصد نهایی برای هر آدرس سیستم از رابطه زیر محاسبه می‌گردد:

$$T_{Access} = T_{Translation} + T_{Destination}$$

که پارامترهای آن شامل $T_{Translation}$ یعنی **بخش ترجمه** و $T_{Destination}$ یعنی **بخش مقصد** می‌باشد.

$T_{Translation}$: زمان ترجمه

توجه: مطابق فرض سوال در **بخش ترجمه** یعنی $T_{Translation}$، یک عنصر T_{Mem} برای ترجمه وجود دارد که مقدار آن بیان نشده است، بنابراین مقدار آنرا صفر در نظر می‌گیریم. به صورت زیر:

$$T_{Mem} = 0$$

بنابراین زمان **ترجمه** به صورت زیر است:

$$T_{Translation} = 0$$

$T_{Destination}$: میانگین زمان دسترسی به مقصد موردنظر

توجه: مطابق فرض سوال در **بخش مقصد** یعنی $T_{Destination}$، دو عنصر T_{Mem} و T_{Disk} برای دسترسی به مقصد مورد نظر وجود دارد، بنابراین باید میانگین آنها محاسبه گردد.

برای محاسبه $T_{Destination}$ میانگین T_{Mem} و T_{Disk} را محاسبه می‌کنیم که همان $T_{Destination}$ است.

مطابق مفروضات مطرح شده در صورت سوال سه وضعیت زیر برقرار است:

الف) صفحه مورد درخواست در دیسک قرار دارد که باید به حافظه منتقل شود، اما قاب خالی در حافظه موجود نیست، بنابراین یک صفحه باید از حافظه خارج گردد، تا فضای کافی برای ورود صفحه مورد درخواست فراهم گردد. ولی صفحه‌ای که باید از حافظه خارج گردد، **تمیز** است، یعنی کپی صفحه موجود در حافظه با کپی موجود در دیسک یکی است، یعنی صفحه تغییر نکرده

است، بنابراین نیاز به دوباره نویسی ندارد. یعنی صفحه جدید بر روی صفحه کاندید برای خروج، جایگزین می‌گردد، بدون آنکه صفحه کاندید برای خروج، اطلاعتش بر روی دیسک ذخیره گردد.
مطابق فرض مسأله $\frac{1}{2}$ صفحات **تمیز** یعنی تغییر نیافته‌اند به شکلی که قاب خالی هم در حافظه موجود نیست یا مستقل از **تمیز** و **کثیف** بودن صفحات، قاب خالی در حافظه موجود است که مطابق آنچه گفتیم رابطه زیر برقرار است:

$$\frac{1}{2}\times T_{Disk}$$

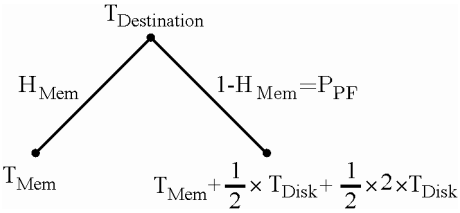
ب) صفحه مورد درخواست در دیسک قرار دارد که باید به حافظه منتقل شود، اما قاب خالی در حافظه موجود است. بنابراین صفحه مورد درخواست در قاب خالی قرار می‌گیرد.
مطابق فرض مسأله $\frac{1}{2}$ صفحات **تمیز** یعنی تغییر نیافته‌اند به شکلی که قاب خالی هم در حافظه موجود نیست یا مستقل از **تمیز** و **کثیف** بودن صفحات، قاب خالی در حافظه موجود است که مطابق آنچه گفتیم رابطه زیر برقرار است:

$$\frac{1}{2}\times T_{Disk}$$

ج) صفحه مورد درخواست در دیسک قرار دارد که باید به حافظه منتقل شود، اما قاب خالی در حافظه موجود نیست، بنابراین یک صفحه باید از حافظه خارج گردد تا فضای کافی برای ورود صفحه مورد درخواست فراهم گردد. ولی صفحه‌ای که باید از حافظه خارج گردد، **کثیف** است، یعنی کپی صفحه‌ی موجود در حافظه با کپی صفحه موجود در دیسک یکی نیست، یعنی صفحه تغییر کرده‌است، بنابراین نیاز به دوباره‌نویسی دارد. یعنی قبل از آنکه صفحه جدید بر روی صفحه-ی کاندید برای خروج جایگزین گردد، ابتدا اطلاعات صفحه کاندید برای خروج بر روی دیسک ذخیره می‌گردد که یک $T_{RW}=T_{Disk}$ زمان نیاز دارد، سپس صفحه جدید بر روی صفحه کاندید برای خروج، جایگزین می‌گردد، که مجددا یک T_{Disk} زمان نیاز دارد.
مطابق فرض مسأله $\frac{1}{2}$ صفحات **کثیف** یعنی تغییر یافته‌اند به شکلی که قاب خالی هم در حافظه موجود نیست که مطابق آنچه گفتیم رابطه زیر برقرار است:

$$\frac{1}{2}\times\left[T_{RW}+T_{Disk}\right]=\frac{1}{2}\times 2\times T_{Disk}$$

بنابراین صفحات تغییریافته یا **کثیف** نیاز به دوبار جابه‌جایی بین حافظه و دیسک دارند، یک بار برای قرار دادن صفحه کثیف از حافظه به دیسک و یک بار دیگر برای قرار دادن صفحه مورد درخواست از دیسک به حافظه. البته اگر قاب خالی در حافظه نباشد.
برای بدست آوردن $T_{Destination}$ درخت زیر را در نظر بگیرید:



پس از ساده‌سازی درخت فوق رابطه زیر را برای محاسبه $T_{Destination}$ خواهیم داشت:

$$T_{Destination} = T_{Mem} + \left(1-H_{Mem}\right) \times\left[\frac{1}{2}\times T_{Disk} + \frac{1}{2}\times 2\times T_{Disk}\right]$$

$$T_{Destination} = T_{Mem} + \left(P_{PF}\right) \times\left[\frac{1}{2}\times T_{Disk} + \frac{1}{2}\times 2\times T_{Disk}\right]$$

$$T_{Destination} = 0 + \left(P\right) \times\left[\frac{1}{2}\times d + \frac{1}{2}\times 2\times d\right]$$

$$T_{Destination} = \left(P\right) \times\left[\frac{1}{2}\times d + \frac{1}{2}\times 2\times d\right] = \frac{1}{2}\times p\times d + \frac{1}{2}\times p\times 2d = \frac{p\times d}{2} + \frac{2\times p\times d}{2} = \frac{3pd}{2}$$

و در نهایت براساس رابطه کل زمان دسترسی سیستم خواهیم داشت:

$$T_{Access(old)} = T_{Translation} + T_{Destination}$$

$$T_{Access(old)} = 0 + \frac{3pd}{2} = \frac{3pd}{2}$$

حال اگر سرعت حافظه جانبی ۲ برابر شود، یعنی زمان d به $\frac{d}{2}$ کاهش یابد، آنگاه رابطه زیر برقرار است:

$$T_{Access(new)} = T_{Translation} + T_{Destination}$$

$$T_{Access(new)} = 0 + \frac{3p\frac{d}{2}}{2} = \frac{3pd}{4}$$

میزان کاهش برابر تفاضل دو حالت **قدیم** و **جدید** می‌باشد، به صورت زیر:

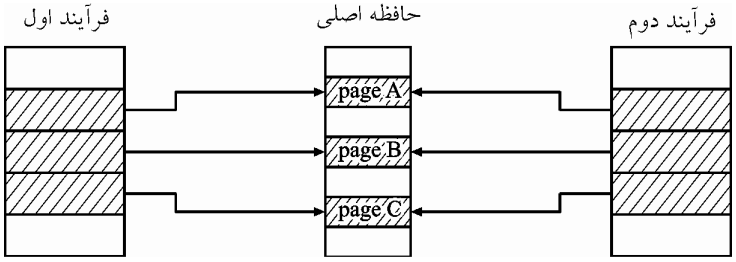
$$T_{Access(old)} - T_{Access(new)} = \frac{3pd}{2} - \frac{3pd}{4} = \frac{6pd}{4} - \frac{3pd}{4} = \frac{3pd}{4} = \frac{3}{4}pd$$

بنابراین گزینه سوم پاسخ سؤال خواهد بود.

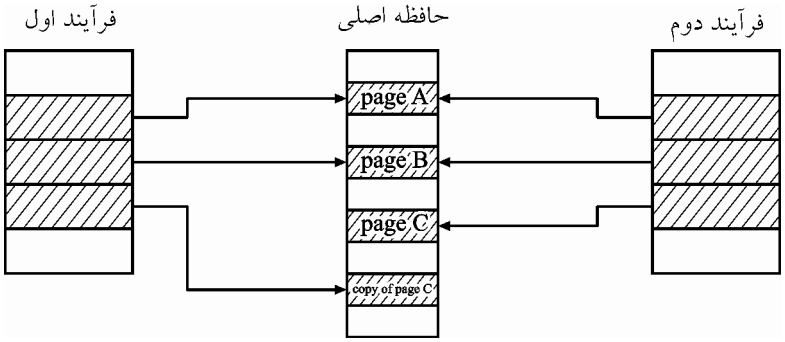
۱۳- گزینه (۱ و ۳) صحیح است.

در unix برای ایجاد یک فرآیند جدید از دستور ( )fork استفاده می‌شود و این کار را با ایجاد یک فرآیند تکراری و دقیقاً یکسان با داده‌های مشترک با فرآیند پدر انجام می‌دهد. در واقع در ابتدای کار داده‌های Heap و Code بین فرآیند پدر و فرزند به اشتراک گذاشته می‌شود. اما در ادامه اگر

پدر و فرزند قصد اعمال تغییراتی را در داده‌های مشترک خود داشته باشند، در اینصورت هـر یـک حافظه‌های مختص به خود را در اختیار می‌گیرند.



قبل از اینکه فرآیند اول صفحه C را تغییر دهد.



بعد از اینکه فرآیند اول صفحه C را تغییر دهد.

فراخوانی fork یک مقدار برمی‌گرداند که برای فرآیند فرزند برابر صفر و برای فرآیند پدر برابـر بـا شناسه فرآیند (PID) فرزند خواهد بود. بنابراین با استفاده از عدد PID حاصل از بازگشت، می‌توان متوجه شد که بین دو فرآیند، کدامیک پدر و کدامیک فرزند است.

توجه: هر فرآیند Process id، Stack و رجیسترهای مختص به خود را دارد.

توجه: سازمان سنجش آموزش کشور، ابتدا در کلید اولیه خود گزینه اول را به عنوان پاسـخ اعـلام نمود، اما در کلید نهایی گزینه‌های اول و سوم را به عنوان پاسخ نهایی اعلام نمود، که کـار درسـتی بوده است.

۱۴- گزینه (۴) صحیح است.

یک راه مقابله با Trashing کنترل تعداد وقفه‌های نقص صـفحه بـا اسـتفاده از الگـوریتم فرکـانس نقص صفحه یا PFF (Page Fault Frequency) است. این الگوریتم زمان کاهش یا افـزایش تعـداد قاب صفحه تخصیص یافته به یک فرآیند را بیان می‌کند، اما کنترلی در مورد اینکه کدام صفحه باید

در هنگام نقص صفحه جایگزین شود، انجام نمی‌دهد. این الگوریتم فقط اندازه مجموعه تخصیص را کنترل می‌کند. که بهتر است، این مجموعه تخصیص برابر مجموعه کاری (working set) باشد.
روال کار بدین صورت است که اگر تعداد نقص‌های صفحه برای یک فرآیند افزایش یافت، باید تعدادی قاب حافظه به آن اختصاص یابد و اگر تعداد نقص‌های صفحه برای یک فرآیند از یک حد پایین کمتر شد، باید تعدادی قاب از آن فرآیند پس گرفته شود. نکته مهم اینجاست که اگر تعداد نقص‌های صفحه یک فرآیند بالا رفت، ولی قاب آزاد در حافظه وجود نداشت، باید درجه چند برنامگی سیستم را کاهش داد و یک یا چند فرآیند را به حالت معلق در آورد تا قاب‌هایی که در اختیار دارد، آزاد شود.

۱۵- گزینه (۴) صحیح است.

یک راه حل مقابله با Trashing کنترل تعداد وقفه‌های نقص صفحه با استفاده از الگوریتم فرکانس نقص صفحه یا PFF (Page Fault Frequency) است. این الگوریتم زمان کاهش یا افزایش تعداد قاب صفحه تخصیص یافته به یک فرآیند را بیان می‌کند، اما کنترلی در مورد این که کدام صفحه باید در هنگام نقص صفحه جایگزین شود، انجام نمی‌دهد. این الگوریتم فقط اندازه مجموعه تخصیص را کنترل می‌کند. که بهتر است، این مجموعه تخصیص برابر مجموعه کاری (working set) باشد. روال کار بدین صورت است که اگر تعداد نقص‌های صفحه برای یک فرآیند افزایش یافت، باید تعدادی قاب حافظه به آن اختصاص یابد و اگر تعداد نقص‌های صفحه برای یک فرآیند از یک حد پایین کم‌تر شد، باید تعدادی قاب از آن فرآیند پس گرفته شود. نکته مهم اینجاست که اگر تعداد نقص‌های صفحه یک فرآیند بالا رفت، ولی قاب آزاد حافظه وجود نداشت، باید درجه چند برنامگی سیستم را کاهش داد و یک یا چند فرآیند را به حالت معلق در آورد تا قاب‌هایی که در اختیار دارد، آزاد شود.

۱۶- گزینه (۳) صحیح است.

الگوریتم LRU (Least Recently Used)

ایده اصلی این الگوریتم این است که اگر صفحه‌ای در چند دستور اخیر مراجعات زیادی داشته است، به احتمال قوی در دستورات بعدی هم ارجاعات زیادی خواهد داشت، همچنین اگر یک صفحه، اخیرا هیچ مراجعه‌ای نداشته، احتمالاً در آینده نزدیک هم ارجاعی نخواهد داشت. در واقع این الگوریتم بیان می‌کند هنگام وقوع خطای نقص صفحه، صفحه‌ای را حذف کنید که طولانی‌ترین زمان عدم استفاده را دارد.
می‌توان گفت LRU تقریبی از الگوریتم بهینه می‌باشد که در آن به جای توجه به آینده، به گذشته توجه می‌شود.

توجه: الگوریتم LRU ناهنجاری بی‌لیدی ندارد، بنابراین با افزایش تعداد قاب، همواره، نقص صفحه کاهش می‌یابد.

توجه: بهتر بود طراح محترم، تعداد قاب‌های صفحه موجود در RAM را بیان می‌نمود، اما به واسطه آزمون و خطا می‌توان به پاسخ احتمالی طراح رسید.

اگر 1 قاب در نظر گرفته شود:

واضح است که در این حالت، به ازای تمام در خواست‌ها، نقص صفحه رخ خواهد داد، که این مقدار برابر 20 نقض صفحه می‌باشد.

اگر 2 قاب در نظر گرفته شود:

نگاه به گذشته ←

رشته مراجعات	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
قاب 1	7	7	1	1	0	0	0	0	2	2	0	0	2	2	2	2	1	1	0	0
قاب 2		0	0	2	2	3	3	4	4	3	3	3	3	1	1	0	0	7	7	1
نقص صفحه	*	*	*	*	*	*		*	*	*	*		*	*		*	*	*	*	*

در این حالت 17 نقص صفحه به وقوع پیوست.

اگر 3 قاب در نظر گرفته شود:

نگاه به گذشته ←

رشته مراجعات	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
قاب 1	7	7	7	2	2	2	2	4	4	4	0	0	0	1	1	1	1	1	1	1
قاب 2		0	0	0	0	0	0	0	0	3	3	3	3	3	3	0	0	0	0	0
قاب 3			1	1	1	3	3	3	2	2	2	2	2	2	2	2	2	7	7	7
نقص صفحه	*	*	*	*		*		*	*	*	*			*		*		*		

در این حالت 12 نقص صفحه به وقوع پیوست، که برابر گزینه سوم است.

اگر 4 قاب در نظر گرفته شود:

نگاه به گذشته ←

رشته مراجعات	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
قاب 1	7	7	7	7	7	3	3	3	3	3	3	3	3	3	3	3	3	7	7	7
قاب 2		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
قاب 3			1	1	1	1	1	4	4	4	4	4	4	1	1	1	1	1	1	1
قاب 4				2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
نقص صفحه	*	*	*	*		*		*						*				*		

در این حالت 8 نقص صفحه به وقوع پیوست.

توجه: اگر این روال را ادامه دهیم، یعنی تعداد قاب‌ها را افزایش دهیم، مطابق خاصیت الگوریتم LRU، واضح است که نقص صفحه کاهش می‌یابد. بنابراین واضح است که گزینه سوم درست است.

توجه: سازمان سنجش آموزش کشور، در کلید اولیه خود، گزینه سوم را به عنوان پاسخ اعلام کرده بود. اما در کلید نهایی این سوال حذف گردید، که کار درستی بوده است.

سخنی با طراح: طراح محترم، سوال استاندارد باید مستقل از گزینه‌ها قابل حل باشد و وابسته به گزینه‌ها نباشد. این را دیگه همه می‌دانند، همه ...

۱۷- گزینه (۲) صحیح است.

به طور کلی میانگین زمان دسترسی سیستم به مقصد نهایی برای هر آدرس سیستم از رابطه زیر محاسبه می‌گردد:

$$T_{Access} = T_{Translation} + T_{Destination}$$

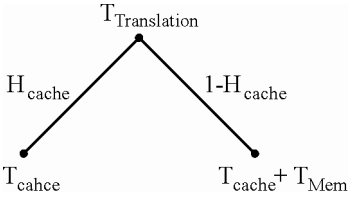
که پارامترهای آن شامل $T_{Translation}$ یعنی **بخش ترجمه** و $T_{Destination}$ یعنی **بخش مقصد** می‌باشد.

$\mathbf{T_{Translation}}$: میانگین زمان ترجمه

توجه: مطابق فرض سوال در **بخش ترجمه** یعنی $T_{Translation}$، دو عنصر T_{Cache} و T_{Mem} برای ترجمه وجود دارد، بنابراین باید میانگین آنها محاسبه گردد.

برای محاسبه $T_{Translation}$ میانگین T_{Cache} و T_{Mem} را محاسبه می‌کنیم که همان $T_{Translation}$ است. به صورت زیر:

برای بدست آوردن $T_{Translation}$ درخت زیر را در نظر بگیرید:

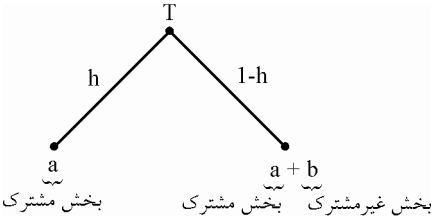


پس از ساده‌سازی درخت فوق رابطه زیر را برای محاسبه $T_{Translation}$ خواهیم داشت:

$$T_{Translation} = T_{Cache} + (1-H_{Cache}) \times T_{Mem}$$

$$T_{Translation} = 10 + (1-H_{Cache}) \times 200$$

توجه: جهت ساده‌سازی رابطه درخت فوق، همواره می‌توان از **اتحاد درخت احتمال** به صورت زیر، استفاده نمود:



$T =$ بخش مشترک $+ (1-h) \times$ بخش غیرمشترک

$$T = a + (1-h) \times b$$

پس از ساده‌سازی درخت مطرح شده براساس **اتحاد درخت احتمال**، رابطه زیر را برای محاسبه $T_{Translation}$ خواهیم داشت:

$$T_{Translation} = T_{Cache} + (1\text{-}H_{Cache}) \times T_{Mem}$$

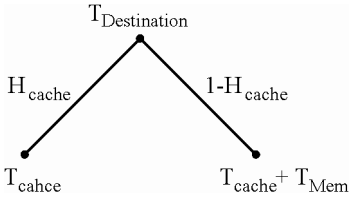
$$T_{Translation} = 10 + (1\text{-}H_{Cache}) \times 200$$

$T_{Destination}$: میانگین زمان دسترسی به مقصد موردنظر

توجه: مطابق فرض سوال در **بخش مقصد** یعنی $T_{Destination}$، دو عنصر T_{Cache} و T_{Mem} برای دسترسی به مقصد مورد نظر وجود دارد، بنابراین باید میانگین آنها محاسبه گردد.

برای محاسبه $T_{Destination}$ میانگین T_{Cache} و T_{Mem} را محاسبه می‌کنیم که همان $T_{Destination}$ است. به صورت زیر:

برای بدست آوردن $T_{Destination}$ درخت زیر را در نظر بگیرید:



پس از ساده‌سازی درخت فوق رابطه زیر را برای محاسبه $T_{Destination}$ خواهیم داشت:

$$T_{Destination} = T_{Cache} + (1\text{-}H_{Cache}) \times T_{Mem}$$

$$T_{Destination} = 10 + (1\text{-}H_{Cache}) \times 200$$

و در نهایت براساس رابطه کل زمان دسترسی سیستم خواهیم داشت:

$$T_{Access} = T_{Translation} + T_{Destination}$$

$$T_{Access} = \left[10 + (1-H_{Cache}) \times 200\right] + \left[10 + (1-H_{Cache}) \times 200\right]$$

مطابق فرض سوال، زمان دسترسی موثر (T_{Access}) در این سیستم، 10 درصد بیشتر از زمان دسترسی به کش (T_{cache}) است، بنابراین داریم:

$$T_{Access} = T_{cache} + \frac{1}{10} T_{cache} = 10 + \frac{1}{10} \times 10 = 11 \text{ ns}$$

همچنین براساس رابطه کل زمان دسترسی سیستم خواهیم داشت:

$$T_{Access} = T_{Translation} + T_{Destination}$$

$$T_{Access} = \left[T_{cache} + (1 - H_{cache}) T_{Mem}\right] + \left[T_{cache} + (1 - H_{cache}) T_{Mem}\right]$$

$$T_{Access} = \left[10 + \left(1 - H_{Cache}\right) \times 200\right] + \left[10 + \left(1 - H_{Cache}\right) \times 200\right]$$

$$11 = \left[10 + \left(1 - H_{Cache}\right) \times 200\right] + \left[10 + \left(1 - H_{Cache}\right) \times 200\right]$$

$$11 = 20 + \left(1 - H_{Cache}\right) \times 400$$

رابطه فوق نشان می‌دهد که حاصل $\left(1 - H_{Cache}\right) \times 400$ باید برابر مقدار 9- باشد که معنا ندارد.
نتیجه اینکه منظور طراح محترم از زمان دسترسی موثر سیستم همان فقط زمان ترجمه سیستم است، و نه کل زمان دسترسی موثر سیستم که البته فرض نادرستی هم هست، در ادامه بر اساس فرض طراح، مساله را حل می‌کنیم.
همانطور که پیشتر گفتیم رابطه زیر برای محاسبه $T_{Translation}$ برقرار است:

$$T_{Translation} = T_{Cache} + (1\text{-}H_{Cache}) \times T_{Mem}$$

$$T_{Translation} = 10 + (1\text{-}H_{Cache}) \times 200$$

مطابق فرض سوال، زمان دسترسی موثر در این سیستم، 10 درصد بیشتر از زمان دسترسی به کش (T_{cache}) است، بنابراین داریم:

$$T_{Translation} = T_{cache} + \frac{1}{10} T_{cache} = 10 + \frac{1}{10} \times 10 = 11 \text{ ns}$$

همچنین براساس رابطه زمان ترجمه سیستم داریم:

$$T_{Translation} = T_{cache} + (1 - H_{cache}) \times T_{Mem}$$

$$11 = 10 + (1 - H_{cache}) \times 200$$

$$1 = (1 - H_{cache}) \times 200$$

$$1 - H_{cache} = \frac{1}{200}$$

$$H_{cache} = 1 - \frac{1}{200} = \frac{199}{200}$$

بنابراین گزینه دوم پاسخ مد نظر طراح سوال بوده است. سازمان سنجش آموزش کشور نیز گزینه دوم را به عنوان پاسخ نهایی سوال اعلام کرده بود.