موسسه بابان

انتشارات بابان و انتشارات راهیان ارشد

درس و کنکور ارشد

سیستم عامل

(مدیریت حافظه اصلی)

ویژه‌ی داوطلبان کنکور کارشناسی ارشد مهندسی کامپیوتر و IT

براساس کتب مرجع

آبراهام سیلبرشاتز، ویلیام استالینگز و اندرو اس تنن‌بام

ارسطو خلیلی‌فر

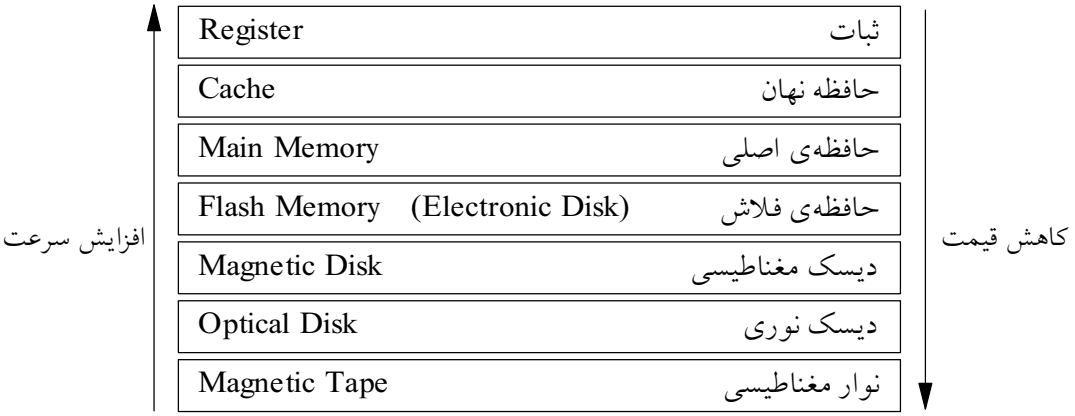
۴ مدیریت حافظه (قطعه‌بندی و صفحه‌بندی)

مقدمه

برنامه‌نویسان همگی تمایل دارند یک حافظه با وسعت بی‌نهایت، بسیار سریع و غیرفرار (پایدار) در اختیار داشته باشند. اما در عمل چنین چیزی ممکن و در دسترس نیست، در عوض اکثر سیستم‌ها از یک حافظه سلسله مراتبی استفاده می‌کنند.

سلسله مراتب حافظه

در سیستم‌های کامپیوتری از حافظه‌های مختلفی استفاده می‌شود. حافظه‌ها را می‌توان براساس قیمت و سرعت به صورت زیر طبقه‌بندی کرد:



سلسله مراتب حافظه

نکته: سه رده‌ی اول (یعنی ثبات، حافظه‌های نهان و اصلی) ناپایدار هستند. به این معنا که با قطع برق داده‌های خود را از دست می‌دهند.

نکته: چهار رده‌ی آخر به حافظه‌ی ثانویه نیز معروف هستند.

یکی از مهم‌ترین وظایف سیستم عامل، مدیریت حافظه‌ی اصلی می‌باشد. منظور از حافظه‌ی اصلی، حافظه‌ای است که پردازنده برای دستیابی به دستورالعمل‌ها و داده‌ها مستقیماً به آن رجوع می‌کند (مثلاً RAM).

برخی از مسائلی که بخش مدیریت حافظه در سیستم عامل باید به آنها بپردازد عبارتند از:

ـ آیا فقط یک کاربر حق استفاده از حافظه را دارد یا چند کاربر به طور همزمان می‌توانند از حافظه استفاده کنند؟

ـ آیا در آنِ واحد فقط یک برنامه می‌تواند در حافظه باشد یا چندین برنامه می‌توانند به طور همزمان در حافظه باشند؟

ـ آیا به همه‌ی برنامه‌ها و کاربران قسمت‌های مساوی از حافظه تخصیص می‌یابد؟

ـ کدام قسمت‌ها در اختیار کدام برنامه‌ها هستند و کدام بخش‌های حافظه، خالی هستند؟

ـ آیا به یک برنامه می‌توان دو بخش مجزا از حافظه را تخصیص داد یا باید حتماً بخش‌های همجوار به برنامه‌ها داده شود؟

ـ اگر چند برنامه کاندیدای ورود به حافظه هستند، کدام‌یک باید انتخاب شوند؟

ـ اگر یک برنامه با اولویت بسیار بالا از راه رسید و در حافظه جای خالی وجود نداشت، کدام برنامه باید فداکاری کرده و جای خود را به وی بدهد؟

و مسائل بسیاری از این دست ...

پیوند آدرس (Address Binding)

آدرس‌ها در برنامه‌ی نوشته شده توسط برنامه‌نویس، معمولاً به فرم سمبولیک هستند. به عنوان مثال در روش استاده از متغیرها به جای آدرس حافظه، برنامه‌نویس از یک نام نمادین برای کلمات حافظه استفاده می‌کند. مثلاً در زبان C، برنامه‌نویس برای استفاده از حافظه دستور زیر را به کار می‌برد.

int i;

در این حالت نام i در واقع فقط یک اسم سمبولیک و نمادین برای دو بایت خاص از حافظه می‌باشد. اما سؤال اصلی این است که این دو بایت که در اختیار متغیرi می‌باشد واقعاً کجای حافظه و در چه آدرسی قرار دارند؟ سؤال دیگر این است که اگر این برنامه دوباره اجرا شود باز هم متغیر i در همان مکان از حافظه قرار داده می‌شود؟ همچنین اگر این برنامه بر روی سیستم دیگری با مشخصات و اندازه‌ی حافظه متفاوت اجرا شود، تکلیف متغیر i چیست؟

واقعیت این است که همان‌طور که دیدیم، برنامه‌نویس از آدرس‌ها سمبولیک (مانند متغیرها) استفاده می‌کند و مترجم وظیفه دارد که این آدرس‌های سمبولیک را به آدرس‌های قابل جابه‌جایی تبدیل کند. اما این پایان کار نیست و این بار یک بارکننده (Loader) باید این آدرس‌های قابل جابه‌جایی را به آدرس‌های مطلق تبدیل کند.

آدرس‌های فیزیکی و منطقی

آدرس منطقی همان آدرس تولید شده توسط پردازنده است. اما آدرس فیزیکی آدرس قابل رویت و قابل فهم برای واحد حافظه می‌باشد. برنامه کاربر همواره با آدرس منطقی سر و کار دارد و در نهایت هنگام دسترسی به حافظه، این آدرس به آدرس فیزیکی تبدیل می‌شود.

در واقع هنگامی که پیوند آدرس‌ها در زمان اجرا صورت می‌گیرد، آدرس‌های منطقی باید به آدرس‌های فیزیکی تبدیل شوند.

تخصیص حافظه

برای اینکه حافظه را به عنوان یک منبع در اختیار فرآیندها قرار دهیم، روش‌ها و راهکارهای مختلفی وجود دارد که در این بخش به بررسی آنها می‌پردازیم.

تک برنامگی

تخصیص حافظه به صورت یک پارچه یک روش ساده مدیریت حافظه می‌باشد که نیاز به پشتیبانی سخت‌افزار خاصی ندارد. در این سیستم‌ها عملکرد چند برنامگی در میان نیست و در هر لحظه یک کاربر و یک فرآیند وجود دارد. در این حالت کل فضای حافظه به سه بخش تقسیم می‌شود، بخشی از حافظه به طور ثابت در اختیار سیستم عامل است، بخشی از حافظه در اختیار برنامه قرار دارد و مابقی حافظه نیز بلااستفاده باقی می‌ماند.

در این روش وقتی یک فرآیند برای اجرا انتخاب می‌شود، کل فضای حافظه (به‌جز بخش سیستم عامل) را می‌تواند در اختیار بگیرد و هنگامی که فرآیند خاتمه یافت، کل حافظه را آزاد می‌کند.

نکته: یکی از معایب این روش عدم استفاده بهینه از حافظه می‌باشد به گونه‌ای که اگر یک فرآیند کوچک در حافظه قرار داشته باشد مابقی حافظه بلااستفاده می‌ماند.

نکته: در این روش برنامه‌ها از نظر اندازه، محدود به اندازه حافظه هستند و برنامه‌های بزرگتر از حافظه، هیچ‌گاه نمی‌توانند اجرا شوند.

نکته: با استفاده از تکنیکی موسوم به **جایگذاشت (Overlay)** می‌توان برنامه‌های بزرگ‌تر از حافظه را نیز در این حالت اجرا کرد. در این تکنیک، هر زمانی که داده یا کدی از برنامه موردنیاز است به حافظه منتقل می‌شود. در واقع تا جایی که حافظه گنجایش دارد از فرآیند موردنظر به حافظه آورده می‌شود، اما هنگامی که بخش دیگری از فرآیند موردنیاز است، آن بخش به حافظه آورده می‌شود.

نکته قابل توجه در این تکنیک این است که برای این منظور سیستم عامل هیچ پشتیبانی خاصی نمی‌کند و کاربر (برنامه‌نویس) همه این کار را انجام می‌دهد. در واقع برنامه‌نویس باید به طور دقیق و با علم به اندازه حافظه، بخش‌های دیگر برنامه را (در واقع شبیه به عملیات بار کردن فایل‌ها)، به حافظه دعوت کرده و آنها را اجرا و یا از آنها استفاده کند. در این حالت سیستم عامل فقط گمان می‌کند عملیات I/O در حال انجام است!

چندبرنامگی

در سیستم عامل‌های چندبرنامگی، در هر لحظه چندین فرآیند در حافظه قرار دارند. در این حالت تخصیص حافظه به دو روش انجام می‌شود:

۱ ـ روش بخش‌بندی ایستا (Static)

در این روش سیستم عامل یک بخش از حافظه را در اختیار می‌گیرد و مابقی حافظه می‌تواند در قالب بخش‌هایی با **اندازه ثابت** در اختیار برنامه‌ها قرار گیرد.

هر برنامه که اندازه آن مساوی یا کمتر از اندازه بخش باشد می‌تواند به داخل آن بار شود. اگر همه‌ی بخش‌ها پر باشند و هیچ‌یک از فرایندهای موجود در حافظه، در حالت اجرا یا آماده نباشند، سیستم عامل می‌تواند فرآیندی را به خارج از حافظه منتقل کرده و یک فرآیند دیگر را به درون حافظه بار کند تا پردازنده بیکار نماند.

روش بخش‌بندی ایستا عموماً به دو شیوه پیاده‌سازی می‌شود:

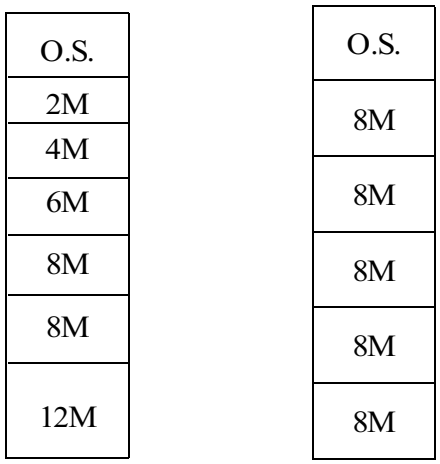
الف) حافظه به بخش‌های **مساوی** تقسیم می‌شود و در اختیار فرایندها قرار می‌گیرد. این شیوه دو نقص عمده دارد:

۱ـ ممکن است یک برنامه بزرگتر از آن باشد که در یک بخش قرار گیرد که البته می‌توان از تکنیک Overlay استفاده کرد.

۲ـ استفاده از حافظه اصلی قدری ناکارآمد می‌شود زیرا هر برنامه‌ای هر قدر هم که کوچک باشد، یک بخش کامل را اشغال می‌کند که به این گونه به هدر رفتن فضا، **تکه تکه شدن داخلی (Internal Fragmentation)** گویند.

ب) امکان استفاده از بخش‌های نامساوی هم وجود دارد. در این حالت بخش‌های مشخص شده لزوماً هم اندازه نیستند. با این امکان دو ایراد روش قبل تا حدودی کمتر می‌شوند به این ترتیب که هر فرآیند می‌تواند در کوچکترین بخشی وارد شود که در آن جای می‌گیرد. بنابراین تکه‌تکه شدن داخلی به حداقل می‌رسد.

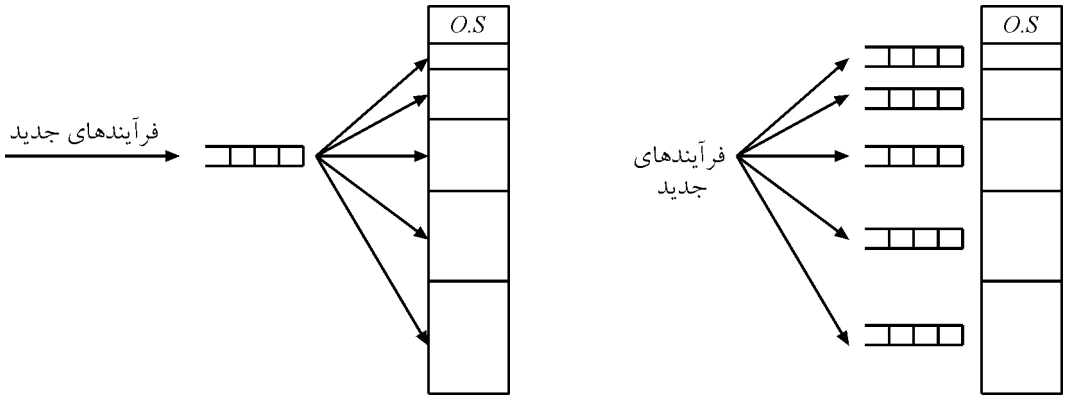
شکل زیر، یک مثال از این دو روش را نشان می‌دهد.



الف: قسمت‌های مساوی ب: قسمت‌های نامساوی
مثالی از بخش‌پذیری ایستای یک حافظه ۴۰ مگابایتی

با استفاده از بخش‌های نامساوی، دو راه برای تخصیص فرآیندها به بخش‌ها وجود دارد:
در **روش اول**، برای هر بخش از یک صف جداگانه استفاده می‌کنیم و وقتی فرآیندی وارد شد، با توجه به اندازه آن، به مناسب‌ترین صف وارد می‌شود (شکل الف). در این حالت فضای به هدر رفته به عنوان تکه تکه شدن داخلی، به حداقل می‌رسد اما گاهی از نظر سیستم بهینه نیست. مثلاً فرض کنید در صف مربوط به یک بخش با اندازه کوچک، چندین و چند فرایند منتظر باشند، اما صف مربوط به یک بخش با اندازه بزرگتر خالی باشد! پس بخش بزرگ بدون استفاده می‌ماند، در حالیکه تعدادی فرآیند کوچک منتظرند.

در **روش دوم** یک صف واحد برای همه فرآیندها به کار می‌رود. در این روش وقتی فرآیند به ابتدای صف می‌رسد کوچکترین بخش برای آن انتخاب می‌شود (شکل ب). در این حالت ممکن است یک فرآیند کوچک که در یک بخش کوچک جا می‌شود در یک بخش بزرگ قرار بگیرد و تکه تکه شدن داخلی را تشدید کند، زیرا وقتی این فرآیند به ابتدای صف می‌رسد، آن بخش کوچک آزاد نبوده و فقط همان بخش بزرگ آزاد باشد و فرآیند در آن بخش بزرگ قرار گیرد، در صورتی که اگر این فرآیند قدری صبر می‌کرد، ممکن بود بخش کوچک حافظه آزاد شود...



الف: یک صف برای هر بخش ***ب: یک صف واحد برای همه بخش‌ها***

توزیع فرآیندها در روش بخش‌بندی ایستا

نکته: در حالت کلی بخش‌بندی ایستا (چه با اندازه‌های مساوی و چه با اندازه‌های نامساوی) دو نقص عمده دارد:

۱ـ تعداد فرآیندهای فعال در سیستم محدود به تعداد بخش‌های تعریف شده در زمان ایجاد سیستم است.

۲ـ کارهای با اندازه کوچک، بخش‌بندی ایستا را ناکارآمد می‌کنند و باعث تکه‌تکه شدن داخلی می‌شوند.

نکته: امروزه از روش بخش‌بندی ایستا استفاده نمی‌شود. در گذشته این ایده در سیستم عامل OS/MFT برای مین‌فریم‌های IBM استفاده می‌شد.

روش بخش‌بندی پویا (Dynamic)

برای غلبه بر بعضی از مشکلات روش بخش‌بندی ایستا، روشی با عنوان بخش‌بندی پویا ابداع شد. ایده کلی این روش این است که بخش‌های استفاده شده دارای طول متغیر باشند و تعداد آنها نیز ثابت نباشد. در واقع اگر فرآیندی به داخل حافظه آورده می‌شود، دقیقاً به همان اندازه‌ای که نیاز دارد، حافظه به آن اختصاص می‌یابد. در این روش سیستم عامل باید دقیقاً بداند کدام قسمت‌های حافظه، آزاد و کدام قسمت‌ها اشغال هستند. در ابتدا کل حافظه همانند یک بلوک بزرگ آزاد در نظر گرفته می‌شود و وقتی فرآیندی درخواست حافظه می‌کند، به همان اندازه، حافظه به آن تخصیص می‌دهیم و مابقی حافظه برای درخواست‌های بعدی آزاد می‌ماند. به همین ترتیب فرآیند به حافظه وارد می‌شوند و هرگاه فرآیندی به پایان رسید، حافظه‌ای را که در اختیار داشته به سیستم عامل برمی‌گرداند و سیستم عامل آن بخش را آزاد به حساب می‌آورد.

نکته: روش بخش‌بندی پویا به خوبی آغاز می‌شود، ولی در نهایت حفره‌های کوچک زیادی در حافظه

ایجاد می‌شود، این پدیده را **تکه تکه شدن خارجی (External Fragmentation)** گویند. این مسئله از آنجا ناشی می‌شود که فرآیندی که حافظه را ترک می‌گوید، یک بخش آزاد بر جای می‌گذارد و احتمال اینکه فرآیند جایگزین دقیقاً به همان اندازه باشد، بسیار ضعیف است، پس یک فرآیند کوچک‌تر در آنجا قرار می‌گیرد و مابقی حافظه، آزاد باقی می‌ماند که احتمالاً آنقدر کوچک است که دیگر هیچ فرآیندی در آن، جا نمی‌شود!!!

تفاوت تکه تکه شدن داخلی با تکه تکه شدن خارجی

اجازه دهید یکبار برای همیشه مشکل «تفاوت تکه تکه شدن داخلی و خارجی» را حل کنیم.

تکه تکه شدن داخلی زمانی رخ می‌دهد که حافظه از قبل مرزبندی شده باشد و در هر قسمت بتوان فقط یک قطعه را قرار داد. درا ین حالت چون احتمالاً قطعه‌ای که برای یک جایگاه انتخاب شده است قدری از آن کوچکتر است، در انتهای جایگاه مقداری فضای بلااستفاده به وجود خواهد آمد. به این مشکل تکه تکه شدن داخلی گویند.

اما وقتی از قبل مرزبندی مشخصی برای حافظه در نظر نگیریم احتمال وقوع تکه تکه شدن خارجی وجود دارد. در این حالت احتمالاً فضاهای خالی و قابل استفاده حافظه به صورت جایگاه‌های کوچک، لابه‌لای فرآیندهای پخش شده‌اند. که مجموع این جایگاه‌های کوچک پراکنده احتمالاً نیاز ما را برآورده می‌کنند، اما چون همجوار نیستند بلااستفاده باقی می‌مانند. روش فشرده‌سازی (Compaction) در صورت اجرا شدن، این مشکل را برطرف می‌کند (در ادامه، بررسی می‌شود). دقت کنید به تکه تکه شدن خارجی گاهی به اختصار تکه تکه شدن نیز گفته می‌شود.

روش‌های نگهداری وضعیت حافظه

بخش مدیریت حافظه سیستم عامل باید از وضعیت حافظه و قسمت‌های آزاد و اشغال آن مطلع باشد. برای این منظور معمولاً از دو روش استفاده می‌شود:

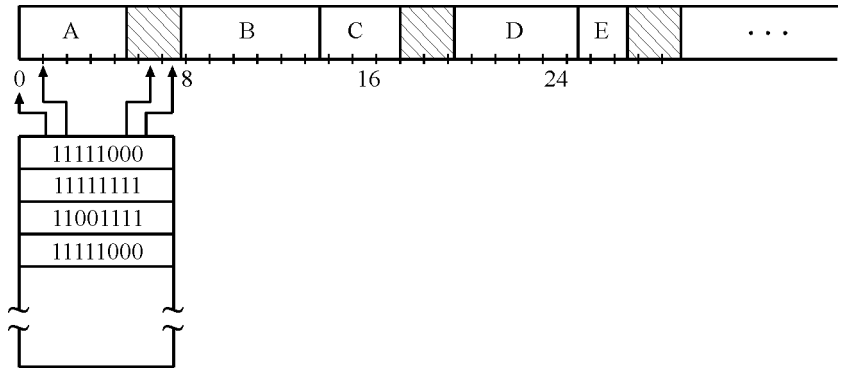
۱ـ روش Bitmap (نقشه بیتی)

۲ـ روش Linked List (لیست پیوندی)

روش Bitmap

در ایده حافظه را به واحدهای کوچکی تقسیم می‌کنند. به طوری که هر بخش در واقع کوچکترین واحد تخصیص می‌باشد (مثلاً یک بلوک یا یک کلمه). سپس متناظر با هر واحد تخصیص، یک بیت در یک دنباله از بیت‌ها در نظر گرفته می‌شود. حال اگر واحدی آزاد باشد، بیت متناظر با آن 0 و اگر اشغال باشد، بیت متناظر با آن 1 تنظیم می‌شود.

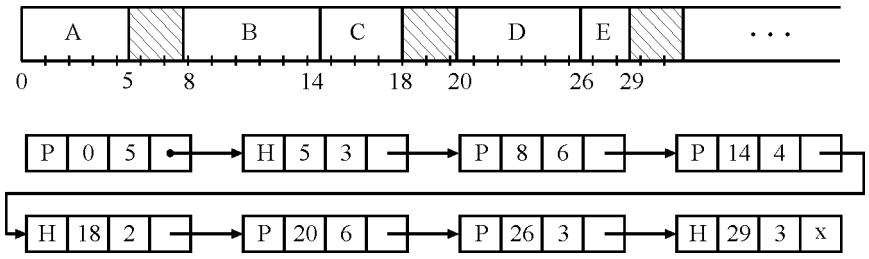
شکل زیر این روش را نشان می‌دهد:



با آنکه پیاده سازی این روش ساده است ولی ایراد این روش آن است که اگر بخواهیم پروسسی را که به K واحد فضا نیاز دارد، به حافظه آوریم، در این صورت مدیر حافظه باید نگاشت بیتی را مورد جستجو قرار دهد تا یک سری متوالی شامل K بیت صفر را پیدا کند و این کار بسیار کند است.

روش Linked List

در این روش برای نگهداری وضعیت حافظه، از یک لیست پیوندی استفاده می‌شود. هر گره‌ی لیست پیوندی یا یک حفره (فضای) آزاد است یا یک فرآیند را در حافظه نشان می‌دهد. در هر گره‌ی این لیست باید اطلاعاتی از قبیل محل شروع قطعه، طول قطعه، وضعیت پر یا خالی بودن قطعه و آدرس شروع قطعه بعدی ذخیره و نگهداری شود.



نکته: یک عیب بزرگ روش Linked List برای نگهداری وضعیت حافظه این است که قدری زمان‌بر است. مثلاً وقتی یک فرآیند خاتمه می‌یابد، پیدا کردن همسایه‌های آن جهت ادغام احتمالی حفره‌ها، عملی وقت‌گیر است.

نکته: یک روش برای مقابله با تکه تکه شدن خارجی، روش فشرده سازی (Compaction) است. در این روش سیستم عامل همه فرآیندها را طوری جابه‌جا می‌کند که همگی کنار هم قرار بگیرند و تمام حافظه‌ی آزاد موجود به صورت یکپارچه درآید. البته این کار بسیار زمانگیر و هزینه‌بر می‌باشد، زیرا سیستم عامل باید پیوند آدرس‌های فرآیندها را به درستی تغییر دهد.

روش‌های تخصیص حافظه به فرآیندها

برای تخصیص حافظه برای فرآیندهای تازه وارد، روش‌های مختلفی وجود دارد که هر یک پیامدهای خاص خود را دارند. در این حالت فرض می‌کنیم پس از مدتی تعدادی فرآیند در حافظه قرار گرفته‌اند و بین آنها تعدادی بخش آزاد با اندازه‌های مختلف وجود دارد، حال می‌خواهیم بررسی کنیم فرآیند جدیدی که به سیستم وارد شده در کدام قسمت قرار داده می‌شود.

روش First Fit

در این روش هنگامی که یک درخواست برای حافظه از راه رسید، سیستم عامل حافظه را از ابتدا جستجو کرده و اولین فضای حافظه‌ای که بتواند فرآیند را در خود بگنجاند، انتخاب می‌شود.
در این حالت چون جستجو همواره از ابتدای حافظه آغاز می‌شود، تراکم فضای اشغال شده معمولاً در ابتدای حافظه بیشتر است.

روش Next Fit

این روش شبیه روش First Fit است با این تفاوت که جستجو برای یافتن اولین محل مناسب، همواره از ابتدای حافظه آغاز **نمی‌شود**، بلکه جستجو از محل آخرین تخصیص به بعد شروع می‌شود.

روش Best Fit

در این روش، مدیریت حافظه با گرفتن یک تقاضا کل حافظه را جستجو می‌کند تا بتواند کوچکترین فضای آزاد را پیدا کند که فرآیند موردنظر در آن جای بگیرد.
در واقع ایده‌ی این روش این است که فضاهای بزرگ (که بعدها ممکن است به آنها نیاز داشته باشیم) را نباید تقسیم کرد و فضایی را برمی‌گزینند که دارای نزدیک‌ترین اندازه به اندازه فرآیند موردنظر باشد.

روش Worst Fit

در این روش نیز باید کل فضای حافظه جستجو شود تا همیشه بزرگترین فضای موجود را به هر فرآیند تخصیص دهیم!
در واقع ایده این روش این است که پس از تخصیص بزرگترین حفره به یک فرآیند، فضای باقیمانده آنقدر بزرگ هست که باز هم بتوان از آن استفاده کرد. در صورتی که در Best Fit، پس از آن که یک حفره به یک فرآیند اختصاص داده شد، فضای باقیمانده آنقدر کوچک است که بعدها به کار هیچ فرآیندی نمی‌آید.

نکته: روش‌های Best Fit و Worst Fit نسبت به روش‌های First Fit و Next Fit قدری کندتر هستند، زیرا در این دو روش تمام حافظه باید جستجو شود.

نکته: در روش Best Fit حافظه پر از حفره‌های بسیار کوچکی می‌شود که به هیچ کار نمی‌آیند. در روش Worst Fit نیز ممکن است فرآیندهای بزرگ دچار گرسنگی شوند، زیرا قسمت‌های بزرگ‌تر،

زودتر تخصیص داده شده و کوچک می‌شوند.

روش Quick Fit

در این ایده لیست‌های جداگانه‌ای برای فرآیندهای با اندازه‌های متداول تهیه می‌شود .
به عنوان مثال در یک جدول، درایه‌ی اول یک اشاره‌گر است به ابتدای یک لیست از حفره‌های ۴KB و درایه دوم یک اشاره‌گر به ابتدای یک لیست از حفره‌های ۸KB و الی آخر. در این روش به سرعت می‌توان به یک فرآیند فضا تخصیص داد.

روش رفاقتی (Buddy)

در سیستم رفاقتی اندازه بخش‌های حافظه، همگی توان صحیحی از ۲ هستند. مانند ۱KB، ۲KB یا ۴KB . در این حالت برای شروع، تمامی فضای موجود برای تخصیص به عنوان یک بلوک واحد به اندازه ۲^u در نظر گرفته می‌شود. حال اگر درخواستی با اندازه s مطرح شود به طوری که $۲^{u-۱} < s \leq ۲^u$ باشد، تمامی بلوک تخصیص می‌یابد، در غیر این صورت این بلوک به دو رفیق با اندازه‌های مساوی $۲^{u-۱}$ تقسیم می‌شود. اگر $۲^{u-۲} < s \leq ۲^{u-۱}$ باشد، یکی از این دو رفیق به فرآیند تخصیص می‌یابد، در غیر این صورت یکی از دو رفیق به دو قسمت مساوی تقسیم می‌شود و این روال ادامه می‌یابد.

فضای ۱ مگابایتی آزاد	۱M					
درخواست A =۱۰۰K	A	۱۲۸K	۲۵۶K	۵۱۲K		
درخواست A =۲۰۰K	A	۱۲۸K	B	۵۱۲K		
درخواست A =۶۰K	A	C	۶۴K	B	۵۱۲K	
درخواست A =۲۰۰K	A	C	۶۴K	B	D	۲۵۶K
درخواست B	A	C	۶۴K	۲۵۶K	D	۲۵۶K
درخواست A	۱۲۸K	C	۶۴K	۲۵۶K	D	۲۵۶K
درخواست E =۷۵K	E	C	۶۴K	۲۵۶K	D	۲۵۶K
آزاد سازی C	E	۱۲۸K	۲۵۶K	D	۲۵۶K	
آزاد سازی E	۵۱۲K			D	۲۵۶K	
آزاد سازی D	۱M					

مثالی از روش رفاقتی (از کتاب استالینگز)

نکته: روش رفاقتی مشکل تکه تکه شدن داخلی دارد.

مدیریت حافظه به روش قطعه‌بندی (Segmentation)

یکی دیگر از روش‌های مدیریت حافظه برای سیستم‌های چندبرنامگی، روش قطعه‌بندی است.
در این روش فرآیندها به تعدادی قطعه تقسیم می‌شوند و هیچ لزومی ندارد که اندازه قطعه‌ها یکسان باشند. این تقسیم‌بندی توسط برنامه‌نویس انجام می‌شود. در واقع برنامه‌نویس (یا حتی کامپایلر) اطلاعات مرتبط با هم را در یک قطعه قرار می‌دهد (به عنوان مثال زیر روال‌ها یا حتی داده‌های مربوط به هم).

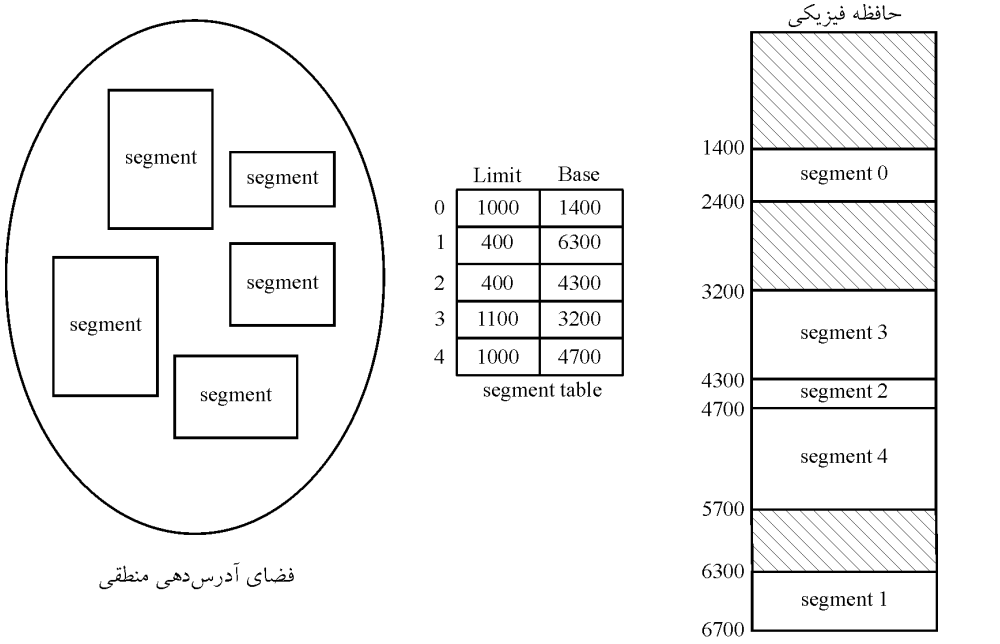
از آنجا که طول قطعات با هم برابر نیستند، روش قطعه‌بندی همانند روش بخش‌بندی پویا است. در واقع در این روش، یک فرآیند به چندین قطعه تقسیم می‌شود که کماکان برای اجرای فرآیند، همه قطعات باید به داخل حافظه آورده شوند، اما این قطعات لزوماً نباید در حافظه همجوار باشند.

نکته: در روش قطعه‌بندی، هر فرآیند باید یک جدول قطعه (Segment Table) داشته باشد که در آن به ازای هر قطعه یک درایه وجود دارد و مشخص می‌کند هر قطعه در کدام بخش از حافظه‌ی اصلی قرار گرفته است.

نکته: روش قطعه‌بندی دقیقاً مشابه روش بخش‌بندی پویا است. با این تفاوت که هر برنامه می‌تواند بیش از یک بخش را اشغال کند و در ضمن لزومی ندارد این بخش‌ها پیوسته باشند.

نکته: در روش قطعه‌بندی، تکه تکه شدن داخلی **نداریم**، اما همانند روش بخش‌بندی پویا، تکه‌تکه شدن خارجی داریم، البته از آنجا که یک فرآیند به قطعات کوچکتری تقسیم می‌شود، میزان این تکه‌تکه شدن خارجی کمتر است.

شکل زیر نمونه‌ای پیاده سازی شده از این سیستم را نشان می‌دهد:



فرآیند تبدیل آدرس منطقی به آدرس فیزیکی

در این روش آدرس‌های منطقی به صورت زیر هستند:

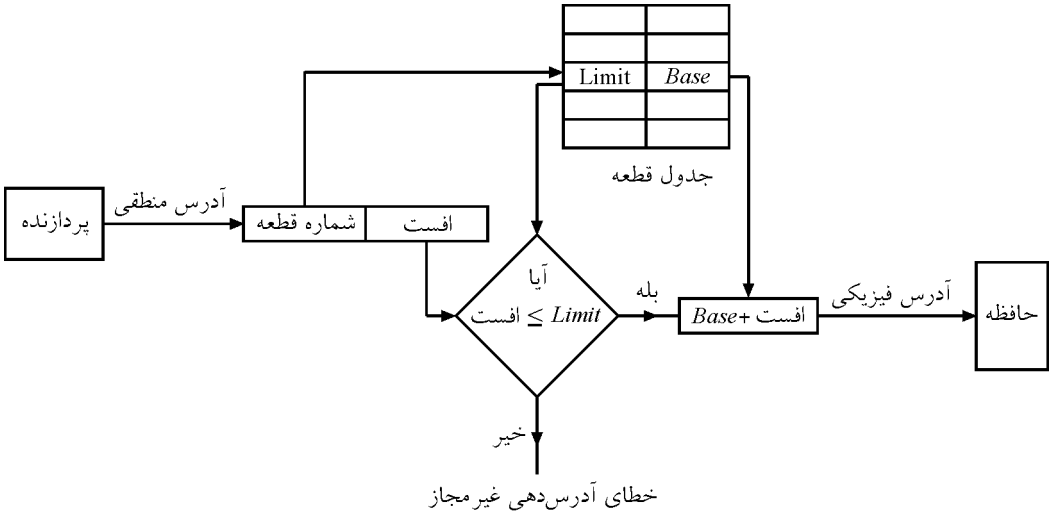
	n بیت	m بیت
: آدرس منطقی	شماره قطعه	آفست

که باید از n بیت مربوط به شماره قطعه برای ارجاع به جدول قطعه استفاده شود و آدرس فیزیکی شروع این قطعه از جدول قطعه استخراج شده و به ابتدای آفست متصل گردد.

نکته: همانطورکه ذکر شد آدرس منطقی از دو قسمت آفست و شماره قطعه تشکیل شده است. جهت تبدیل این آدرس به آدرس فیزیکی از جدول قطعه استفاده می‌شود. جدول قطعه به ازای هر فرآیند وجود دارد و به ازای تعداد قطعه‌های هر فرآیند درایه دارد. در هر درایه، آدرس شروع قطعه در حافظه اصلی و طول قطعه ذخیره شده است. روال تبدیل آدرس به این صورت است که از آدرس منطقی، شماره قطعه استخراج می‌شود و با استفاده از شماره قطعه به عنوان اندیس، به سراغ جدول قطعه می‌رویم. آفست موجود در آدرس منطقی باید کمتر از طول قطعه باشد، به همین جهت ابتدا آفست را با طول قطعه مقایسه می‌کنیم و اگر آفست بزرگتر از طول قطعه بود، وقفه خطا صادر می‌شود. در غیر این صورت آدرس شروع قطعه با آفست جمع جبری شده و آدرس فیزیکی به دست می‌آید.

نکته: در جدول قطعه به قسمت طول قطعه، **حد (Limit)** و به آدرس شروع قطعه، **پایه (Base)** گویند.

نکته: روال تبدیل آدرس در قطعه‌بندی در شکل زیر نشان داده شده است.



تبدیل آدرس منطقی به فیزیکی در قطعه‌بندی

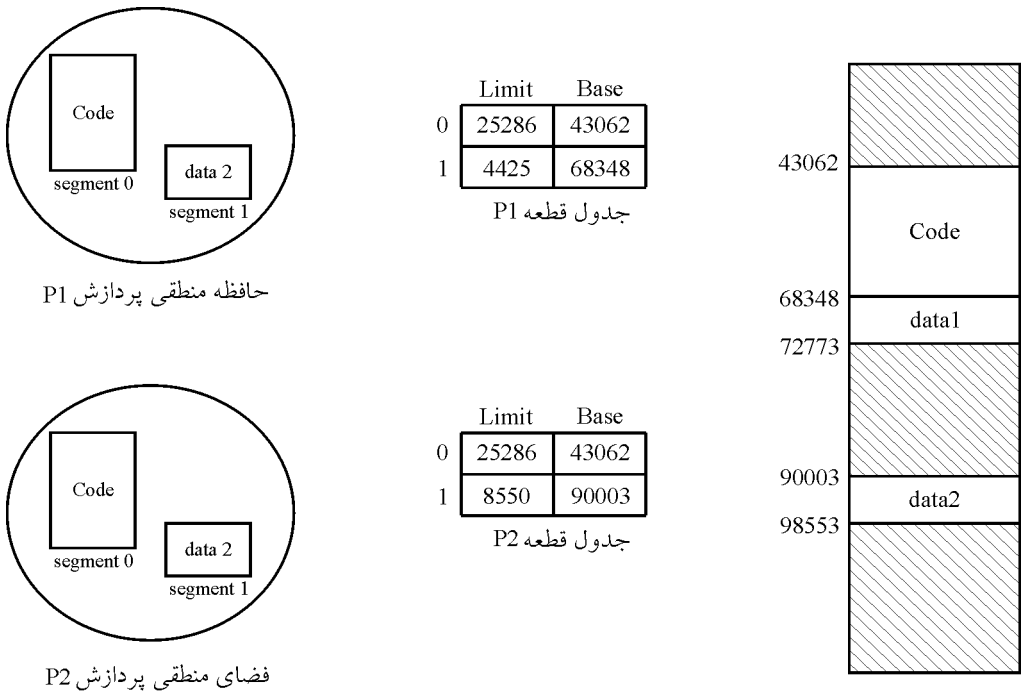
نکته: تکنیک استفاده از TLB همانند صفحه‌بندی می‌تواند در قطعه‌بندی نیز جهت افزایش سرعت دسترسی به حافظه استفاده شود.

اشتراک در قطعه‌بندی

یکی از مزایای قطعه‌بندی، امکان به اشتراک گذاشتن قطعات بین فرآیندها است.

به عنوان مثال دو فرآیند را در نظر بگیرید که می‌خواهند کد یکسانی را اجرا کنند اما هر کدام داده‌های مجزا و مخصوص به خود را دارند، در این صورت فرآیندها می‌توانند قطعات مربوط به کد را با هم به اشتراک گذارند. البته باید دقت داشت در حالتی که یک قطعه بین چند فرآیند به اشتراک گذاشته شود، معمولاً آن قطعه نباید تغییر کند (در واقع باید فقط خواندنی باشد).

شکل زیر نمونه‌ای از اشتراک را نمایش می‌دهد.



نکته: در جداول قطعه، معمولاً تعدادی بیت‌های کنترلی و حفاظتی هم وجود دارند که به عنوان مثال مشخص می‌کنند یک قطعه فقط خواندنی است یا خواندنی / نوشتنی یا مشخص می‌کنند یک قطعه قابلیت اجرا دارد یا خیر.

مدیریت حافظه به روش صفحه‌بندی (Paging)

یک راه حل کلی جهت مقابله با تکه تکه شدن خارجی این است که اجازه دهیم یک فرآیند در قسمت‌های غیرهمجوار در حافظه قرارگیرد. یکی از روش‌هایی که از این ایده استفاده می‌کند، تکنیک

صفحه‌بندی است. در این روش حافظه به بخش‌های با اندازه‌ی یکسان به نام **قاب (Frame)** تقسیم می‌شود. از طرفی برنامه‌ها نیز به قسمت‌های مساوی و هم اندازه با قاب‌ها تقسیم می‌شوند که به آنها **صفحه (Page)** می‌گویند. حال هنگامی که برنامه‌ای به حافظه منتقل می‌شود باید **تمام** صفحاتش به داخل قاب‌های خالی آورده شوند. در این حالت اصلاً نیازی نیست صفحات مربوط به یک فرآیند در قاب‌های همجوار قرار گیرند.

مزیت عمده این روش از بین بردن تکه تکه شدن خارجی و به حداقل رساندن تکه تکه شدن داخلی می‌باشد، اما در عوض عملیات محاسبه آدرس‌ها و مدیریت این صفحات قدری هزینه‌بر و زمان‌گیر است.

نکته: برای پیاده‌سازی این روش به پشتیبانی سخت‌افزار نیاز است.

نکته: این روش از دید کاربر و برنامه‌نویس مخفی می‌ماند.

نکته: برای پیاده‌سازی این روش و مدیریت صفحه‌ها و از همه مهم‌تر تبدیل و نگاشت آدرس‌ها باید یک جدول صفحه به **ازای هر فرآیند** در نظر گرفت. در واقع جدول صفحه‌ی هر فرآیند دارای یک درایه به ازای هر صفحه می‌باشد که مشخص می‌کند هر صفحه از یک فرآیند در کدام قاب حافظه نگهداری می‌شود.

نکته: برای ساده‌تر شدن صفحه‌بندی، اندازه قاب‌ها و صفحه‌ها را به صورت توان صحیحی از ۲ در نظر می‌گیرند.

نکته: نقطه ضعف اصلی مکانیزم صفحه‌بندی این است که اگر فقط احتیاج به ناحیه بسیار کوچکی از حافظه باشد، در این صورت مقداری از فضای حافظه تلف می‌شود، زیرا کوچکترین واحدی از حافظه که می‌توان آن را به استفاده کننده اختصاص داد، یک صفحه است.

تبدیل آدرس در صفحه‌بندی

فرض کنیم اندازه صفحات، توانی از ۲ باشند. در این حالت آدرس‌های تولید شده توسط CPU (آدرس منطقی یا مجازی) از دو بخش تشکیل شده‌اند: شماره صفحه و آفست (انحراف).

n بیت سمت چپ شماره صفحه و m بیت سمت راست آفست را نشان می‌دهد.

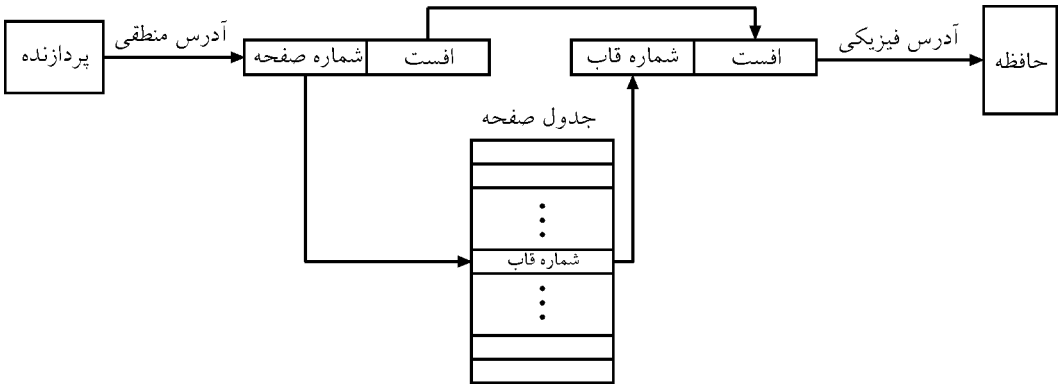
	n بیت	m بیت
آدرس منطقی :	شماره صفحه	آفست

با استفاده از شماره صفحه (n بیت سمت چپ) به سراغ درایه مربوطه در جدول صفحه می‌رویم تا شماره قاب موردنظر در حافظه را استخراج کنیم.

برای به دست آوردن آدرس فیزیکی، شماره قاب را به جای شماره صفحه در آدرس منطقی قرار

می‌دهیم و m بیت سمت راست (آفست) را بدون تغییر باقی می‌گذاریم.

نکته: نحوه تبدیل آدرس‌های منطقی به آدرس‌های فیزیکی در تکنیک صفحه‌بندی را می‌توان در شکل زیر مشاهده کرد:



مفهوم جدول صفحه

نکته: در این حالت، قسمت شماره صفحه (n بیت سمت چپ) لزوماً با شماره قاب هم اندازه نیست، معمولاً شماره قاب از نظر طول آدرس بلندتر از شماره صفحه می‌باشد.

به عنوان یک مثال از نحوه عملکرد صفحه‌بندی، شکل زیر را در نظر بگیرید. در این مثال کل فرآیند به ۴ صفحه تقسیم و حافظه فیزیکی نیز از ۸ قاب تشکیل شده است. در این مثال نحوه مقداردهی جدول صفحه به روشنی قابل درک است.

فرآیند

صفحه 0
صفحه 1
صفحه 2
صفحه 3

0	1
1	6
2	3
3	7

جدول صفحه

حافظه

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

مثالی از صفحه‌بندی

نکته: در ایده‌ی صفحه‌بندی تکه تکه شدن خارجی نداریم اما ممکن است قدری تکه تکه شدن داخلی داشته باشیم و آن هم در هر فرآیند و به ازای آخرین صفحه رخ می‌دهد. در واقع چون فرآیند می‌تواند هر طولی داشته باشد، ممکن است مضرب صحیحی از اندازه صفحه‌ها نباشد و آخرین صفحه قدری خالی بماند. به همین دلیل **به طور متوسط نیم صفحه به ازای هر فرآیند** تکه تکه شدن داخلی خواهیم داشت.

نکته: جهت کاهش تکه تکه شدن داخلی در روش صفحه‌بندی، می‌توان اندازه صفحه‌ها را کوچک کرد، اما در این صورت تعداد صفحات یک فرآیند افزایش یافته و در نتیجه اندازه جدول صفحه بزرگتر می‌شود.

ساختمان جدول صفحه (Page Table)

کارکرد اصلی جدول صفحه، تبدیل و نگاشت آدرس‌های مجازی به آدرس‌های فیزیکی می‌باشد. از نظر ریاضی جدول صفحه در واقع فقط یک تابع است که ورودی آن شماره صفحه مجازی و خروجی آن شماره قاب فیزیکی می‌باشد.

اما در این بین دو مسئله اساسی وجود دارد:

۱ـ جدول صفحه می‌تواند بسیار بزرگ شود.

۲ـ نگاشت آدرس‌ها باید بسیار سریع صورت گیرد.

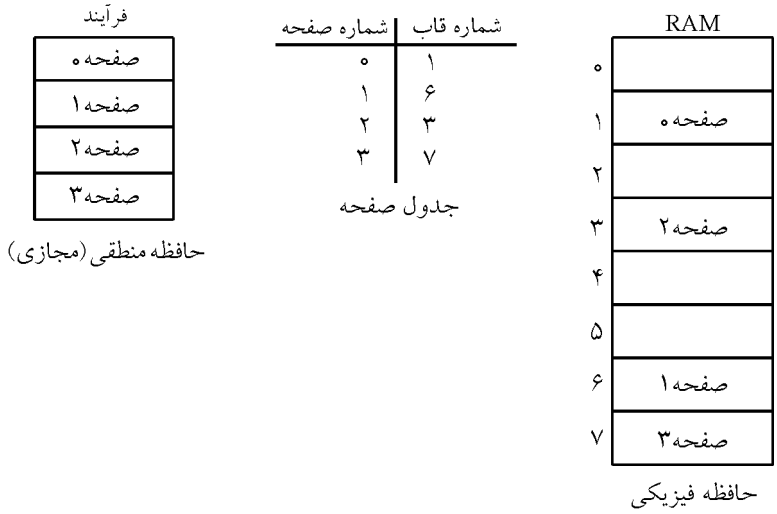
ساده‌ترین و ابتدایی‌ترین راه برای پیاده‌سازی جدول صفحه، استفاده از ثبات‌های سخت‌افزاری پرسرعت می‌باشد. در واقع در این روش به ازای هر درایه جدول صفحه به یک ثبات نیاز داریم. با این کار سرعت نگاشت و تبدیل آدرس‌ها بسیار بالا می‌رود. اما این روش هنگامی کاربرد دارد که تعداد صفحات به طرز بسیار معقولی کم باشند. این روش بسیار گران تمام می‌شود به همین جهت بیشتر کامپیوترهای امروزی جدول صفحه را در حافظه اصلی نگهداری می‌کنند و فقط یک ثبات به عنوان اشاره‌گر به محل جدول صفحه در CPU وجود دارد. به این ثبات PTBR (Page Table Base Register) گویند. در این حالت به هنگام Context Switching، فقط مقدار این ثبات عوض می‌شود. (باید مقدار این ثبات به ازای هر فرآیند در PCB نگهداری شود).

نکته: در حالتی که جدول صفحه را در حافظه اصلی نگهداری می‌کنیم، برای هر بار دسترسی به حافظه، باید ۲ بار به آن مراجعه کنیم، زیرا ابتدا باید شماره قاب را از جدول صفحه به دست آورده، سپس به محل موردنظر دسترسی پیدا کنیم. بنابراین سرعت دسترسی به حافظه با ضریب ۲ کاهش می‌یابد.

جداول صفحه چندسطحی (Multi Level Page Table)

با بزرگ شدن فرآیندها و حافظه کامپیوترها، اندازه جدول صفحه بسیار بزرگ می‌شود. برای کوچک کردن اندازهٔ جدول صفحه می‌توان اندازه صفحات را بزرگ کرد. اما در این صورت تکه تکه شدن داخلی افزایشی می‌یابد. جهت حل مشکل ذخیره کردن جداول صفحه، با اندازه زیاد، در حافظه کامپیوتر، راه حل استفاده از جداول صفحه چند سطحی ابداع گردید. برای درک این ساختار، به مثال‌های زیر توجه کنید:

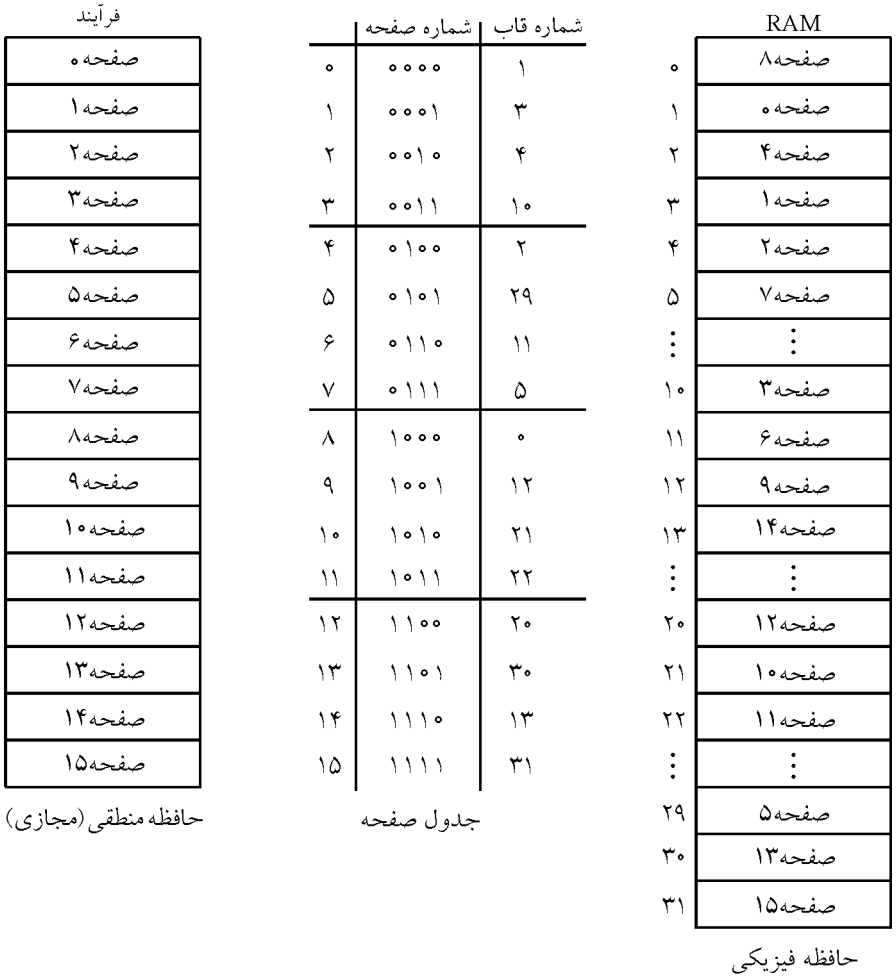
مثال: اگر فرآیندی که از ۴ صفحه منطقی تشکیل شده است به صورت زیر در قاب‌های حافظه فیزیکی قرار گرفته باشد، آنگاه جدول صفحه به صورت زیر درخواهد آمد:



در راه حل فوق، از یک جدول صفحه تک سطحی، برای مدیریت صفحه بندی فرآیند استفاده شده است. فرآیند فوق شامل ۴ صفحه می‌باشد و واضح است که جدول صفحه نیز باید شامل ۴ سطر باشد. برای مدیریت صفحه‌ها و نگاشت آدرس‌ها باید یک جدول صفحه به ازای هر فرآیند در نظر گرفت. در واقع جدول صفحه هر فرآیند دارای یک درایه به ازای هر صفحه می‌باشد که مشخص می‌کند هر صفحه از یک فرآیند در کدام قاب حافظه نگهداری می‌شود.

مثال: فرآیندی شامل ۱۶ صفحه می‌باشد، اگر اندازه جدول صفحه دارای محدودیت باشد و حداکثر هر جدول صفحه بتواند دارای ۴ سطر باشد، آنگاه جدول صفحه این فرآیند چند سطحی خواهد بود؟ (اندازه صفحات ۸ بایت است.)

پاسخ: واضح است که اگر اندازه جدول صفحه دارای محدودیت نباشد، این فرآیند دارای یک جدول صفحه تک سطحی با ۱۶ سطر خواهد بود. شکل زیر جدول صفحه تک سطحی این فرآیند را نشان می‌دهد:

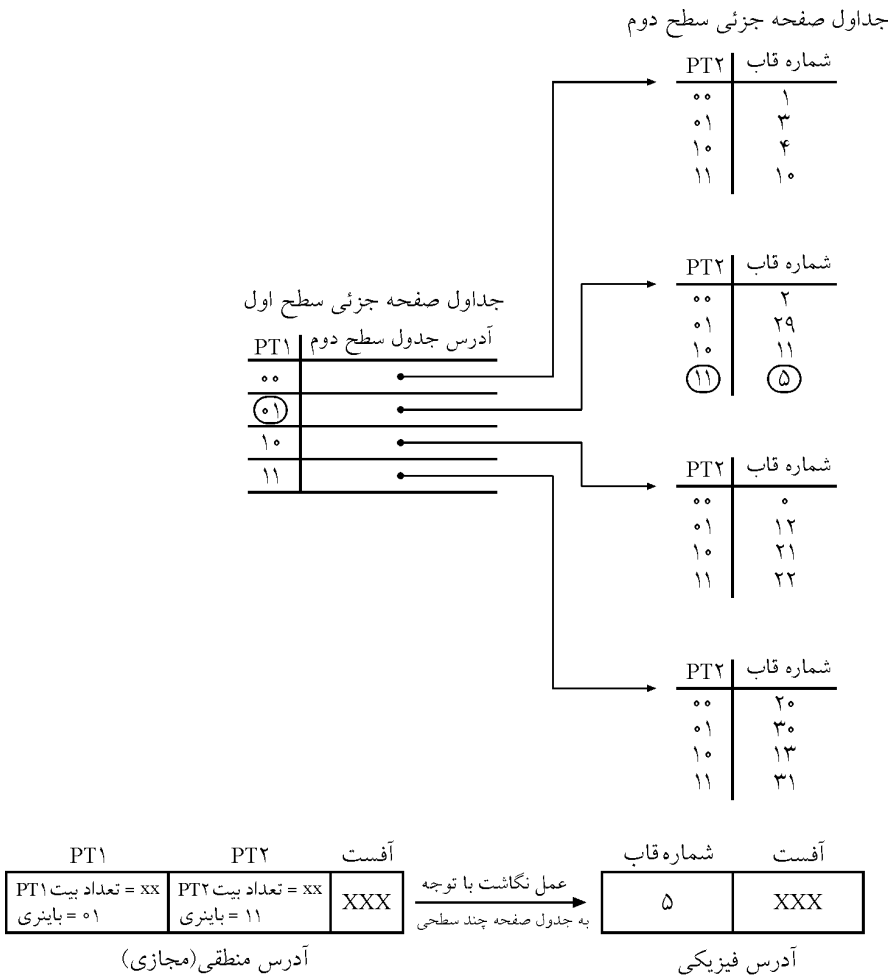


$$\text{بیت ۴} = \log_2^{2^4} = \log_2^{\text{تعداد صفحات فرآیند}} = \text{تعداد بیت شماره صفحه}$$

$$\text{بیت ۳} = \log_2^{2^3} = \log_2^{8} = \log_2^{\text{اندازه صفحه}} = \text{تعداد بیت آفست}$$

اما اگر اندازه جدول صفحه دارای محدودیت باشد و حداکثر بتواند دارای ۴ سطر باشد، واضح است که در این حالت باید از راه حل جدول صفحه چند سطحی استفاده شود. بنابراین با ۱۶ صفحه مواجه هستیم که هر ۴ صفحه آن می‌تواند داخل یک جدول صفحه جزئی قرار بگیرد.

در شکل زیر واضح است که با توجه به شرایط مسأله یک جدول صفحه دو سطحی مسأله را حل می‌کند.



وقتی که یک آدرس مجازی به مدیر حافظه می‌رسد، ابتدا فیلد PT1 این آدرس استخراج شده و به کمک آن درایه مورد نظر در جدول سطح اول پیدا می‌شود. هر یک از درایه‌های این جدول، نماینده یک جدول سطح دوم می‌باشد. در واقع از مقادیر موجود در درایه‌های جدول سطح اول برای پیدا کردن جدول سطح دوم مناسب استفاده می‌شود. اکنون از فیلد PT2 به عنوان اندیس جدول سطح دوم استفاده و به کمک آن، شماره فیزیکی قاب حافظه استخراج می‌شود و در نهایت بخش آفست نیز به این آدرس متصل می‌گردد.

نکته: در مثال فوق مشاهده می‌شود که برای دسترسی به هر قاب فقط به ۲ جدول نیاز است. یکی جدول سطح اول و دیگری یکی از جداول سطح دوم. در واقع مزیت این روش از اینجا ناشی می‌شود که **فقط جداولی به حافظه آورده می‌شوند که مورد نیاز هستند.**

دقت کنید این مفهوم با مفهوم حافظه مجازی متفاوت است. در مبحث حافظه مجازی مسأله این است که همه صفحات یک فرآیند به حافظه آورده نشوند. اما اینجا **جداول** صفحه به حافظه آورده نمی‌شوند.
توجه: به جداول موجود در سطوح مختلف جدول چند سطحی، **جداول جزئی** نیز گفته می‌شود. مجموع این جداول جزئی، جدول چند سطحی را ایجاد می‌کنند.
روابط زیر در جدول صفحه چند سطحی برقرار است:

تعداد صفحات فرآیند : f

تعداد سطرهای جدول صفحه جزئی = r

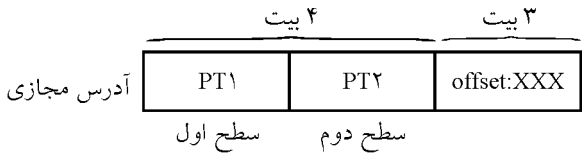
$$\text{تعداد جداول صفحه جزئی در سطح دوم} = \frac{\text{تعداد صفحات فرآیند}}{r} = \frac{f}{r} = \frac{۱۶}{۴} = ۴$$

$$\text{تعداد جداول صفحه جزئی در سطح اول} = \frac{\text{تعداد جداول صفحه در سطح دوم}}{r} = \frac{۴}{۴} = ۱$$

روال فوق گویای این مفهوم نیز می‌باشد، که این تقسیم متوالی تا جایی ادامه پیدا می‌کند که خارج قسمت کوچکتر یا برابر یک شود. یعنی به یک جدول برسیم، به اندازه محدودیت.
همچنین این نتیجه، تعداد سطوح جدول چند سطحی را نیز مشخص می‌کند. یعنی تعداد تقسیم از ابتدا تا به انتها با شرط مذکور. در اینجا تعداد تقسیم متوالی برابر ۲ است، بنابراین تعداد سطوح جدول چند سطحی برابر ۲ است.
توجه: روال فوق را نیز می‌توان در مفهوم لگاریتم نیز جستجو کرد، در واقع تقسیم فوق مفهوم لگاریتم را پیاده سازی می‌کند:

$$\text{تعداد سطوح جدول صفحه چند سطحی} = d = \left\lceil \log \frac{f}{r} \right\rceil = \left\lceil \log_{۴} ۱۶ \right\rceil = \left\lceil \log_{۲^۲} ۲^۴ \right\rceil = ۲$$

بنابراین در این مثال، آدرس‌های مجازی به سه بخش تقسیم می‌شوند.



$$\text{اندازه فرآیند} = \text{تعداد صفحات} \times \text{اندازه صفحه} = ۲^۴ \times ۲^۳ = ۲^۷ = ۱۲۸ \text{ بایت}$$

$$\text{تعداد بیت آفست} = \log_{۲}^{\text{اندازه صفحه}} = \log_{۲} ۸ = \log_{۲} ۲^۳ = ۳ \text{ بیت}$$

$$PT۱ + PT۲ = \log_{۲}^{\text{تعداد صفحات فرآیند}} = \log_{۲} ۲^{۴} = ۴$$

$$PT۲ = \log_{۲} r = \log_{۲} ۴ = \log_{۲} ۲^{۲} = ۲$$

توجه: فرمول PT2 خیلی ساده است، جدول صفحه حداکثر می‌تواند چهار سطر داشته باشد، چهار سمبل داریم، برای آدرس دهی چهار سمبل به چند بیت نیاز داریم، واضح است که ۲ بیت، که این همان مفهوم لگاریتم است.

$$PT۱ = (PT۱ \not{+} PT۲) - \not{PT۲} = ۲ \text{ بیت}$$

راه حل ساده‌تر (تجزیه): اندازه فرآیند را در اندازه r تجزیه کنید:

توجه: اندازه r، برابر تعداد سطرهای جداول صفحه است.

PT1 PT2

تعداد صفحات فرآیند $= ۲^{۴} = ۲^{(۲)} \times ۲^{(۲)}$

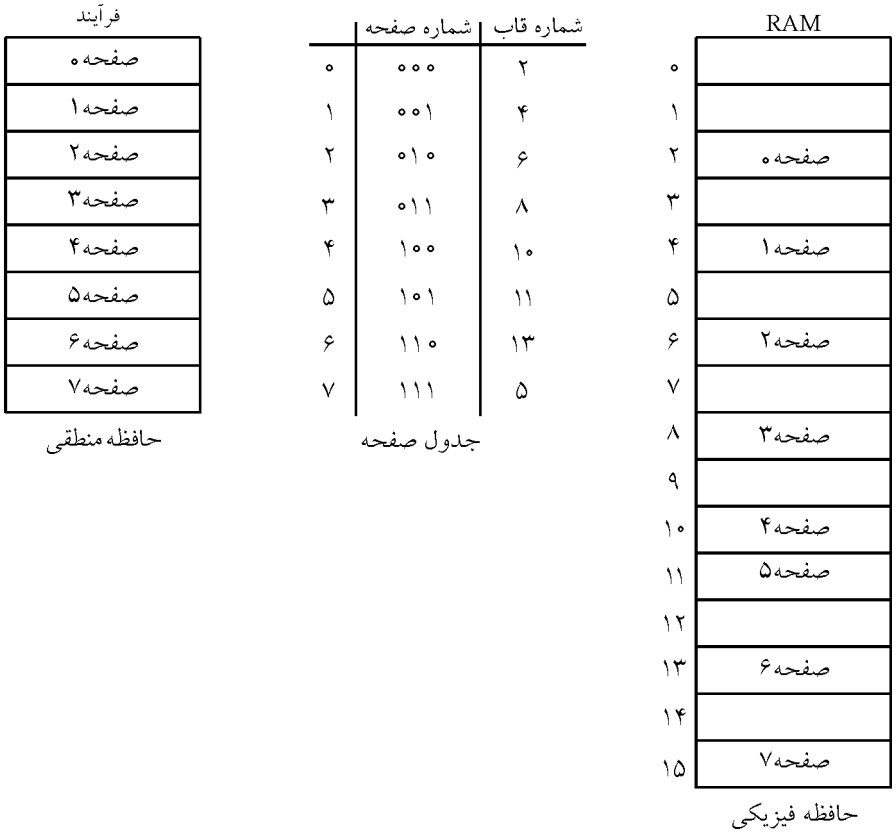
توجه: تجزیه را از چپ به راست و از اندیس n به ۱ شروع کنید و کمترین مقدار در اندیس ۱ قرار داده شود.

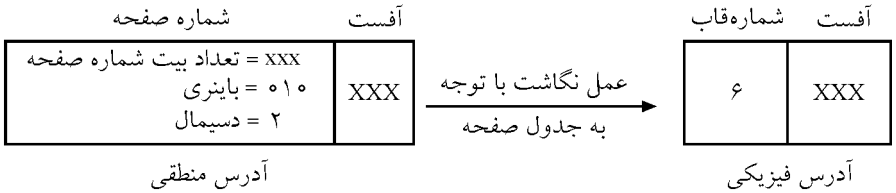
نکته مهم: توان کمتر همواره PT1 است. که در این مثال PT1 و PT2 برابر هستند.

نکته مهم: تعداد عملوندها در رابطهٔ بالا برابر تعداد سطوح جدول چند سطحی نیز می‌باشد، در رابطه فوق تعداد عملوندها برابر ۲ است. بنابراین تعداد سطوح جدول چند سطحی نیز برابر ۲ است، این نتیجه از آن جایی ناشی می‌شود که تجزیه فوق همان مفهوم لگاریتم را پیاده سازی می‌کند.

مثال: فرآیندی شامل ۸ صفحه می‌باشد، اگر اندازه جدول صفحه دارای محدودیت باشد و حداکثر هر جدول صفحه بتواند دارای ۲ سطر باشد، آنگاه جدول صفحه این فرآیند چند سطحی خواهد بود؟ (اندازه صفحات ۸ بایت است.)

پاسخ: واضح است که اگر اندازه جدول صفحه دارای محدودیت نباشد، این فرآیند دارای یک جدول صفحه تک سطحی با ۸ سطر خواهد بود. شکل زیر جدول صفحه تک سطحی این فرآیند را نشان می‌دهد:



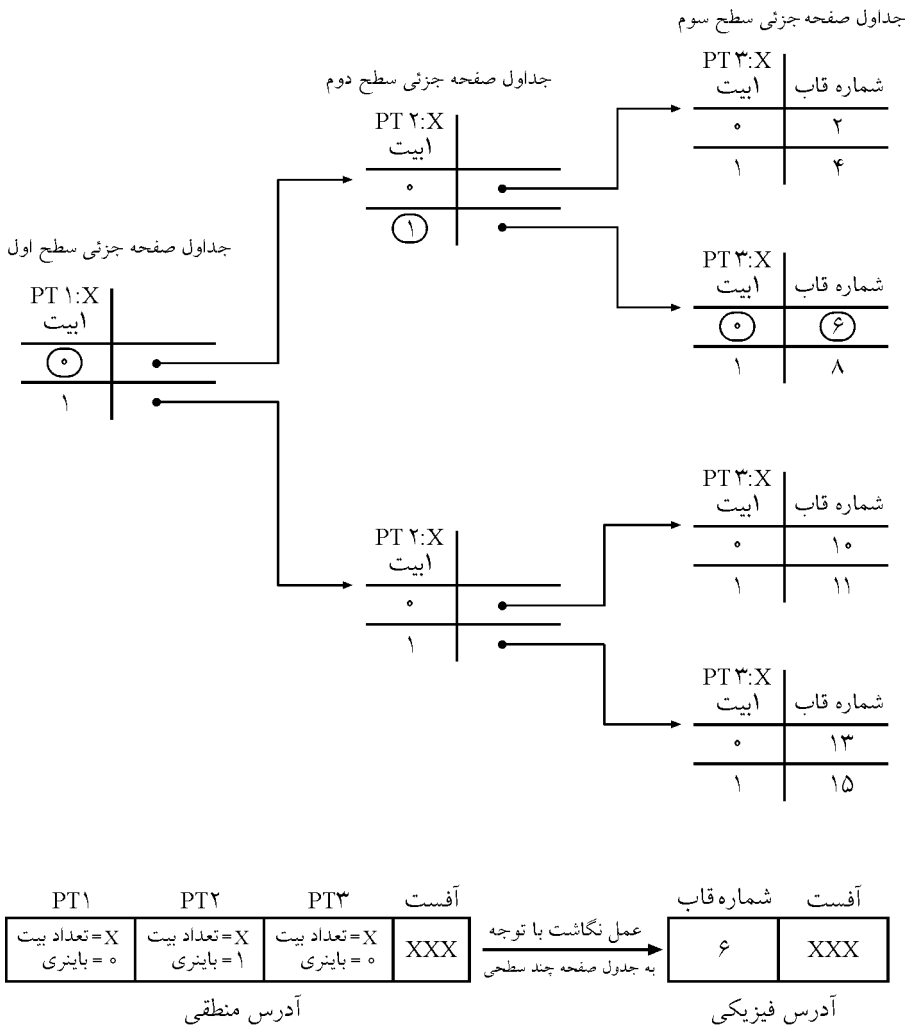


$$\text{بیت ۳} = \log_{۲}^{۲^{۳}} = \log_{۲}^{\text{تعداد صفحات فرآیند}} = \text{تعداد بیت شماره صفحه}$$

$$\text{بیت ۳} = \log_{۲}^{۲^{۳}} = \log_{۲}^{۸} = \log_{۲}^{\text{اندازه صفحه}} = \text{تعداد بیت آفست}$$

اما اگر اندازه جدول صفحه دارای محدودیت باشد و حداکثر بتواند دارای ۲ سطر باشد، واضح است که در این حالت باید از راه حل جدول صفحه چند سطحی استفاده شود. بنابراین با ۸ صفحه مواجه هستیم که هر ۲ صفحه آن می‌تواند داخل یک جدول صفحه جزئی قرار بگیرد.

شکل زیر واضح است که با توجه به شرایط مسأله یک جدول صفحه دو سطحی مسأله را حل می‌کند.



روابط زیر در جدول صفحه چند سطحی برقرار است:

f : تعداد صفحات فرآیند = ۸

r : تعداد سطرهای جدول صفحه جزئی = ۲

روش تقسیم متوالی:

$$\text{تعداد جداول صفحه جزئی در سطح سوم} = \frac{\text{تعداد صفحات فرآیند}}{r} = \frac{f}{r} = \frac{۸}{۲} = ۴$$

$$\text{تعداد جداول صفحه جزئی در سطح دوم} = \frac{\text{تعداد جداول در سطح سوم}}{r} = \frac{۴}{۲} = ۲$$

$$\text{تعداد جداول صفحه جزئی در سطح اول} = \frac{\text{تعداد جداول در سطح دوم}}{r} = \frac{۲}{۲} = ۱$$

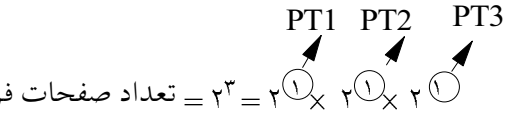
توجه: در اینجا تعداد تقسیم متوالی برابر ۳ است، بنابراین تعداد سطوح جدول چند سطحی برابر ۳ است.

روش لگاریتم

$$\text{تعداد سطوح جدول صفحه چند سطحی} = d = \left\lceil \log_{r}^{f} \right\rceil = \left\lceil \log_{۲}^{۸} \right\rceil = \left\lceil \log_{۲}^{۲^{۳}} \right\rceil = ۳$$

روش تجزیه

تعداد صفحات فرآیند باید در اندازه r، تجزیه گردد.



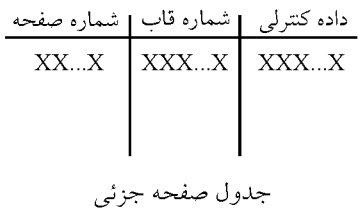
توجه: تعداد عملوندها برابر ۳ است، بنابراین تعداد سطوح جدول چند سطحی نیز برابر ۳ است.

مثال: سیستمی از آدرس‌های مجازی $۲^{۴۰}$ B پشتیبانی می‌کند. در این سیستم اندازه حافظه فیزیکی قابل دسترسی $۲^{۳۲}$ B و طول هر قاب (Frame) حافظه در این سیستم $۲^{۱۰}$ B می‌باشد. این سیستم از روش صفحه‌بندی (Paging) برای مدیریت حافظه استفاده کرده است. با فرض اینکه هر مدخل از جدول صفحه به bit ۱۰ بیت به عنوان بیت‌های کنترلی (بیت حضور و غیاب و ...) نیاز داشته باشد، در این صورت برای اینکه هر جدول صفحه جزئی دقیقاً در یک قاب قرار گیرد (الزامی برای پیوسته قرار گرفتن هر جدول صفحه در حافظه اصلی نباشد) باید حداقل، از جدول صفحه چند سطحی استفاده شود؟

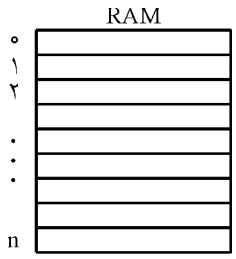
پاسخ: در این جا برای جدول صفحه جزئی محدودیتی به اندازه یک قاب داریم.

بنابراین اندازه جدول صفحه جزئی برابر اندازه قاب می‌باشد. بنابراین برای محاسبه تعداد سطرهای جدول صفحه جزئی، کافی است، اندازه قاب که برابر اندازه جدول صفحه جزئی است بر اندازه عرض جدول صفحه جزئی تقسیم گردد.

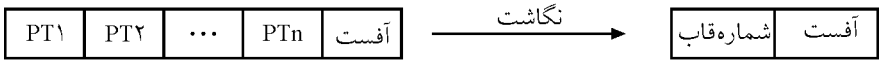
به شکل زیر توجه کنید:



جدول صفحه جزئی



حافظه فیزیکی



آدرس منطقی(مجازی) آدرس فیزیکی

توجه: عرض جدول صفحه همواره برابر حاصل جمع تعداد بیت‌های کنترلی و تعداد بیت‌های شماره قاب است، دقت کنید که تعداد بیت‌های شماره صفحه جزو عرض جدول صفحه نمی‌باشد، بلکه شماره صفحه، اندیس هر سطر جدول صفحه می‌باشد.

بنابراین داریم:

تعداد بیت‌های کنترلی + تعداد بیت‌های شماره قاب = عرض جدول صفحه جزئی

۱۰ + تعداد بیت‌های شماره قاب = عرض جدول صفحه جزئی

توجه: ۱۰ بیت کنترلی داریم، مطابق فرض صورت سؤال.

$$\text{تعداد بیت‌های شماره قاب} = b = \log_{۲}^{\text{تعداد قاب‌های حافظه فیزیکی}}$$

$$\text{تعداد قاب‌های حافظه فیزیکی} = \frac{\text{اندازه حافظه فیزیکی}}{\text{اندازه قاب}} = \frac{۲^{۳۲}B}{۲^{۱۰}B} = ۲^{۲۲}$$

بنابراین:

$$\text{تعداد بیت‌های شماره قاب} = b = \log_{۲}^{\text{تعداد قاب‌های حافظه فیزیکی}} = \log_{۲} ۲^{۲۲} = ۲۲$$

تعداد بیت‌های کنترلی + تعداد بیت‌های شماره قاب = عرض جدول صفحه جزئی

بایت ۴ یا بیت ۳۲ = ۱۰ + ۲۲ = عرض جدول صفحه جزئی

$$\text{تعداد سطرهای جدول صفحه جزئی} = \frac{\text{اندازه قاب}}{\text{عرض جدول صفحه}} = \frac{۲^{۱۰}B}{۲^{۲}B} = ۲^{۸} = ۲۵۶$$

$$\text{اندازه فرآیند (فضای آدرس مجازی)} = ۲^{۴۰}B$$

$B = ۲^{۱۰}$ = اندازه صفحه = اندازه قاب

f : تعداد صفحات فرآیند $= \frac{\text{اندازه فرآیند}}{\text{اندازه صفحه}} = \frac{۲^{۴۰}}{۲^{۱۰}} = ۲^{۳۰}$

r : تعداد سطرهای جدول صفحه جزئی $= ۲۵۶ = ۲^{۸}$

حال اطلاعات کافی برای محاسبه تعداد سطوح جدول صفحه چند سطحی، در اختیار داریم:

روش تجزیه:

تعداد صفحات فرآیند باید در اندازه r ($۲^{۸}$) تجزیه گردد.

PT1 PT2 PT3 PT4

تعداد صفحات فرآیند $= ۲^{۳۰} = ۲^{۶} \times ۲^{۸} \times ۲^{۸} \times ۲^{۸}$

توجه: تعداد عملوندها برابر ۴ است، بنابراین تعداد سطوح جدول چند سطحی نیز برابر ۴ است.

روش لگاریتم:

$$\text{تعداد سطوح جدول چند سطحی} = d = \left\lceil \log_{r} f \right\rceil = \left\lceil \log_{۲^{۸}} ۲^{۳۰} \right\rceil = ۴$$

روش تقسیم متوالی:

$$\text{تعداد جداول صفحه جزئی در سطح چهارم} = \frac{\text{تعداد صفحات فرآیند}}{r} = \frac{f}{r} = \frac{۲^{۳۰}}{۲^{۸}} = ۲^{۲۲}$$

$$\text{تعداد جداول صفحه جزئی در سطح سوم} = \frac{\text{تعداد جداول صفحه جزئی در سطح چهارم}}{r} = \frac{۲^{۲۲}}{۲^{۸}} = ۲^{۱۴}$$

$$\text{تعداد جداول صفحه جزئی در سطح دوم} = \frac{\text{تعداد جداول صفحه جزئی در سطح سوم}}{r} = \frac{۲^{۱۴}}{۲^{۸}} = ۲^{۶}$$

$$\text{تعداد جداول صفحه جزئی در سطح اول} = \frac{\text{تعداد جداول صفحه جزئی در سطح دوم}}{r} = \frac{۲^{۶}}{۲^{۸}} < ۱$$

توجه: سطح اول، یک جدول به حساب می‌آید، که $۲^{۶}$ سطر بیشتر نیاز ندارد.

توجه: تعداد تقسیم متوالی برابر ۴ است، بنابراین تعداد سطوح جدول صفحه چند سطحی برابر ۴

است.

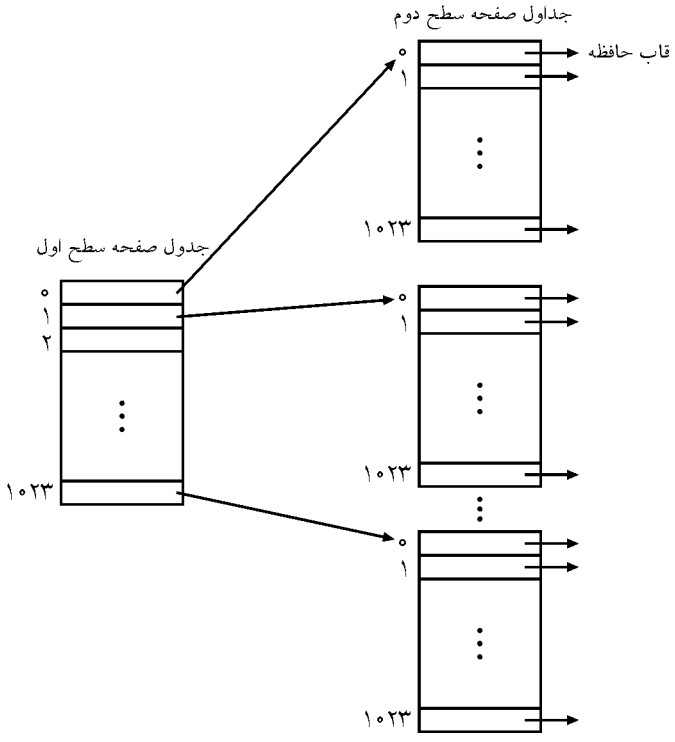
مثال: فرض کنید در یک سیستم آدرس‌های مجازی، ۳۲ بیتی هستند. در این کامپیوتر آدرس‌های مجازی به سه بخش تقسیم شده‌اند. دو فیلد ۱۰ بیتی به نام‌های PT1 و PT2 و یک آفست ۱۲ بیتی به صورت زیر:

۱۰ بیت	۱۰ بیت	۱۲ بیت
PT1	PT2	Offset

آدرس‌های مجازی

دقت کنید چون فیلد آفست ۱۲ بیتی است، اندازه صفحات 2^{12} می‌باشند و چون جمعاً ۲۰ بیت برای شماره صفحه داریم (۲ تا ۱۰ بیت) هر فرآیند می‌تواند 2^{20} صفحه داشته باشد.

در این مثال از ۲ سطح جدول صفحه استفاده می‌کنیم. در سطح اول فقط یک جدول داریم که ۱۰۲۴ داریه دارد. این جدول متناظر با فیلد ۱۰ بیتی PT1 می‌باشد. در سطح دوم ۱۰۲۴ جدول داریم که هر کدام ۱۰۲۴ درایه دارند. در شکل زیر این جداول دو سطحی مشاهده می‌شوند.



جداول صفحه دو سطحی

محاسبه تعداد سطوح بر اساس روش‌های مختلف به صورت زیر است:

f : تعداد صفحات فرآیند = $۲^{۲۰}$

r : تعداد سطرهای جدول صفحه جزئی = $۱۰۲۴ = ۲^{۱۰}$

محاسبه تعداد سطوح جدول چند سطحی به روش‌های مختلف به صورت زیر است:

روش تجزیه:

تعداد صفحات فرآیند باید در اندازه r ($۲^{۱۰}$) تجزیه گردد.

PT1 PT2

تعداد صفحات فرآیند = $۲^{۲۰} = ۲^{(۱۰)} \times ۲^{(۱۰)}$

توجه: تعداد عملوندها برابر ۲ است، بنابراین تعداد سطوح جدول صفحه چند سطحی نیز برابر ۲ است.

روش لگاریتم:

$$\text{تعداد سطوح جدول چند سطحی} = d = \left[\log \frac{f}{r}\right] = \left[\log \frac{۲^{۲۰}}{۲^{۱۰}}\right] = ۲$$

روش تقسیم متوالی:

$$\text{تعداد جداول صفحه جزئی در سطح دوم} = \frac{\text{تعداد صفحات فرآیند}}{r} = \frac{f}{r} = \frac{۲^{۲۰}}{۲^{۱۰}} = ۲^{۱۰} = ۱۰۲۴$$

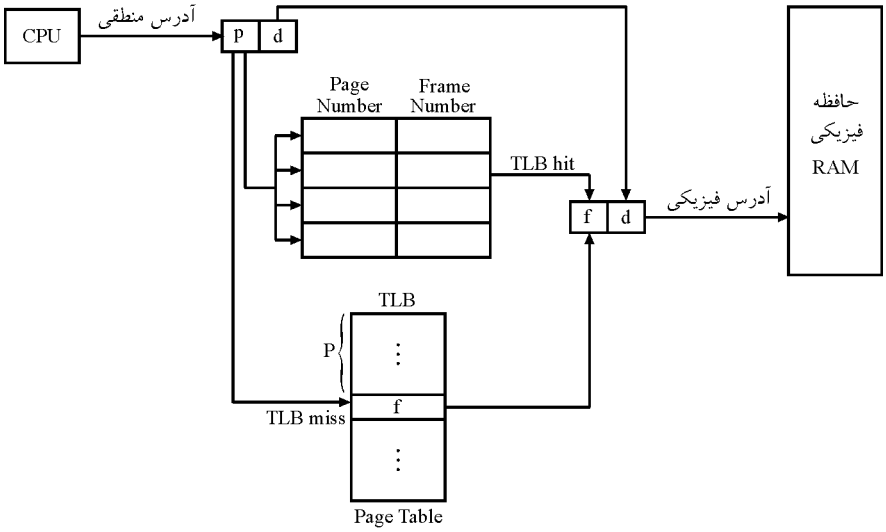
$$\text{تعداد جداول صفحه جزئی در سطح اول} = \frac{\text{تعداد جداول صفحه جزئی در سطح دوم}}{r} = \frac{۲^{۱۰}}{۲^{۱۰}} = ۱$$

توجه: تعداد تقسیم متوالی برابر ۲ است، بنابراین تعداد سطوح جدول صفحه چند سطحی برابر ۲ است.

تکنیک TLB(Translation Look-aside Buffer)

هنگامی که جداول صفحه در حافظه اصلی باشند برای دسترسی به یک خانه حافظه، باید دو یا چند بار به حافظه سر زد. در این صورت وقتی جداول به صورت چند سطحی پیاده سازی شوند، این تعداد بیشتر هم می‌شود و سرعت دسترسی به حافظه به شدت کاهش می‌یابد. در این حالت برای افزایش سرعت دسترسی به حافظه از تکنیک TLB استفاده می‌شود. TLB از حافظه‌های با سرعت بسیار بالا و

گران قیمت ساخته شده است که در واقع مجموعه‌ای از رجیسترها موسوم به Associative Register هستند. هر رجیستر دو بخش دارد: یکی کلید و دیگری مقدار. وقتی فرآیندی جهت اجرا انتخاب شود، **بخشی از درایه‌های صفحه** آن به TLB منتقل می‌شود. در این حالت وقتی CPU یک آدرس منطقی تولید می‌کند، شماره صفحه آن ابتدا در TLB جستجو می‌شود، اگر شماره صفحه در TLB یافت شد که آدرس قاب متناظر به دست می‌آید، اما اگر شماره صفحه در TLB نباشد آنگاه طبق روال قبل به سراغ جدول صفحه در حافظه می‌رویم. نکته مهم در مورد TLB این است که عمل جستجو در یک لحظه و همزمان در تمام سطرها صورت می‌گیرد. به همین دلیل سرعت دسترسی به آن بسیار بالاست.



نکته: با توجه به ساختار TLB و گران بودن آن، فقط بخش کوچکی از جدول صفحه این شانس را پیدا می‌کند که به TLB وارد شود. به این ترتیب اگر هنگام تبدیل آدرس، شماره صفحه در TLB موجود باشد اصطلاحاً HIT و در غیر این صورت MISS رخ داده است. با این اوصاف ضریب موفقیت یا نسبت اصابت به صورت زیر محاسبه می‌شود:

$$\text{HIT Ratio} = \frac{\text{HIT}}{\text{HIT} + \text{MISS}}$$

مثال: فرض کنید در یک سیستم، زمان دسترسی به حافظه ۶۰ نانوثانیه و زمان دسترسی به TLB، برابر ۵ نانوثانیه باشد. اگر احتمال وجود شماره صفحه در TLB برابر ۸۰٪ باشد، زمان دسترسی به حافظه چقدر است؟

حل: هنگامی که یک درخواست برای حافظه از راه می‌رسد، دو حالت ممکن است رخ دهد:
در حالت اول، شماره صفحه در TLB موجود است، که در این صورت یک بار به TLB و یک بار به حافظه رجوع می‌کنیم، یعنی جمعاً ۶۵ ns=۵+۶۰.

اما در حالت دوم، شماره صفحه در TLB پیدا نمی‌شود، در این صورت یک بار به TLB و دو بار به حافظه رجوع می‌کنیم (یک بار برای رجوع به جدول صفحه و پیدا کردن شماره صفحه و یک بار هم برای دستیابی به داده مورد نظر)، یعنی جمعاً ۱۲۵ ns=۵+۶۰+۶۰ . لذا زمان کل دسترسی مؤثر به حافظه با توجه به ضریب موفقیت برابر است با:

$$۰/۸۰ \times ۶۵ + ۰/۲۰ \times ۱۲۵ = ۷۷ \text{ ns}$$

نکته: هر فرآیند باید در مقابل تداخل‌های ناخواسته فرآیندهای دیگر محافظت شود (خواه این تداخل عمدی باشد، خواه غیرعمدی). بنابراین فرآیندها نباید قادر باشند بدون اجازه به محل‌های حافظه فرآیند دیگری مراجعه نمایند، به این مسئله **حفاظت** گویند.

نکته: معمولاً جهت نیل به حفاظت در تکنیک صفحه‌بندی، از چندین بیت کنار هر سطر در جدول صفحه استفاده می‌کنند. برای مثال می‌توان از یک بیت برای قابلیت نوشتن بر روی یک صفحه استفاده کرد و یا یک بیت می‌تواند قابل اجرا بودن محتویات یک صفحه را مشخص کند (و یا هر خاصیت موردنیاز دیگر). در این صورت انجام هر کار غیرمجازی می‌تواند به تولید یک وقفه از جانب سخت‌افزار برای سیستم عامل منجر شود.

جداول صفحه معکوس (Inverted Page Table)

در تکنیک صفحه‌بندی، برای هر فرآیند یک جدول صفحه 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 مقرون به صرفه نیست.

برای حل این مشکل از جداول صفحه معکوس استفاده می‌کنیم. در این حالت 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.

نکته: با استفاده از تکنیک جداول صفحه معکوس، مقدار زیادی در حافظه صرفه‌جویی می‌شود اما تبدیل آدرس مجازی به فیزیکی سخت‌تر و زمان‌گیر می‌شود. در واقع وقتی فرایند p برای صفحه n مراجعه می‌کند باید تمام جدول صفحه معکوس را برای یافتن درایه (p,n) جستجو کرد تا بتوان آدرس قاب مربوط به این صفحه را یافت. البته برای افزایش سرعت این روش می‌توان از ایده TLB نیز بهره

برد.

نکته: در حالت کلی تفاوت‌های قطعه‌بندی و صفحه‌بندی عبارتند از:

۱ـ در قطعه‌بندی، تکه تکه شدن داخلی نداریم اما در صفحه‌بندی تکه تکه شدن داخلی رخ می‌دهد.

۲ـ در قطعه‌بندی تکه تکه شدن خارجی رخ می‌دهد اما در صفحه‌بندی تکه تکه شدن خارجی نداریم.

۳ـ قطعه‌بندی توسط برنامه‌نویس صورت می‌گیرد اما برنامه‌نویس از صفحه‌بندی کاملاً بی‌خبر است.

۴ـ در قطعه‌بندی ملاک تقسیم‌بندی، ارتباط منطقی اطلاعات با هم است اما در صفحه‌بندی، فقط اندازه صفحات ملاک تقسیم‌بندی می‌باشد.

۵ـ در قطعه‌بندی از آنجا که قطعات به صورت منطقی تقسیم‌بندی شده‌اند، امکان به اشتراک گذاشتن قطعات به خوبی وجود دارد اما در صفحه‌بندی چون تقسیم‌بندی فقط براساس اندازه و توسط سیستم عامل صورت می‌گیرد، عموماً نمی‌توان صفحات را به اشتراک گذاشت.

۶ـ در صفحه‌بندی اندازه صفحات با هم برابرند اما در قطعه‌بندی اندازه قطعات لزوماً با هم برابر نیستند.

۷ـ در قطعه‌بندی، نحوه انتخاب محل قطعه در حافظه، بر روی کارایی تأثیر مستقیم دارد (رجوع شود به تفاوت روش‌های Best Fit و First Fit و Worst Fit و ...) اما در صفحه‌بندی، یک صفحه در هر قابی که قرار بگیرد تأثیری بر روی کارایی ندارد.

۸ـ در حالت کلی هدف قطعه‌بندی، کاهش تکه تکه شدن خارجی و به اشتراک گذاشتن قطعات است. اما هدف اصلی صفحه‌بندی، امکان استفاده از حافظه مجازی می‌باشد (رجوع شود به مبحث حافظه مجازی).

۹ـ در صفحه‌بندی، سیستم عامل یک جدول صفحه برای هر فرآیند نگهداری می‌کند تا نشان دهد هر صفحه در کدام قاب حافظه قرار گرفته است، اما در قطعه‌بندی، سیستم عامل یک جدول قطعه برای هر فرآیند نگهداری می‌کند که آدرس بار شدن و طول هر قطعه در حافظه را نشان می‌دهد.

قطعه‌بندی همراه با صفحه‌بندی

از یک دیدگاه قطعه‌بندی دو نقص عمده دارد: اگر اندازه قطعات قدری بزرگ باشد، تکه تکه شدن خارجی قابل اغماض نیست. همچنین ممکن است زمان جستجو در حافظه برای یافتن مکان مناسب برای یک قطعه طولانی شود.

برای حل این مشکل قطعات را صفحه‌بندی می‌کنیم!!! در واقع قطعات تولید شده توسط برنامه‌نویس یا کامپایلر، توسط سیستم عامل صفحه‌بندی می‌شوند. در این حالت به ازای هر فرآیند یک جدول قطعه و به ازای هر قطعه، یک جدول صفحه ایجاد می‌شود که تعداد درایه‌های جدول قطعه، بستگی به تعداد

قطعات و تعداد درایه‌های جدول صفحه، بستگی به اندازه قطعه دارد.

نکته: واضح است در این روش نیز آخرین صفحه هر قطعه پر نمی‌شود، بنابراین به طور میانگین **به ازای هر قطعه نصف صفحه** تکه تکه شدن داخلی داریم.

نکته: در این سیستم در واقع آدرس‌های منطقی ساختاری مانند شکل زیر دارند:

انحراف در صفحه	شماره صفحه	شماره قطعه

نکته: در حالت کلی می‌توان مبحث تکه تکه شدن داخلی و خارجی را در جدول زیر خلاصه کرد:

تکه تکه شدن در روش‌های تخصیص حافظه

روش	تکه تکه شدن داخلی	تکه تکه شدن خارجی
بخش بندی ایستا	دارد	ندارد
بخش‌بندی پویا	ندارد	دارد
قطعه‌بندی	ندارد	دارد
صفحه‌بندی	دارد (نیم صفحه به ازای هر فرآیند)	ندارد
قطعه‌بندی همراه با صفحه‌بندی	دارد (نیم‌صفحه به ازای هر قطعه)	ندارد

تکنیک مبادله (Swapping)

فرض کنید یک سیستم چندبرنامگی داریم و در آنِ واحد چندین فرآیند در این سیستم موجودند که مطمئناً یکی از آنها در حالت اجرا، تعدادی در حالت آماده و تعدادی در حالت انتظار (مسدود) قرار دارند. تمام حافظه به این فرآیندها اختصاص یافته و هیچ جای خالی در حافظه نداریم (در این مثال روش مدیریت حافظه موردنظر نمی‌باشد). حال فرض کنید در این لحظه به مقداری حافظه بیشتر نیاز پیدا کنیم، برای مثال برنامه در حال اجرا قصد دارد چیزی را در حافظه لود کند یا برنامه‌ای با اولویت بالا از راه رسیده است. در این حالت یکی از برنامه‌ها باید **به طورکامل** به دیسک منتقل شود (که انتخاب این فرآیند وظیفه زمانبند میان مدت است). به انتقال یک فرآیند **به طورکامل** از حافظه به دیسک، مبادله (Swqpping) گویند.

دقت کنید Swapping با حافظه مجازی تفاوت دارد. در حافظه مجازی فقط بخشی از فرآیند به دیسک منتقل می‌شود اما در این حالت کل فرآیند به دیسک انتقال می‌یابد. (حافظه مجازی در فصل بعد مورد بررسی قرار می‌گیرد).

تست‌های فصل چهارم: مدیریت حافظه اصلی

۱- آدرس منطقی 0001010010111010 را در نظر بگیرید. با مدیریت صفحه‌بندی ۲۵۶ صفحه‌ای برای یک حافظه با 256 قاب (frame) و استفاده از جدول صفحه‌ای که در آن هر شماره قاب $\frac{1}{4}$ شماره صفحه باشد، کدام گزینه در مورد مدیریت این حافظه و آدرس فیزیکی متناظر با آدرس منطقی فوق صحیح است؟ (مهندسی کامپیوتر – دولتی۸۹)

۱) اگرچه این روش نگاشت صفحه مشکل دارد ولی آدرس فیزیکی متناظر 0000101010111010 است.

۲) اگرچه این روش نگاشت صفحه مشکل دارد ولی آدرس فیزیکی متناظر 0000010110111010 است.

۳) این روش نگاشت صفحه بدون مشکل کار می‌کند و آدرس فیزیکی متناظر 0000010110111010 است.

۴) این روش نگاشت صفحه بدون مشکل کار می‌کند و آدرس فیزیکی متناظر 0000101010111010 است.

۲- مدیریت حافظه در یک سیستم فرضی به صورت قطعه‌بندی صفحه‌بندی شده (paged segmentation) است. و اندازه هر صفحه 4 کیلو بایت است. هر درایه (entry) جدول قطعه دارای 3 بایت و به صورت زیر است: (مهندسی IT- دولتی۸۹)

بایت سوم	بایت دوم بایت اول
LIMIT	PTBA

و هر درایه جدول صفحه یک بایتی است و نشان‌دهنده شماره قاب (frame) است. در PCB یک فرآیند برای آدرس پایه جدول قطعه (STBA) مقدار 0AFEH دیده می‌شود. اگر در این فرآیند آدرس منطقی [3456H , 02H] تولید شود، آدرس فیزیکی نظیر چه خواهد بود. بخش اول آدرس منطقی شماره قطعه است. حرف H به معنی Hex است. محتویات حافظه به صورت زیر است:

0B00H	08H	0B08H	08H
0B01H	09H	0B09H	09H
0B02H	03H	0B0AH	00H
0B03H	0AH	0B0BH	0BH
0B04H	0BH	0B0CH	0AH
0B05H	09H	0B0DH	0CH
0B06H	05H	0B0EH	04H
0B07H	0BH	0B0FH	05H

۱) 08345H

۲) 0A456H

۳) 0B560H

۴) 08456H

۳- حافظه اصلی با وضعیت نشان داده شده در شکل را در نظر بگیرید. اگر مدیریت حافظه اصلی براساس اختصاص‌دهی پویا باشد و اختصاص‌دهی فضای خالی به فرآیندها براساس Next-fit انجام گیرد و فرآیندهای $P_0, P_1, ..., P_5$ جهت اجرا شدن، مطابق با اطلاعات جدول زیر وارد سیستم شوند، با فرض اینکه از بین فضاهای پر شماره 1 تا 4، فقط فضای پر 3 در لحظه t+20 آزاد گردد (دیگر فضاهای پر تا اتمام اجرای فرآیندهای فوق آزاد نمی‌گردند)، متوسط زمان بازگشت (turn around) و متوسط زمان انتظار فرآیندهای فوق در روش FCFS به ترتیب چقدر است؟ (مهندسی کامپیوتر – دولتی۹۰)

فرآیند	زمان ورود	حافظه مورد نیاز (KB)	زمان سرویس
P_0	t	25	30
P_1	t+1	20	40
P_2	t+2	40	20
P_3	t+3	25	45
P_4	t+4	10	35
P_5	t+5	35	15

فضای پر 1 (30KB)
فضای خالی 1 (20KB)
فضای پر 2 (10KB)
فضای خالی 2 (40KB)
فضای پر 3 (20KB)
فضای خالی 3 (30KB)
فضای پر 4 (40KB)
فضای خالی 4 (30KB)

۱) 110.83 و 80 ۲) 109.5 و 85.83 ۳) 109.5 و 78.67 ۴) 116.67 و 85.83

۴- فرض کنید سیستمی از فضای آدرس‌دهی مجازی $2^{40}B$ پشتیبانی می‌کند. در این سیستم اندازه حافظه فیزیکی قابل دسترسی $2^{32}B$ و طول هر قاب (Frame) حافظه در این سیستم $2^{10}B$ می‌باشد. این سیستم از روش صفحه‌بندی (Paging) برای مدیریت حافظه استفاده کرده است. با فرض اینکه هر مدخل از جدول صفحه به 10bit به عنوان بیت‌های کنترلی (بیت حضور - غیاب و ...) نیاز داشته باشد، در این صورت برای اینکه هر جدول صفحه جزئی دقیقاً در یک قاب قرار گیرد (الزامی برای پیوسته قرار گرفتن هر جدول صفحه در حافظه اصلی نباشد) باید حداقل از جدول صفحه چند سطحی استفاده شود؟ (مهندسی IT- دولتی۹۰)

۱) سه ۲) دو ۳) چهار ۴) پنج

۵- کدام گزینه در مورد جدول صفحه وارونه درست نیست؟ (مهندسی IT- دولتی۹۳)

۱) به ازاء هر صفحه فیزیکی حافظه در جدول صفحه وارونه تنها یک خانه دارد.

۲) در جدول صفحه وارونه هر پردازه یک جدول صفحه دیگر برای خود نگهداری می‌نماید.

۳) در جدول صفحه وارونه، تنها کل اطلاعات فضای آدرس منطقی برای یک پردازه در دسترس است.

۴) هر خانه از جدول صفحه وارونه دست کم دارای آدرس منطقی صفحه و شماره پردازه استفاده‌کننده از این صفحه است.

۶- کدامیک از عبارات زیر صحیح است؟ (مهندسی کامپیوتر-دولتی۹۴)

۱) اندازه آدرس‌های مجازی و فیزیکی با هم برابر است.

۲) dispatcher اولویت ریسه‌ها (thread) را تعیین و مقدار دهی می‌نماید.

۳) اگر یک ریسه (thread)، CPU-Bound باشد باید اولویت آن برای دسترسی به I/O بالاتر از ریسه‌های I/O Bound باشد.

۴) اگر در یک سیستم مدیریت حافظه garbage collection استفاده شود این سیستم دچار fragmentation نمی‌شود.

۷- فرض کنید اندازه هر صفحه 1KB است و هر مدخل صفحه 4 بایت فضا می‌گیرد. برای اینکه بتوانیم یک آدرس 34 بیتی را نگاشت کنیم به طوری که هر جدول صفحه تنها در یک صفحه ذخیره شود. چند سطح جدول صفحه نیاز است؟ فرض کنید که هر جدول به اندازه یک صفحه است. (مهندسی IT – دولتی ۹۴)

۱) 4 ۲) 3 ۳) 2 ۴) 1

۸- در یک سیستم حافظه مجازی از نوع قطعه – صفحه‌ای، بخشی از جدول TLB به صورت زیر است. اگر تعداد کلمات هر صفحه 4096 باشد، حجم حافظه مجازی چند برابر حافظه اصلی است؟ (مهندسی کامپیوتر– دولتی ۹۵)

← 4 →	← 12 →	← 12 →
قطعه	صفحه	بلوک
1	2FF	012
5	02A	2A5

۱) 16
۲) 8
۳) 4
۴) 2

۹- سیستمی علاوه بر ذخیره جدول صفحه در حافظه اصلی، از جدول TLB نیز با نرخ miss برابر 20% استفاده می‌کند. اگر خواندن از حافظه اصلی 100ms زمان بردارد و درصد کارایی سیستم در صورت استفاده نکردن از جدول TLB برابر با 80% باشد، خواندن از TLB چند نانوثانیه زمان لازم دارد؟ (مهندسی کامپیوتر– دولتی ۹۵)

۱) 20 ۲) 40 ۳) 50 ۴) 60

۱۰- در یک ساختار حافظه، سیستم صفحه‌بندی سه سطحی از آدرس مجازی مطابق فرمت زیر استفاده می‌شود. اگر نرخ برخورد هر سطح 90% باشد، احتمال برخورد تبدیل یک آدرس مجازی به فیزیکی چقدر است؟ فرض شود که احتمال برخورد هر سطح مستقل از احتمال برخورد در سطح دیگر است. (مهندسی IT – دولتی ۹۵)

P3	P2	P1	D

۱) 0.001 ۲) 0.1 ۳) 0.729 ۴) 0.9

۱۱- فرض کنید که اندازه فضای آدرس منطقی، 8 صفحه 1024 بایتی و اندازه حافظه فیزیکی 32 قاب است. طول آدرس منطقی و آدرس فیزیکی (به ترتیب از راست به چپ) هر کدام چند بیت می‌باشد؟ (مهندسی IT – دولتی ۹۵)

۱) 11,9 ۲) 13,11 ۳) 15,13 ۴) 13,15

۱۲- کدام گزینه درباره نرخ برخورد (hit ratio) در TLB درست نیست؟ (مهندسی IT – دولتی ۹۵)

۱) اگر تعداد خانه‌های TLB بیشتر شود نرخ برخورد بیشتر می‌شود.

۲) اگر اندازه صفحه‌ها بزرگتر شود نرخ برخورد بیشتر می‌شود.

۳) برای صفحه‌های با اندازه ثابت، اگر طول آدرس فیزیکی بیشتر شود نرخ برخورد تغییر نمی‌کند.

۴) برای صفحه‌های با اندازه ثابت، با افزایش اندازه قطعه (segment)، نرخ برخورد بیشتر می‌شود.

۱۳- یک کامپیوتری با 8 گیگابایت حافظه را در نظر بگیرید که اندازه هر صفحه 8 کیلوبایت و هر خانه از جدول 4 بایت باشد. در صورتیکه این کامپیوتر از جدول چند سطحی استفاده نماید که هر جدول صفحه در یک صفحه ذخیره شود و بخواهیم آدرس مجازی 46 بیتی را به آدرس فیزیکی تبدیل نماییم. برای خواندن یک کلمه 32 بیتی به چند دسترسی به حافظه نیاز است؟

(مهندسی کامپیوتر – دولتی ۹۶)

۱) 4 ۲) 3 ۳) 2 ۴) 1

۱۴- 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 وارونه (Inverted Page Table) با 128 مگابایت حافظه، جدول صفحه دارای چند خانه است؟

(مهندسی IT – دولتی ۹۶)

۱) 2^{16} ۲) 2^{17} ۳) 2^{23} ۴) 2^{34}

۱۵- سه سیستم مدیریت حافظه S_1، S_2 و S_3 را در نظر بگیرید. S_1 دارای یک TLB با زمان پاسخ 120ns است که نرخ برخورد آن 60% می‌باشد. S_2 دارای یک TLB بزرگ‌تر ولی کندتر می‌باشد که زمان پاسخ آن 150ns و نرخ برخورد آن 80% می‌باشد. S_3 دارای TLB با زمان پاسخ 120ns و نرخ برخورد 70% است. دو سیستم S_1 و S_2 همزمان با ارسال درخواست به TLB، آن را به حافظهٔ اصلی نیز ارسال می‌کنند تا در صورت عدم یافتن آدرس در TLB، زمان پاسخ کاهش یابد. سیستم S_3 پس از دریافت پاسخ از TLB و عدم یافتن آدرس، درخواست را به حافظهٔ اصلی

ارسال می‌کند. کمترین و بیشترین میانگین زمان پاسخ این سه سیستم به ترتیب از راست به چپ چند نانوثانیه است؟ **(مهندسی IT – دولتی ۹۶)**

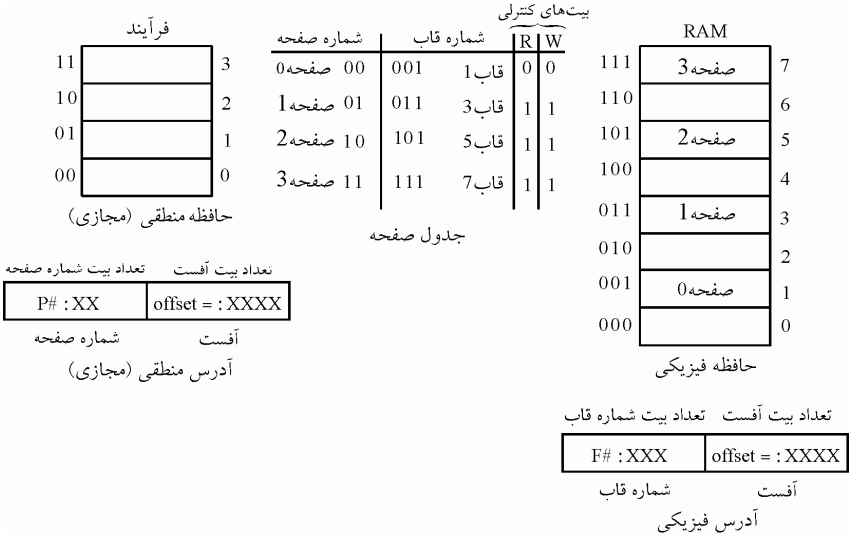
۱) 312-300 ۲) 312-264 ۳) 300-312 ۴) 300-264

پاسخ تست‌های فصل چهارم: مدیریت حافظه اصلی

۱- گزینه (۲) صحیح است.

بطور کلی روابط میان آدرس منطقی و آدرس فیزیکی در راه حل صفحه بندی به صورت زیر است:

توجه: فرض کنید اندازه هر صفحه 16 بایت، اندازه هر فرآیند 64 بایت و اندازه حافظه فیزیکی برابر 128 بایت باشد.



$$\text{تعداد صفحات فرآیند} = \frac{\text{اندازه فرآیند}}{\text{اندازه صفحه یا اندازه قاب}} = \frac{64}{16} = 4$$

$$\text{تعداد قاب‌های حافظه‌ی فیزیکی} = \frac{\text{اندازه حافظه فیزیکی}}{\text{اندازه صفحه یا اندازه قاب}} = \frac{128}{16} = 8$$

$$\text{تعداد بیت شماره صفحه} = b = \log_2^{\text{تعداد صفحات فرآیند}} = \log_2^4 = 2bit$$

$$\text{تعداد بیت شماره قاب} = b = \log_2^{\text{تعداد قاب‌های حافظه فیزیکی}} = \log_2^8 = 3bit$$

$$\text{تعداد بیت آفست} = b = \log_2^{\text{اندازه صفحه یا اندازه قاب}} = \log_2^{16} = 4bit$$

عرض جدول صفحه × تعدادصفحات فرآیند (تعداد درایه‌های جدول صفحه) = اندازه جدول صفحه

توجه: عرض جدول صفحه برابر حاصل جمع تعداد بیت‌های کنترلی و تعداد بیت‌های شماره قاب می‌باشد. دقت کنید که تعداد بیت‌های شماره صفحه جزو عرض جدول صفحه نمی‌باشد، بلکه شماره صفحه، اندیس هر سطح جدول صفحه می‌باشد.

در جدول فوق 2 بیت مربوط به بیت‌های کنترلی و 3 بیت مربوط به تعداد بیت‌های شماره قاب می‌باشد.

مثال: عرض جدول صفحه فوق برابر $3bit + 2bit = 5bit$ می‌باشد، بنابراین داریم:

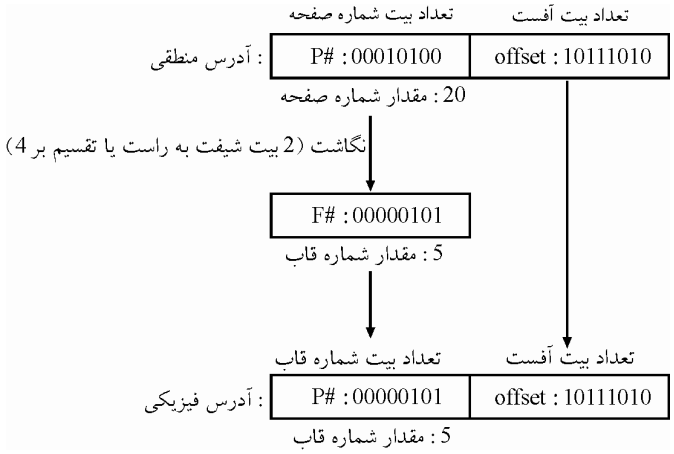
اندازه جدول صفحه $= 4 \times 5bit = 20bit$

حال برگردید به مسئله مطرح شده در صورت سوال .....

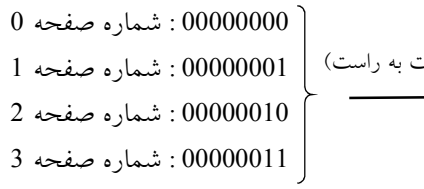
توجه: در صورت سوال مطرح شده، برای تبدیل شماره صفحه به شماره قاب یک الگوی ثابت و از قبل تعیین شده، مشخص گردیده است که این فرض نادرست می‌باشد. تعیین شماره قاب در حافظه فیزیکی برای یک صفحه از یک فرآیند براساس قاب‌های آزاد در حافظه فیزیکی و توسط سیستم عامل انجام می‌گردد و نه براساس یک الگوی از قبل مشخص شده و نگاشت ثابت، زیرا یک صفحه از فرآیند همواره به یک قاب خاص از حافظه فیزیکی نگاشت نمی‌شود.

از آنجا که فرآیند موجود در این سیستم شامل 256 صفحه است، بنابراین 8 بیت سمت چپ $(b = \log_2^{256} = 8bit)$ از آدرس منطقی 16 بیتی مربوط به شماره صفحه و 8 بیت سمت راست باقیمانده مربوط به آفست است. همچنین از آنجا که حافظه فیزیکی موجود در این سیستم نیز شامل 256 قاب است، بنابراین 8 بیت سمت چپ $(b = \log_2^{256} = 8bit)$ از آدرس فیزیکی نیز مربوط به تعداد بیت‌های شماره قاب می‌باشد. با توجه به اینکه در ترجمه آدرس منطقی به فیزیکی در سیستم‌های صفحه‌بندی شده، تعداد بیت‌های آفست ثابت مانده و فقط شماره صفحه با شماره قاب جای‌گزین می‌گردد، بنابراین آدرس فیزیکی نیز دارای طولی برابر 16 بیت خواهد بود.

باتوجه به فرض مطرح شده در صورت سؤال که شماره قاب $\frac{1}{4}$ شماره صفحه است (که فرض نادرستی است) اما برای این کار، کافی است شماره صفحه را بر 4 تقسیم کنیم، یا مقدار دودویی شماره صفحه را دوبار به سمت راست شیفت دهیم تا مقدار شماره قاب بدست بیاید و سپس 8 بیت سمت راست آدرس منطقی را به عنوان آفست به آن بیافزاییم:



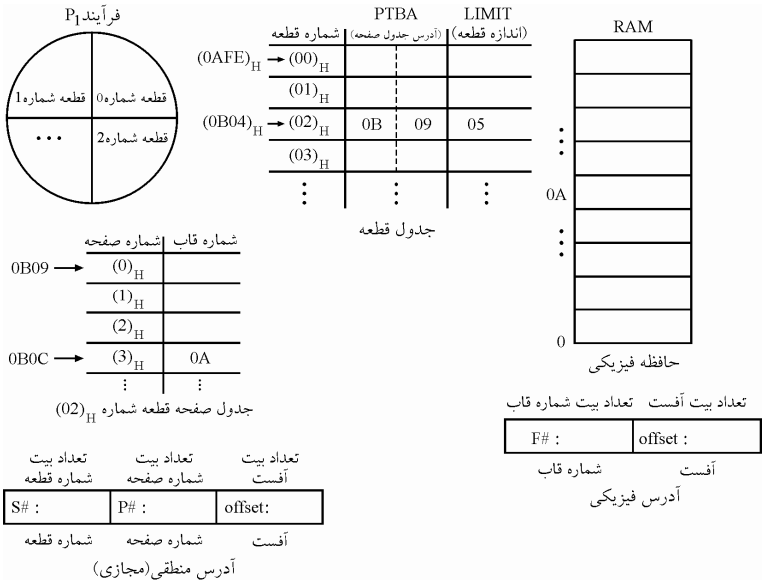
ولی این روش نگاشت نادرست است، چرا که همیشه مجموع صفحاتی که در 2 بیت سمت راست شماره صفحه و به شکل 00، 01، 10، 11 متفاوت ولی در 6 بیت باقی‌مانده یکسان هستند به دلیل انتقال 2 بیت به سمت راست، همگی یک شماره قاب واحد خواهند داشت. یعنی به هر قاب موجود در حافظه فیزیکی، 4 صفحه از آدرس منطقی نسبت داده می‌شود. مثلاً صفحات شماره 0، 1، 2 و 3 همگی به شماره قاب 0 نگاشت می‌شوند و نمی‌توانند همزمان با هم درون حافظه فیزیکی قرار داده شوند.



۲- گزینه (۲) صحیح است.

در راه‌حل قطعه‌بندی همراه با صفحه‌بندی، قطعات تولید شده توسط برنامه‌نویس یا کامپایلر، توسط سیستم عامل صفحه‌بندی می‌شوند. در این حالت به ازای هر فرآیند یک جدول قطعه و به ازای هر قطعه، یک جدول صفحه ایجاد می‌شود که تعداد درایه‌های جدول قطعه، بستگی به اندازه فرآیند و تعداد قطعات حاصل و تعداد درایه‌های جدول صفحه، بستگی به اندازه قطعه و تعداد صفحات حاصل دارد.

به طور کلی روابط میان آدرس منطقی و آدرس فیزیکی در راه‌حل قطعه‌بندی همراه با صفحه‌بندی، به صورت زیر است:



مطابق صورت سوال، آدرس منطقی به صورت [02H,3456H] بیان شده است. در ابتدا از این آدرس منطقی مطرح شده می‌توان متوجه شد که از میان قطعات موجود در یک فرآیند، قطعه شماره $(02)_H$ مورد درخواست است. از آنجا که هر قطعه صفحه‌بندی شده است، پس بخش دوم آدرس منطقی، یعنی (3456H) از #P و offset تشکیل شده‌است، با توجه به اینکه صفحات 4 کیلوبایتی ($4KB = 2^2 \times 2^{10} = 2^{12}$) هستند، پس 12 بیت کم ارزش آن (سه رقم سمت راست هگزا دسیمال) که برابر $(456)_H$ است، بیانگر آفست و $(3)_H$ نشان‌دهنده‌ی شماره صفحه مورد درخواست خواهد بود.

$$\text{تعداد بیت آفست} = b = \log_{2^{\text{اندازه صفحه یا اندازه قاب}}} = \log_2^{2^{12}} = 12bit$$

توجه: 12 بیت در مبنای باینری برابر 3 رقم در مبنای هگزا دسیمال است. هر رقم هگزا دسیمال برابر با 4 رقم باینری است.

توجه: دقت کنید که همه‌ی اعداد به جای دسیمال، به شکل هگزا دسیمال بیان شده است.

با توجه به مطالب بیان شده، آدرس منطقی به صورت زیر خواهد بود:

	تعداد بیت شماره قطعه	تعداد بیت شماره صفحه	تعداد بیت آفست
آدرس منطقی	S# : $(02)_H$	P# : $(3)_H$	offset: $(456)_H$
	شماره قطعه	شماره صفحه	آفست

توجه: از آنجا که تعداد قطعات فرآیند مشخص نشده است، بنابراین، تعداد بیت شماره قطعه قابل تشخیص نیست.

توجه: از آنجا که تعداد صفحات یک قطعه مشخص نشده است، بنابراین، تعداد بیت شماره صفحه قابل تشخیص نیست.

توجه: در این سوال، داشتن اطلاعات مربوط به دو مورد فوق ضروری نیست.

چون در آدرس منطقی داده شده، قطعه شماره $(02)_H$ مورد درخواست است، پس باید از ابتدای جدول قطعه سطر به سطر حرکت کنیم تا به قطعه شماره $(02)_H$ برسیم. مطابق مفروضات مسأله، هر سطر جدول قطعه 3 بایت در نظر گرفته شده است، در صورت سوال، آدرس ابتدای جدول قطعه به صورت $(0AFE)_H$ بیان شده است، مطابق جدول قطعه واضح است که برای کشف آدرس سطر حاوی قطعه شماره‌ی $(02)_H$ باید، از دو سطر جدول قطعه گذر شود، یعنی $2 \times 3 = 6$ بایت از ابتدای جدول قطعه به جلو برویم، بنابراین آدرس سطر حاوی قطعه مورد نظر برابر است با:

$$\begin{array}{r} 0AFE \\ +\quad 6 \\ \hline (0B04)_H \end{array}$$

به آدرس به دست آمده رفته و 3 بایت را خارج می‌کنیم که باتوجه به الگوی داده شده در صورت سؤال، داریم:

بایت اول	بایت دوم	بایت سوم
PTBA		LIMIT
0B	09	05

آدرس جدول صفحه — اندازه قطعه

بنابراین آدرس شروع جدول صفحه برای قطعه شماره (02) برابر $(0B09)_H$ می‌باشد. چون در آدرس منطقی داده شده، صفحه شماره $(3)_H$ مورد درخواست است، پس باید از ابتدای جدول صفحه سطر به سطر حرکت کنیم تا به صفحه شماره $(3)_H$ برسیم، مطابق مفروضات مسأله، هر سطر جدول صفحه 1 بایت در نظر گرفته شده است. مطابق جدول صفحه واضح است که برای کشف آدرس سطر حاوی صفحه شماره $(3)_H$ باید، از سه سطر جدول صفحه گذر شود، یعنی $3 \times 1 = 3$ بایت از ابتدای جدول صفحه به جلو برویم، بنابراین آدرس سطر حاوی صفحه مورد نظر برابر است با:

$$\begin{array}{r} 0B09 \\ +\quad 3 \\ \hline (0B0C)_H \end{array}$$

حال با کشف آدرس فوق، به سطر مورد نظر در جدول می‌رویم، محتوای آدرس $(0B0C)_H$ برابر مقدار $(0A)_H$ می‌باشد، که برابر شماره قاب مورد نظر برای صفحه مورد درخواست است. بنابراین آدرس فیزیکی به صورت زیر خواهد بود:

F#: $(0A)_H$	offset: $(456)_H$

شماره قاب — آفست

آدرس فیزیکی

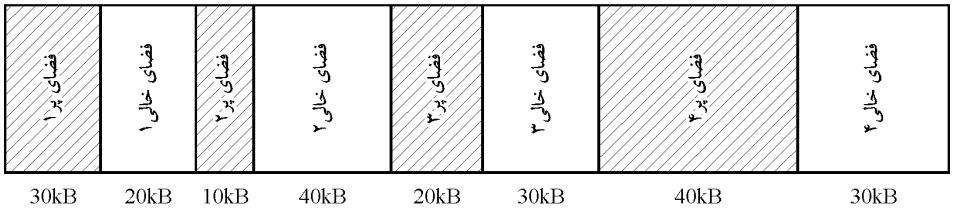
آدرس فیزیکی $= (0A456)_H$

۳- گزینه (۴) صحیح است.

یک فرآیند پس از ورود به حافظه و قرارگیری در صف آماده پردازنده، بر اساس الگوریتم زمان‌بندی پردازنده (زمان‌بندی کوتاه مدت) می‌تواند پردازنده را دریافت کند و اجرا گردد. در این سوال الگوریتم قرارگیری فرآیندها در حافظه Next-Fit در نظر گرفته شده است.

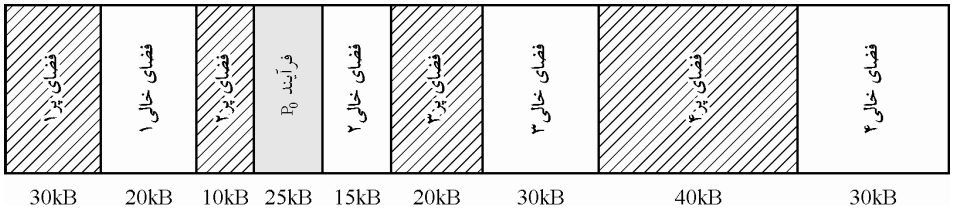
الگوریتم Next-Fit : این روش شبیه به روش First-Fit است، با این تفاوت که جستجو برای یافتن اولین محل مناسب، همواره از ابتدای حافظه آغاز نمی‌شود، بلکه جستجو از محل آخرین تخصیص به بعد شروع می‌شود. این الگوریتم از ابتدای حافظه، شروع به کار می‌کند.

محتویات حافظه قبل از زمان t به صورت زیر است:

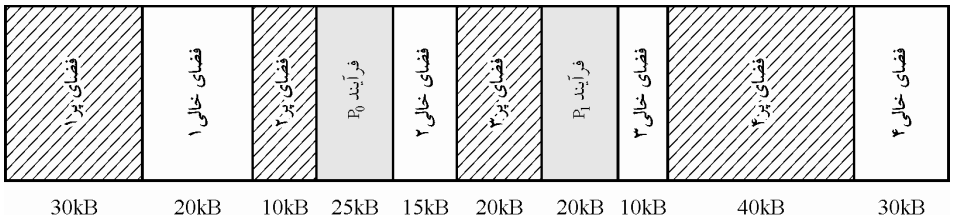


توجه: مطابق وعده سؤال، فقط فضای پُر 3 در لحظه t+20 آزاد می‌گردد و مابقی فضاهای پُر تا اتمام اجرای فرآیندهای مطرح شده، آزاد نمی‌گردد.

در لحظه t، فرآیند P_0، علاقه‌مندی خود را برای ورود به حافظه اعلام می‌کند، میزان حافظه مورد نیاز این فرآیند 25KB است، پس از بررسی واحد مدیریت حافظه توسط الگوریتم Next-Fit مشخص می‌شود این فضا به صورت همجوار موجود است. بنابراین با ورود این فرآیند به حافظه و قرارگیری در صف آماده پردازنده موافقت می‌گردد. بنابراین شکل حافظه به صورت زیر خواهد بود:

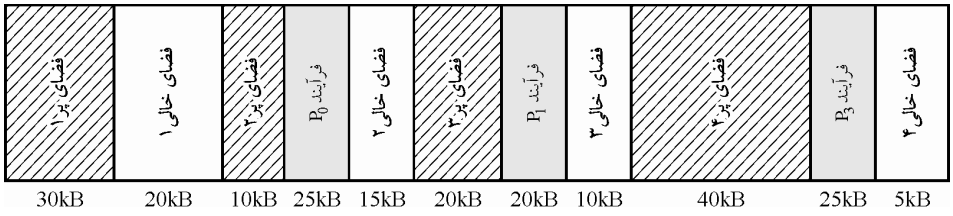


در لحظه t+1، فرآیند P_1، علاقه‌مندی خود را برای ورود به حافظه اعلام می‌کند، میزان حافظه‌ی مورد نیاز این فرآیند 20KB است، پس از بررسی واحد مدیریت حافظه توسط الگوریتم Next-Fit مشخص می‌شود این فضا به صورت همجوار موجود است. بنابراین با ورود این فرآیند به حافظه و قرارگیری در صف آماده پردازنده موافقت می‌گردد. فضای خالی پس از آخرین تخصیص، فضای خالی 2 است، اما کافی نیست، پس جلوتر می‌رویم تا به فضای خالی 3 برسیم که کافی است، بنابراین شکل حافظه به صورت زیر خواهد بود:

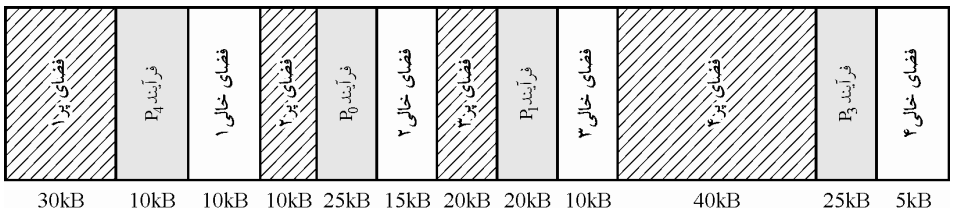


در لحظه t+2، فرآیند P_2، علاقه‌مندی خود را برای ورود به حافظه اعلام می‌کند، میزان حافظه‌ی مورد نیاز این فرآیند 40KB است، پس از بررسی واحد مدیریت حافظه توسط الگوریتم Next-Fit

مشخص می‌شود این فضا به صورت همجوار موجود نیست. بنابراین با ورود این فرآیند به حافظه و قرارگیری در صف آماده پردازنده موافقت نمی‌گردد. بنابراین فرآیند P_2، باید منتظر بماند تا حافظه‌ی مورد نیازش آزاد گردد. اولین آزادسازی حافظه در لحظه t+20 به میزان 20KB می‌باشد که مربوط به فضای پُر 3 می‌باشد، بنابراین فرآیند P_2 فعلاً تا آن لحظه باید صبر کند. در لحظه t+3 فرآیند P_3، علاقه‌مندی خود را برای ورود به حافظه اعلام می‌کند، میزان حافظه مورد نیاز این فرآیند 25KB است، پس از بررسی واحد مدیریت حافظه توسط الگوریتم Next-Fit مشخص می‌شود این فضا به صورت همجوار موجود است. بنابراین با ورود این فرآیند به حافظه و قرارگیری در صف آماده پردازنده موافقت می‌گردد. فضای خالی پس از آخرین تخصیص، فضای خالی 3 است، اما کافی نیست، پس جلوتر می‌رویم تا به فضای خالی 4 برسیم که کافی است، بنابراین شکل حافظه به صورت زیر خواهد بود:



در لحظه t+4، فرآیند P_4، علاقه‌مندی خود را برای ورود به حافظه اعلام می‌کند، میزان حافظه مورد نیاز این فرآیند 10KB است، پس از بررسی واحد مدیریت حافظه توسط الگوریتم Next-Fit مشخص می‌شود این فضا به صورت همجوار موجود است. بنابراین با ورود این فرآیند به حافظه و قرارگیری در صف آماده پردازنده موافقت می‌گردد. فضای خالی پس از آخرین تخصیص، فضای خالی 4 است، اما کافی نیست، از آنجا که الگوریتم به انتهای حافظه رسیده است و امکان جلوتر رفتن وجود ندارد، بنابراین الگوریتم دور زده و مجدداً از ابتدای حافظه شروع به کار می‌کند. اولین فضای خالی در ابتدای حافظه فضای خالی 1 می‌باشد که کافی است. بنابراین شکل حافظه به صورت زیر خواهد بود:



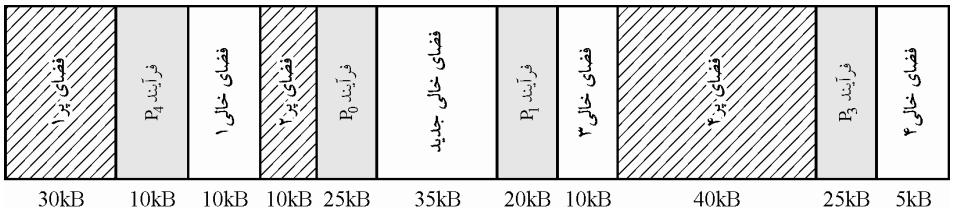
در لحظه t+5، فرآیند P_5، علاقه‌مندی خود را برای ورود به حافظه اعلام می‌کند، میزان حافظه مورد نیاز این فرآیند 35KB است، پس از بررسی واحد مدیریت حافظه توسط الگوریتم Next-Fit مشخص می‌شود این فضا به صورت همجوار موجود نیست. بنابراین با ورود این فرآیند

به حافظه و قرارگیری در صف آماده پردازنده موافقت نمی‌گردد. بنابراین فرآیند P_5 نیز در کنار فرآیند P_2، باید منتظر بماند تا حافظه مورد نیازش آزاد گردد. اولین آزادسازی حافظه در لحظه t+20 به میزان 20KB می‌باشد که مربوط به فضای پُر 3 می‌باشد، بنابراین فرآیند P_5 فعلاً تا آن لحظه باید صبر کند.

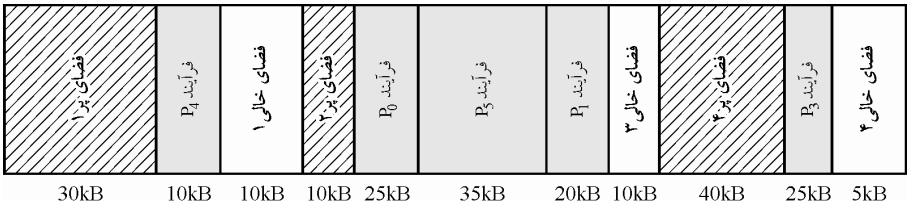
توجه: در حال حاضر فرآیندهای P_0، P_1، P_3 و P_4 در حافظه و در صف آماده پردازنده قرار دارند و مطابق الگوریتم FCFS، به نوبت اجرا خواهند شد.

20 ثانیه بعد...

هم اکنون در لحظه t+20 قرار داریم، در این لحظه 20 ثانیه از اجرای فرآیند P_0 گذشته است و همچنین بالاخره فضای پُر3 مطابق وعده سؤال آزاد می‌گردد. این فضای آزاد شده 20KB، با فضای خالی 2 به میزان 15KB ادغام شده و یک فضای 35KB همجوار را ایجاد می‌کند. بنابراین از بین فرآیندهای منتظر ورود به حافظه یعنی فرآیندهای P_2 و P_5، فرصت برای ورود فرآیند P_5 به حافظه و قرارگیری در صف آماده فراهم می‌گردد. و می‌رود در انتهای صف آماده پردازنده و بعد از فرآیندهای P_0، P_1، P_3 و P_4 قرار می‌گیرد. بنابراین شکل حافظه به صورت زیر خواهد بود:



در لحظه t+20 فرآیندهای P_2 و P_5 هر دو علاقه‌مندی خود را به ترتیب برای ورود به حافظه اعلام می‌کنند، میزان حافظه مورد نیاز این فرآیندها به ترتیب 40KB و 35KB است، پس از بررسی واحد مدیریت حافظه توسط الگوریتم Next-Fit مشخص می‌شود این فضا به صورت همجوار فقط برای فرآیند P_5 موجود است. بنابراین با ورود فرآیند P_5 به حافظه و قرارگیری در صف آماده پردازنده موافقت و با ورود فرآیند P_2 مخالفت می‌گردد. فضای خالی پس از آخرین تخصیص فضای خالی 1 است، که کافی نیست. پس جلوتر می رویم تا به فضای خالی همجوار جدید، حاصل از ادغام دو فضای خالی 2 و 3، به اندازه 35KB برسیم که کافی است. بنابراین شکل حافظه به صورت زیر خواهد بود:

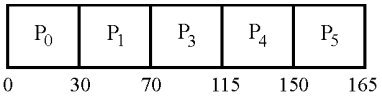


در صورت خروج فرآیندهای P_0، P_1، P_3 و P_4 همچنان فضای آزاد کافی برای ورود فرآیند P_2 به حافظه فراهم نمی‌گردد و فقط با خروج فرآیند P_5 است که فضای آزاد کافی برای ورود فرآیند P_2 فراهم می‌گردد.

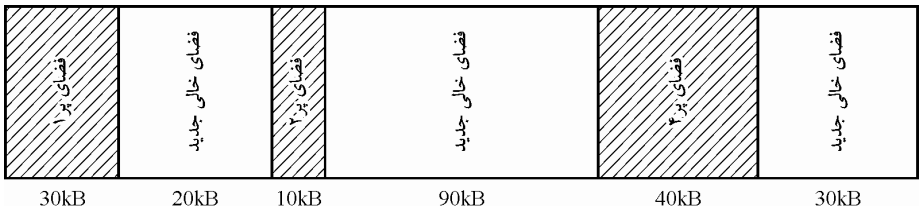
توجه: در حال حاضر فرآیندهای P_0، P_1، P_3، P_4 و P_5 در حافظه و در صف آماده پردازنده قرار دارند و مطابق الگوریتم FCFS، به نوبت اجرا خواهند شد.

165 ثانیه بعد...

نمودار گانت اولیه فرآیندها، براساس زمان اجرای فرآیندها و الگوریتم FCFS به صورت زیر است:

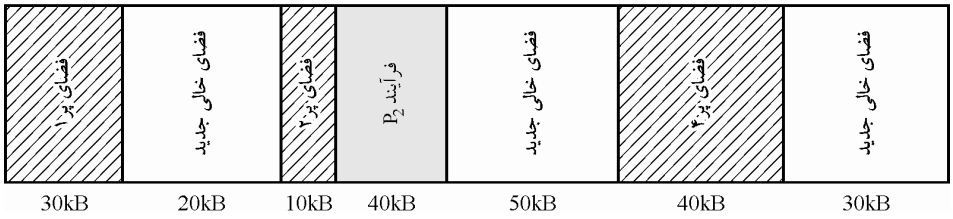


هم اکنون در لحظه t+165 قرار داریم، در این لحظه فرآیندهای P_0، P_1، P_3، P_4 و P_5 به پایان کار خود رسیده‌اند، بنابراین فرصت برای ورود فرآیند P_2 به حافظه و قرارگیری در صف آماده پردازنده فراهم می‌گردد و می‌رود به صورت تنها در ابتدای صف آماده قرار می‌گیرد، زیرا مابقی فرآیندها به دلیل اتمام کار خود از حافظه و صف آماده پردازنده خارج شده‌اند. بنابراین شکل حافظه به صورت زیر خواهد بود:

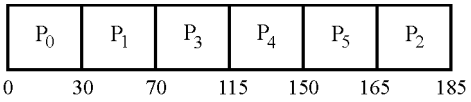


تا قبل از زمان t+165 هرفضایی که به واسطه اتمام فرآیندها آزاد می‌گردد، کل فضای حافظه برای بررسی فضای کافی برای ورود فرآیند P_2 توسط الگوریتم Next-Fit و از آخرین تخصیص مورد بررسی قرار می‌گیرد، اما هیچگاه این الگوریتم موفق به کشف این فضای همجوار کافی برای ورود فرآیند P_2 نمی‌گردد. اما درست در لحظه t+165 اتفاق دیگری می‌افتد.....

در لحظه t+165، فرآیند P_2 پس از آن همه ناکامی و تلاش برای ورود به حافظه و قرارگیری در صف آماده، امیدوارانه باز هم شناس خود را امتحان می‌کند و علاقه‌مندی خود را برای ورود به حافظه باز هم به سیستم اعلام می‌کند، میزان حافظه مورد نیاز این فرآیند 40KB است، پس از بررسی مجدد واحد مدیریت حافظه توسط الگوریتم Next-Fit به صورت غیرمنتظره‌ای مشخص می‌شود که این فضا به صورت همجوار موجود است. بنابراین با ورود این فرآیند به حافظه و قرارگیری در صف آماده پردازنده موافقت می‌گردد و سرانجام فرآیند P_2 در تلاش برای ورود به حافظه و قرارگیری در صف آماده پردازنده، موفق می‌شود.....



نمودار گانت نهایی فرآیندها، براساس زمان اجرای فرآینـدها و الگـوریتم FCFS بـه صـورت زیـر است:



توجه: برای سادگی زمان t را برابر صفر در نظر بگیرید، بنابراین داریم:

فرآیند	زمان ورود	حافظه مورد نیاز	زمان اجرا	زمان انتظار+	زمان بازگشت=
P_0	0	25	30	0	30
P_1	1	20	40	29	69
P_2	2	40	20	163	183
P_3	3	25	45	67	112
P_4	4	10	35	111	146
P_5	5	35	15	145	160

زمان ورود فرآیند- زمان خروج فرآیند= زمان بازگشت فرآیند

P_0 زمان بازگشت = 30-0=30

P_1 زمان بازگشت =70-1=69

P_2 زمان بازگشت =185-2=183

P_3 زمان بازگشت =115-3=112

P_4 زمان بازگشت =150-4=146

P_5 زمان بازگشت =165-5=160

$$\text{میانگین زمان بازگشت} = \frac{30+69+183+112+146+160}{6} = \frac{700}{6} = 116.67$$

زمان اجرای فرآیند - زمان بازگشت فرآیند= زمان انتظار فرآیند

P_0 زمان انتظار =30-30=0

P1 زمان انتظار =69-40=29

P_2 زمان انتظار =183-20=163

P_3 زمان انتظار =112–45=67

P_4 زمان انتظار =146–35=111

P_5 زمان انتظار =160–15=145

$$\text{میانگین زمان انتظار} = \frac{0+29+163+67+111+145}{6} = \frac{515}{6} = 85.83$$

$$\text{میانگین زمان اجرا} = \frac{30+40+20+45+35+15}{6} = \frac{185}{6} = 30.83$$

میانگین زمان انتظار + میانگین زمان اجرا= میانگین زمان بازگشت

116.67 = 30.83 + 85.83

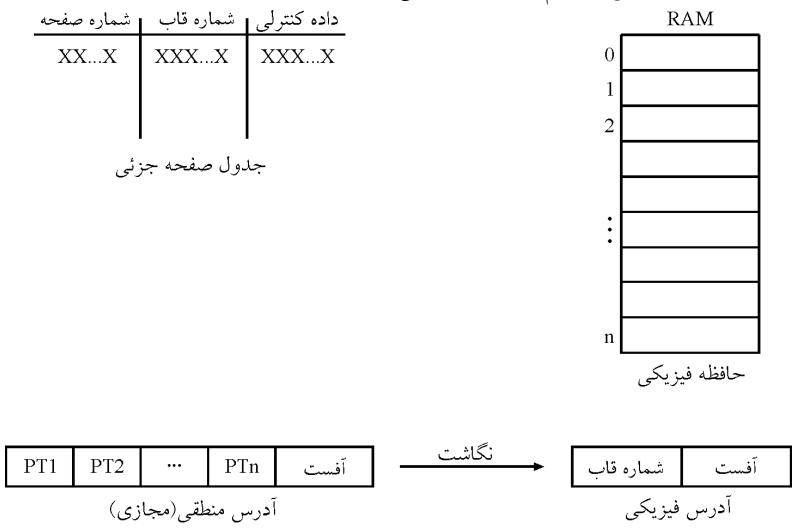
توجه: مطابق رابطه فوق، تفاضل میانگین زمان بازگشت و میانگین زمان انتظار باید برابر میانگین زمان اجرا باشد، که فقط در گزینه اول، سوم و چهارم این مورد رعایت شده است. بنابراین گزینهٔ دوم پاسخ سؤال نیست.

توجه: همچنین مطابق رابطه فوق، میانگین زمان بازگشت همواره از میانگین زمان انتظار بیشتر است، بنابراین از این نگاه همه گزینه‌ها درست هستند، که در اینجا این نگاه کارآمد نیست.

۴- گزینه (۳) صحیح است.

در اینجا برای جدول صفحه جزئی محدودیتی به اندازه یک قاب داریم.

بنابراین اندازه جدول صفحه جزئی برابر اندازه قاب می‌باشد. بنابراین برای محاسبه تعداد سطرهای جدول صفحه جزئی، کافی است، اندازه قاب که برابر اندازه جدول صفحه جزئی است بر اندازه عرض جدول صفحه جزئی تقسیم گردد. به شکل زیر توجه کنید:



توجه: عرض جدول صفحه همواره برابر حاصل جمع تعداد بیت‌های کنترلی و تعداد بیت‌های شماره قاب است، دقت کنید که تعداد بیت‌های شماره صفحه جزو عرض جدول صفحه نمی‌باشد، بلکه شماره صفحه، اندیس هر سطر جدول صفحه می‌باشد.

بنابراین داریم:

تعداد بیت‌های کنترلی + تعداد بیت‌های شماره قاب = عرض جدول صفحه جزئی

10 + تعداد بیت‌های شماره قاب = عرض جدول صفحه جزئی

توجه: 10 بیت کنترلی داریم، مطابق فرض صورت سوال.

تعداد بیت شماره قاب = $\log_2$ تعداد قاب‌های حافظه فیزیکی

تعداد قاب‌های حافظه‌ی فیزیکی = $\frac{\text{اندازه حافظه فیزیکی}}{\text{اندازه قاب}} = \frac{2^{32}B}{2^{10}B} = 2^{22}$

بنابراین:

تعداد بیت شماره قاب = $\log_2$ تعداد قاب‌های حافظه فیزیکی $= \log_2 2^{22} = 22$

عرض جدول صفحه جزئی = 22 + 10 = 32bit = 4Byte

تعداد سطرهای جدول صفحه جزئی = $\frac{\text{اندازه قاب}}{\text{عرض جدول صفحه}} = \frac{2^{10}B}{2^{2}B} = 2^{8} = 256$

اندازه فرآیند (فضای آدرس مجازی) = $2^{40}B$

اندازه صفحه = اندازه قاب = $2^{10}B$

تعداد صفحات فرآیند: f = $\frac{\text{اندازه فرآیند}}{\text{اندازه صفحه}} = \frac{2^{40}}{2^{10}} = 2^{30}$

تعداد سطرهای جدول صفحه جزئی: r = 256 = 2^8

حال اطلاعات کافی برای محاسبه تعداد سطوح جدول صفحه چند سطحی را در اختیار داریم:

روش تجزیه

تعداد صفحات فرآیند باید در اندازه r (2^8) تجزیه گردد:

تعداد صفحات فرآیند = $2^{30} = 2^{⑥} \times 2^{⑧} \times 2^{⑧} \times 2^{⑧}$ (PT1 PT2 PT3 PT4)

توجه: تعداد عملوندها برابر 4 است، بنابراین تعداد سطوح جدول چند سطحی نیز برابر 4 است.

روش لگاریتم

$$\text{تعداد سطوح جدول چند سطحی} = d = \left\lceil \log_r^f \right\rceil = \left\lceil \log_{2^8}^{2^{30}} \right\rceil = 4$$

روش تقسیم متوالی

$$\text{تعداد جداول صفحه جزئی در سطح چهارم} = \frac{\text{تعداد صفحات فرآیند}}{r} = \frac{f}{r} = \frac{2^{30}}{2^8} = 2^{22}$$

$$\text{تعداد جداول صفحه جزئی در سطح سوم} = \frac{\text{تعداد جداول صفحه جزئی در سطح چهارم}}{r} = \frac{2^{22}}{2^8} = 2^{14}$$

$$\text{تعداد جداول صفحه جزئی در سطح دوم} = \frac{\text{تعداد جداول صفحه جزئی در سطح سوم}}{r} = \frac{2^{14}}{2^8} = 2^6$$

$$\text{تعداد جداول صفحه جزئی در سطح اول} = \frac{\text{تعداد جداول صفحه جزئی در سطح دوم}}{r} = \frac{2^6}{2^8} < 1$$

توجه: سطح اول، یک جدول به حساب می‌آید، که 2^6 سطر بیشتر نیاز ندارد.

توجه: تعداد تقسیم متوالی برابر 4 است، بنابراین تعداد سطوح جدول چند سطحی برابر 4 است.

$$\text{تعداد بیت آفست} = \log_2^{\text{اندازه صفحه}} = \log_2^{2^{10}} = 10 \text{ بیت}$$

بنابراین شکل آدرس منطقی (مجازی) به صورت زیر خواهد بود:

PT1	PT2	PT3	PT4	آفست
6 بیت	8 بیت	8 بیت	8 بیت	10 بیت
30 بیت				

۵- گزینه (۳) 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. ($2^{32} / 2^{12} = 2^{20}$). فرض کنید هر درایه جدول صفحه معمولی، شامل 4 بایت باشد، در اینصورت هر فرآیند به (4×2^{20}) 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 گزینه سوم نادرست است.** زیرا در جدول صفحه وارونه (معکوس)، فقط اطلاعات فضای آدرس منطقی برای یک پردازه (فرآیند) در دسترس نیست، بلکه اطلاعات فضای آدرس مربوط به **صفحات حاضر** در حافظه برای **همه** پردازه‌ها (فرآیندها) در دسترس است، اینکه کدام صفحه مربوط به کدام فرآیند در قاب‌های حافظه ذخیره شده است. همچنین تعداد درایه‌های جدول صفحه معکوس برابر تعداد قاب‌های حافظه فیزیکی (اصلی) است. یعنی به ازای هر صفحه فیزیکی حافظه در جدول صفحه وارونه تنها یک خانه وجود دارد، **بنابراین گزینه اول درست 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

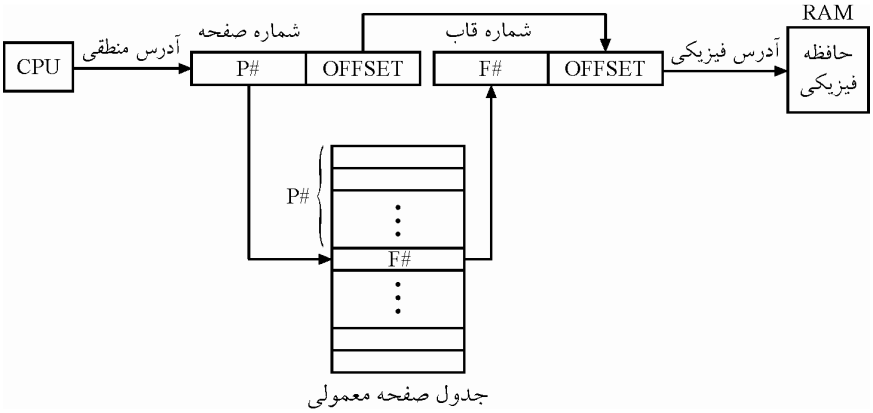
$$\text{تعداد قاب‌های حافظه فیزیکی} = \frac{\text{اندازه حافظه فیزیکی}}{\text{اندازه صفحه یا اندازه قاب}} = \frac{2^7 \times 2^{20}}{2^{11}} = 2^{16}$$

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:

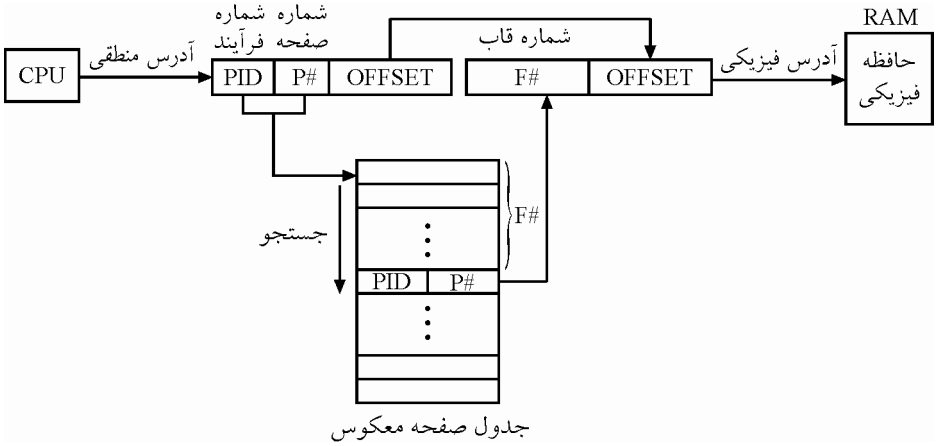
$$\text{تعداد درایه‌های جدول صفحه معکوس} = \frac{\text{اندازه حافظه فیزیکی}}{\text{اندازه صفحه یا اندازه قاب}} = \frac{2^7 \times 2^{20}}{2^{11}} = 2^{16}$$

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.

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



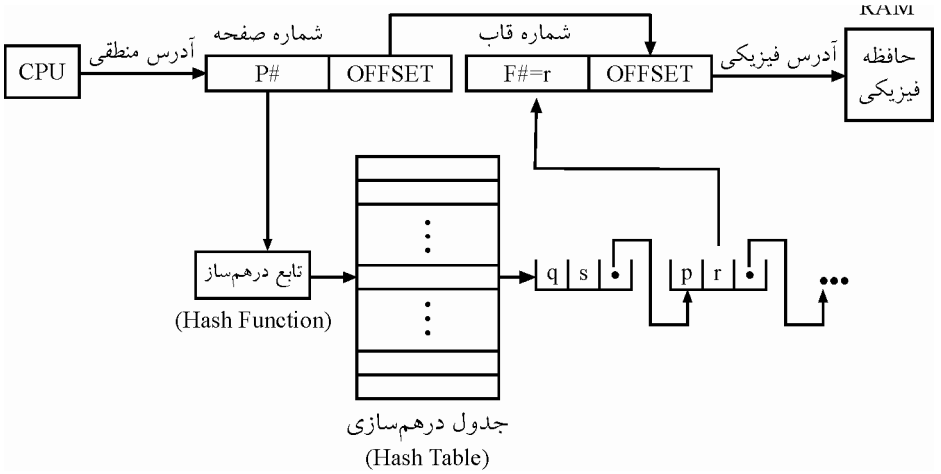
نحوه ترجمه آدرس منطقی به فیزیکی، در سیستم جدول **صفحه معکوس** به صورت زیر است:



بنابراین مطابق شکل فوق **گزینه چهارم درست 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 دوم 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.

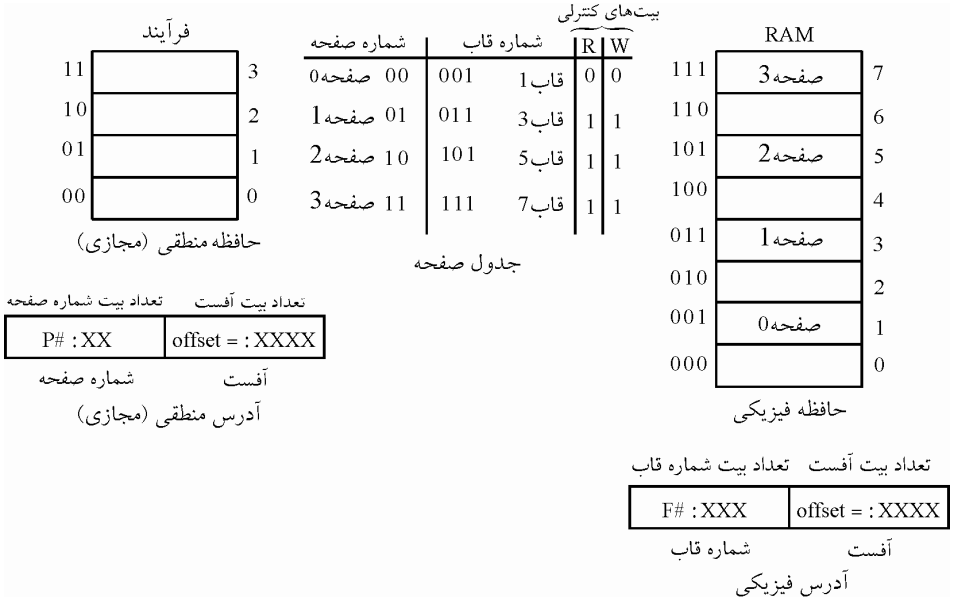
توجه: در سیستم‌هایی که از جدول صفحه معکوس‌شده استفاده می‌کنند، استفاده اشتراکی از صفحات میان فرآیندها در حالت عادی امکان‌پذیر نیست، در حالت کلی اشتراک به شکل چند به یک است، یعنی صفحه مشترک میان فرآیندها در یک قاب مشخص قرار می‌گیرد. که پیاده‌سازی حالت چند به یک در جدول صفحه معکوس درحالت ابتدایی امکان‌پذیر 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 بـرای هـر درایـه، بـیش از یـک شماره صفحه را آدرس‌دهی کند.

۶- گزینه (۳) صحیح است.

به طور کلی روابط میان آدرس منطقی و آدرس فیزیکی در راه حل صفحه‌بنـدی بـه صـورت زیـر است:

مثال: فرض کنید اندازه‌ی هر صفحه 16 بایت، اندازه فرآیند 64 بایت و اندازه حافظه فیزیکی برابـر 128 بایت باشد.



$$\text{تعداد صفحات فرآیند (تعداد درایه‌های جدول صفحه)} = \frac{\text{اندازه فرآیند}}{\text{اندازه صفحه یا اندازه قاب}} = \frac{64}{16} = 4$$

$$\text{تعداد قاب‌های حافظه فیزیکی} = \frac{\text{اندازه حافظه فیزیکی}}{\text{اندازه صفحه یا اندازه قاب}} = \frac{128}{16} = 8$$

$$\text{تعداد بیت شماره صفحه} = b = \log_2^{\text{تعداد صفحات فرآیند}} = \log_2^4 = 2\text{bit}$$

$$\text{تعداد بیت شماره قاب} = b = \log_2^{\text{تعداد قاب‌های حافظه فیزیکی}} = \log_2^8 = 3\text{bit}$$

$$\text{تعداد بیت آفست} = b = \log_2^{\text{اندازه صفحه یا اندازه قاب}} = \log_2^{16} = 4\text{bit}$$

عرض جدول صفحه × تعداد صفحات فرآیند (تعداد درایه‌های جدول صفحه) = اندازه جدول صفحه

توجه: عرض جدول صفحه برابر حاصل جمع تعداد بیت‌های کنترلی و تعداد بیت‌های شماره قاب می‌باشد، دقت کنید که تعداد بیت‌های شماره صفحه جزو عرض جدول صفحه نمی‌باشد، بلکه شماره صفحه، اندیس هر سطر جدول صفحه می‌باشد، به صورت زیر:

تعداد بیت‌های کنترلی + تعداد بیت‌های شماره قاب = عرض جدول صفحه

در جدول فوق 2 بیت مربوط به بیت‌های کنترلی و 3 بیت مربوط به تعداد بیت‌های شماره قاب می‌باشد.

پس عرض جدول صفحه فوق برابر $3\text{bit} + 2\text{bit} = 5\text{bit}$ می‌باشد.

همانطور که گفتیم، اندازه جدول صفحه، از رابطه زیر محاسبه می‌گردد:

عرض جدول صفحه × تعداد صفحات فرآیند (تعداد درایه‌های جدول صفحه) = اندازه جدول صفحه

که مطابق رابطه فوق داریم:

$$\text{اندازه جدول صفحه} = 4 \times 5\,\text{bit} = 20\,\text{bit}$$

همچنین، اندازه آدرس‌های منطقی (مجازی) و فیزیکی به صورت زیر است:

$$\text{اندازه آدرس منطقی} = \text{تعداد بیت شماره صفحه} + \text{تعداد بیت آفست} = 2\text{bit} + 4\text{bit} = 6\,\text{bit}$$

$$\text{اندازه آدرس فیزیکی} = \text{تعداد بیت شماره قاب} + \text{تعداد بیت آفست} = 3\text{bit} + 4\text{bit} = 7\,\text{bit}$$

همچنین داریم:

$$\text{اندازه حافظه منطقی (فرآیند)} = 2^{\text{تعداد بیت آدرس منطقی}} = 2^{\text{تعداد بیت شماره صفحه}} \times 2^{\text{تعداد بیت آفست}}$$

$$\text{اندازه حافظه منطقی (فرآیند)} = 2^6 = 2^2 \times 2^4 = 2^6 = 64\,\text{Byte}$$

$$\text{اندازه حافظه فیزیکی (RAM)} = 2^{\text{تعداد بیت آدرس فیزیکی}} = 2^{\text{تعداد بیت شماره قاب}} \times 2^{\text{تعداد بیت آفست}}$$

$$\text{اندازه حافظه فیزیکی (RAM)} = 2^7 = 2^3 \times 2^4 = 2^7 = 128\ \text{Byte}$$

توجه: بنابراین اندازه آدرس منطقی (مجازی) و فیزیکی (حقیقی) الزاماً برابر نیست. پس گزینه‌ی اول نادرست است.

گزینه‌ی دوم نادرست است. زیرا، عمل تعویض متن (مابین فرآیندها یا نخ‌ها) توسط بخشی از سیستم عامل به نام Dispatcher صورت می‌گیرد. در واقع Dispatcher ماژولی است که کنترل پردازنده را به فرآیندی که توسط زمان‌بند کوتاه‌مدت انتخاب شده است، اعطا می‌کند. همچنین بارگذاری و ذخیره‌سازی PCB و یا به نخی که توسط زمان‌بند نخ (در سطح هسته یا ترکیبی) انتخاب شده است، اعطا می‌کند. همچنین بارگذاری و ذخیره‌سازی TCB.

توجه: نخی که توسط زمان‌بند سطح کاربر، زمان‌بندی می‌شود، توسط سیستم عامل شناخته نمی‌شود، در چنین شرایطی سیستم عامل فقط فرآیند را می‌شناسد.

گزینه‌ی سوم درست است. زیرا، اگر یک نخ (ریسه)، cpu – bound باشد، بهتر است اولویت آن برای دسترسی به I/O بالاتر از نخ‌های I/O – bound باشد، چون زودتر I/O را رها می‌کند و برای استفاده بهتر از پردازنده اقدام می‌کند.

گزینه‌ی چهارم نادرست است. زیرا garbage (زباله) از مشکلات حاصل از استفاده از اشاره‌گرها ایجاد می‌گردد، اما Fragmentation (تکه‌تکه شدن) که بر دو نوع تکه‌تکه شدن داخلی و خارجی است، به دلیل روش‌های مدیریت حافظه ایجاد می‌گردد.

تکه‌تکه شدن داخلی زمانی رخ می‌دهد که حافظه از قبل مرزبندی شده باشد و در هر قسمت بتوان فقط یک قطعه را قرار داد. در این حالت چون احتمالاً قطعه‌ای که برای یک جایگاه انتخاب شده است قدری از آن کوچک‌تر است، در انتهای جایگاه مقداری فضای بدون استفاده به وجود خواهد آمد. به این مشکل تکه‌تکه شدن داخلی گویند.

اما وقتی از قبل مرزبندی مشخصی برای حافظه در نظر نگیریم، احتمال وقوع تکه تکه شدن خارجی وجود دارد. در این حالت احتمالاً فضاهای خالی و قابل استفاده حافظه به صورت جایگاه‌های کوچک، لابه‌لای فرآیندها پخش شده‌اند. که مجموع این جایگاه‌های کوچک پراکنده احتمالاً نیاز ما را برآورده می‌کنند، اما چون همجوار نیستند، بدون استفاده باقی می‌مانند. روش فشرده‌سازی (compaction) در صورت اجرا شدن، این مشکل را برطرف می‌کند. البته در روش قطعه‌بندی نیز با وجود تخصیص غیرهمجوار ولی به دلیل عدم مرزبندی باز هم تکه‌تکه شدن خارجی وجود دارد.

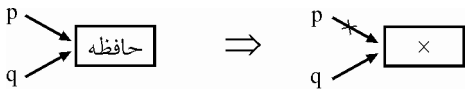
مشکلات اشاره‌گرها

به طور کلی کار کردن با اشاره‌گرها ممکن است باعث ایجاد مشکلاتی گردد این مشکلات عبارتند از:

۱- ارجاع معلق (Dangling Reference): به این معناست که شیء داده‌ای وجود ندارد ولی مسیر دستیابی به آن معتبر است. این مشکل زمانی پیش می‌آید که فضایی که چندین اشاره‌گر به صورت همزمان به آن اشاره می‌کنند توسط یکی از اشاره‌گرها آزاد شده اما بقیه اشاره‌گرها استفاده شوند و از بین نروند.

مثال:

```
int  *p,*q;
p=malloc(20);
q=p
free(p);
```

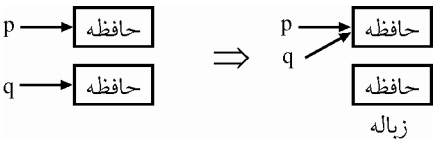


Dangling Reference

۲- زباله (Garbage): به این معناست که شیء داده‌ای وجود دارد ولی هیچ‌گونه امکان دسترسی به آن وجود ندارد. این مشکل زمانی رخ می‌دهد، که فضایی که به یک شیء داده‌ای اختصاص‌یافته است آزاد نشده اما اشاره‌گر به آن شی از بین رفته است.

مثال:

```
int  *p, *q;
p = malloc (20) ;
q = malloc (20);
q = p;
```

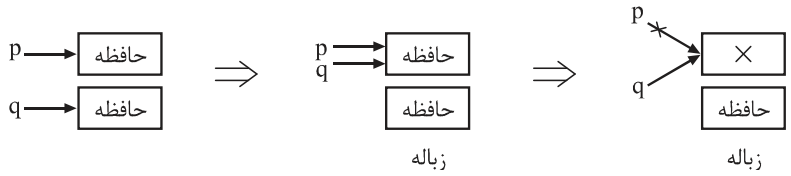


مشکل Garbage

توجه: گاهی ممکن است این دو مشکل به صورت همزمان نیز اتفاق بیفتد.

مثال:

```
int  *p, *q;
p = malloc (20) ;
q = malloc (20);
q = p;
free (p);
```



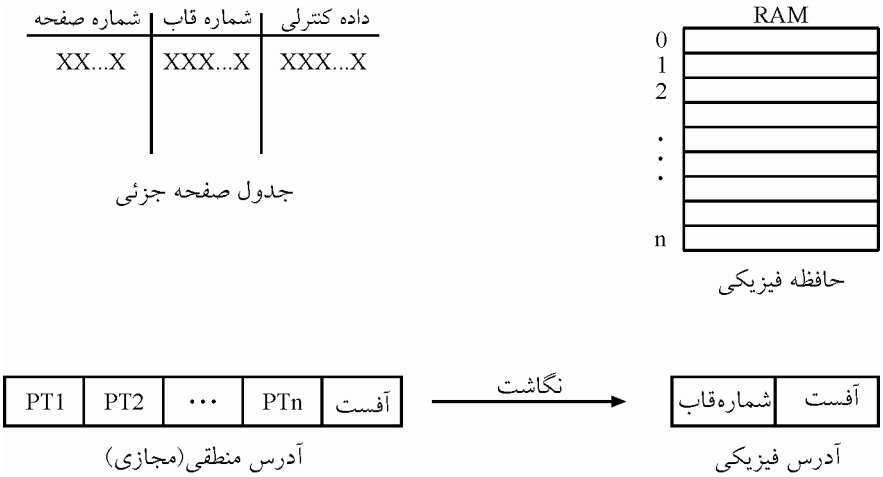
مشکل Garbage و Dangling Reference به صورت همزمان

توجه: یک راه حل برای مشکل زباله (Garbage) این است که هرگاه به یک اشاره‌گر فضایی را اختصاص دادیم، به طور اتوماتیک یک فیلد اضافی به نام Reference Count در نظر بگیریم که نشان‌دهنده تعداد اشاره‌گرها به آن محل از حافظه است. هرگاه یک اشاره‌گر جدید به این محل اشاره کند مقدار Count یکی افزایش یافته و هرگاه یک اشاره‌گر آزاد شود مقدار Count یک واحد کاهش می‌یابد. به این ترتیب هرگاه مقدار Count صفر شود هیچ اشاره‌گری به فضای مربوطه اشاره نمی‌کند و فضای مورد نظر می‌تواند آزاد گردد. عمل آزادسازی Garbageها توسط واحدی تحت عنوان Garbage Collector انجام می‌گیرد.

۷- گزینه (۲) صحیح است.

در اینجا برای جدول صفحه جزئی محدودیتی به اندازه یک قاب (صفحه) داریم.

بنابراین اندازه جدول صفحه جزئی برابر اندازه قاب (صفحه) می‌باشد. بنابراین برای محاسبه تعداد سطرهای جدول صفحه جزئی، کافی است، اندازه قاب که برابر اندازه جدول صفحه جزئی است بر اندازه عرض جدول صفحه جزئی تقسیم گردد. به شکل زیر توجه کنید:



توجه: عرض جدول صفحه همواره برابر حاصل جمع تعداد بیت‌های کنترلی و تعداد بیت‌های شماره قاب است، دقت کنید که تعداد بیت‌های شماره صفحه جزو عرض جدول صفحه نمی‌باشد، بلکه شماره صفحه، اندیس هر سطر جدول صفحه می‌باشد.

بنابراین داریم:

تعداد بیت‌های کنترلی + تعداد بیت‌های شماره قاب= عرض جدول صفحه جزئی

$4B =$ عرض جدول صفحه جزئی

توجه: مطابق فرض سؤال، هر مدخل جدول صفحه (عرض جدول صفحه جزئی) 4 بایت در نظر گرفته شده است.

$$\text{تعداد سطرهای جدول صفحه جزئی} = \frac{\text{اندازه قاب}}{\text{عرض جدول صفحه}} = \frac{2^{10}B}{2^{2}B} = 2^{8} = 256$$

$2^{34}B =$ اندازه فرآیند (فضای آدرس مجازی)

$1KB = 1024B = 2^{10}B =$ اندازه صفحه = اندازه قاب

$$f: \text{تعداد صفحات فرآیند} = \frac{\text{اندازه فرآیند}}{\text{اندازه صفحه}} = \frac{2^{34}}{2^{10}} = 2^{24}$$

$r: \text{تعداد سطرهای جدول صفحه جزئی} = 256 = 2^{8}$

حال اطلاعات کافی برای محاسبه تعداد سطوح جدول صفحه چند سطحی را در اختیار داریم:

روش تجزیه

تعداد صفحات فرآیند باید در اندازه r(2^8) تجزیه گردد:

$$\text{تعداد صفحات فرآیند} = 2^{24} = 2^{\overset{\uparrow}{\textcircled{8}}} \times 2^{\overset{\uparrow}{\textcircled{8}}} \times 2^{\overset{\uparrow}{\textcircled{8}}} \quad \begin{matrix} PT1 & PT2 & PT3 \end{matrix}$$

توجه: تعداد عملوندها برابر 3 است، بنابراین تعداد سطوح جدول چند سطحی نیز برابر 3 است.

روش لگاریتم

$$\text{تعداد سطوح جدول چند سطحی} = d = \left\lceil \log_r^f \right\rceil = \left\lceil \log_{2^8}^{2^{24}} \right\rceil = 3$$

روش تقسیم متوالی

$$\text{تعداد جداول صفحه جزئی در سطح سوم} = \frac{\text{تعداد صفحات فرآیند}}{r} = \frac{f}{r} = \frac{2^{24}}{2^8} = 2^{16}$$

$$\text{تعداد جداول صفحه جزئی در سطح دوم} = \frac{\text{تعداد جداول صفحه جزئی در سطح سوم}}{r} = \frac{2^{16}}{2^8} = 2^8$$

$$\text{تعداد جداول صفحه جزئی در سطح اول} = \frac{\text{تعداد جداول صفحه جزئی در سطح دوم}}{r} = \frac{2^8}{2^8} = 1$$

توجه: سطح اول، یک جدول به حساب می‌آید، که 2^8 سطر دارد.

توجه: تعداد تقسیم متوالی برابر 3 است، بنابراین تعداد سطوح جدول صفحه چند سطحی برابر 3 است.

$$\text{تعداد بیت آفست} = \log_2^{\text{اندازه صفحه}} = \log_2^{2^{10}} = 10 \text{ بیت}$$

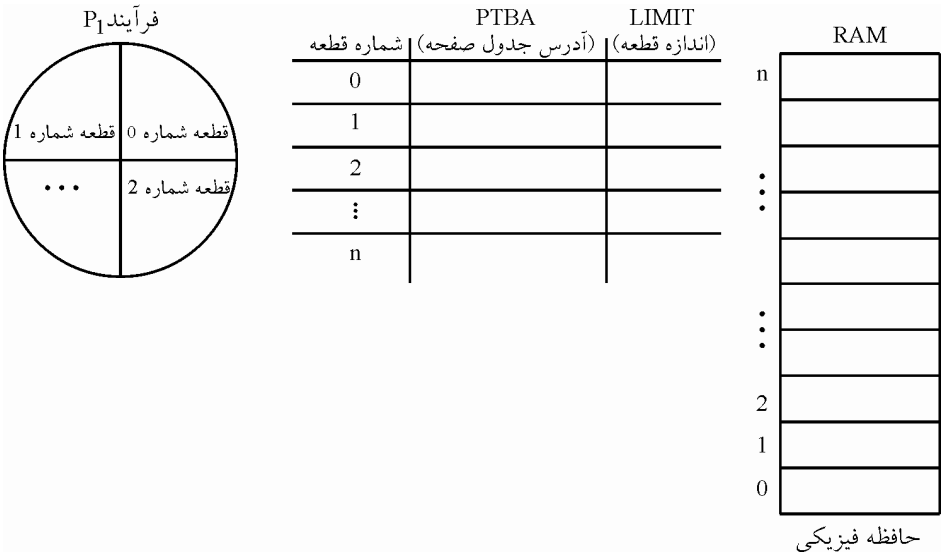
بنابراین شکل آدرس منطقی (مجازی) به صورت زیر خواهد بود:

PT1	PT2	PT3	آفست
8 بیت	8 بیت	8 بیت	10 بیت
24 بیت			

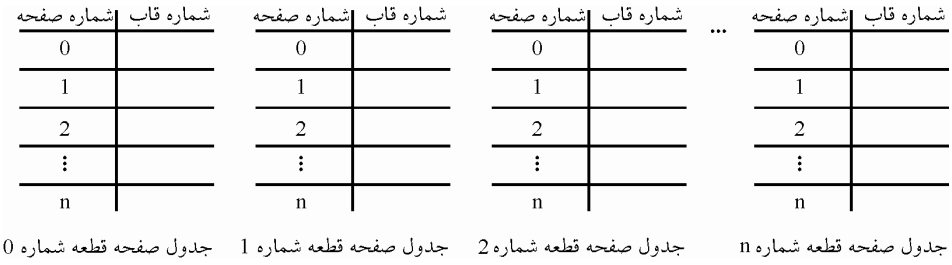
۸- گزینه (۱) صحیح است.

در راه‌حل قطعه‌بندی همراه با صفحه‌بندی، قطعات تولید شده توسط برنامه نویس یا کامپایلر، توسط سیستم عامل صفحه‌بندی می‌شوند. در این حالت به ازای هر فرآیند یک جدول قطعه ایجاد می‌شود که تعداد درایه‌های جدول قطعه، بستگی به اندازه فرآیند و تعداد قطعات حاصل دارد.

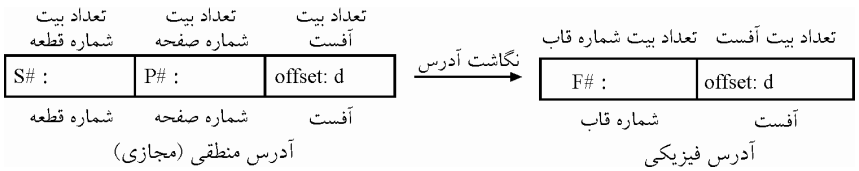
به طور کلی روابط میان آدرس منطقی و آدرس فیزیکی در راه‌حل قطعه‌بندی همراه با صفحه‌بندی، به صورت زیر است:



همچنین به ازای هر قطعه، یک جدول صفحه ایجاد می‌شود که تعداد درایه‌های جدول صفحه بستگی به اندازه قطعه و تعداد صفحات حاصل دارد.



به طور کلی روابط میان آدرس منطقی و آدرس فیزیکی در راه‌حل قطعه‌بندی همراه با صفحه‌بندی، به صورت زیر است:



بر اساس آدرس منطقی (مجازی) و مطابق مفروضات صورت سؤال داریم:

اندازه صفحه یا اندازه قاب × تعداد صفحات فرآیند × تعداد قطعات فرآیند = اندازه حافظه مجازی یا منطقی (فرآیند)

تعداد قطعات فرآیند $= 2^{\text{تعداد بیت شماره قطعه}} = 2^4$

تعداد صفحات فرآیند $= 2^{\text{تعداد بیت شماره صفحه}} = 2^{12}$

اندازه صفحه یا اندازه قاب $= 2^{\text{تعداد بیت آفست}} = 4096$ word (کلمه)

بنابراین داریم:

$2^4 \times 2^{12} \times 4096$ = اندازه حافظه مجازی یا منطقی (فرآیند)

همچنین به بیانی دیگر داریم:

تعداد بیت آدرس منطقی 2 = اندازه حافظه مجازی یا منطقی (فرآیند)

تعداد بیت آفست $2\times$ تعداد بیت شماره صفحه 2 = تعداد بیت شماره قطعه 2 =

$2^4 \times 2^{12} \times 4096$ = اندازه حافظه مجازی یا منطقی (فرآیند)

براساس آدرس فیزیکی و مطابق مفروضات صورت سؤال داریم:

توجه: منظور از کلمه «**بلوک**» در جدول مربوط به صورت سؤال همان فریم یا قاب است.

اندازه صفحه یا اندازه قاب × تعداد قاب‌های حافظه اصلی = اندازه حافظه اصلی یا فیزیکی (RAM)

تعداد قاب‌های حافظه اصلی (فیزیکی) $= 2$ تعداد بیت شماره قاب $= 2^{12}$

اندازه صفحه یا اندازه قاب $= 2$ تعداد بیت آفست $= 4096$ word (کلمه)

بنابراین داریم:

$2^{12} \times 4096$ = اندازه حافظه اصلی یا فیزیکی (RAM)

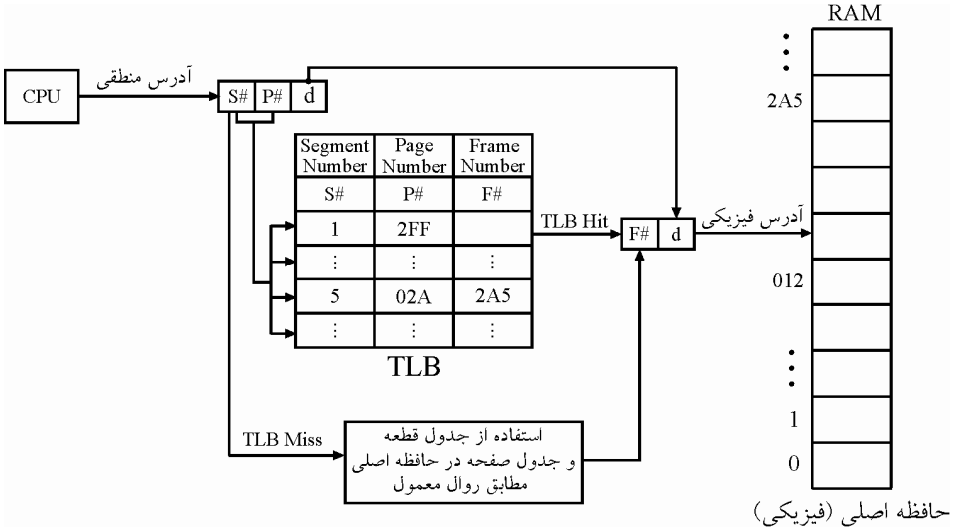
همچنین به بیانی دیگر داریم:

تعداد بیت آفست $2\times$ تعداد بیت شماره قاب 2 = تعداد بیت آدرس فیزیکی 2 = اندازه حافظه اصلی یا فیزیکی (RAM)

$2^{12} \times 4096$ = اندازه حافظه اصلی یا فیزیکی (RAM)

توجه: جهت افزایش سرعت نگاشت آدرس منطقی به فیزیکی می‌توان از TLB نیز استفاده نمود. TLB از حافظه‌های با سرعت بسیار بالا و گران‌قیمت ساخته شده است که در واقع مجموعه‌ای از رجیسترها موسوم به Associative Register هستند. در سیستم قطعه‌بندی همراه با صفحه‌بندی، وقتی فرآیندی جهت اجرا انتخاب شود، بخشی از درایه‌های جدول قطعه و جدول صفحه آن به TLB منتقل می‌شود.

در این حالت وقتی CPU یک آدرس منطقی (مجازی) تولید می‌کند، شماره قطعه و شماره صفحه آن ابتدا در TLB جستجو می‌شود، اگر شماره قطعه و شماره صفحه در TLB یافت شد که آدرس قاب متناظر در حافظه اصلی (فیزیکی) به دست می‌آید. اما اگر شماره قطعه و شماره صفحه در TLB نباشد، آنگاه طبق روال معمول به سراغ جدول قطعه و جدول صفحه در حافظه اصلی می‌رویم. مطابق مفروضات صورت سؤال داریم:



در صورت سؤال مطرح شده است که حجم حافظه مجازی (منطقی) چند برابر حافظه اصلی (فیزیکی) است، که مطابق آنچه بررسی کردیم، به صورت زیر قابل محاسبه است:

$$\frac{\text{حجم یا اندازه حافظه مجازی (منطقی)}}{\text{حجم یا اندازه حافظه اصلی (فیزیکی)}} = \frac{2^4 \times 2^{12} \times 4096}{2^{12} \times 4096} = 2^4 = 16$$

۹- گزینه (۲) صحیح است.

پساکنکور: اغلب بچه‌ها بعد از کنکور دولتی ۱۳۹۵ و همزمان با اوج‌گیری اعتراضات و اعلام کلید اولیه آزمون دولتی ۹۵ از سوی سازمان سنجش آموزش کشور به دلیل وجود دو ابهام در صورت سؤال، به این سؤال معترض و خواهان حذف این سؤال از سوی سازمان سنجش آموزش کشور بودند، اما حکایت به گونه‌ای دیگر رقم خورد و سازمان محترم سنجش و آموزش کشور بی‌اعتنا و بی‌توجه به اعتراض‌های به حق بچه‌ها، در نهایت در کلید نهایی خود (موسوم به کلید ثانویه) نیز مجدداً گزینه دوم را به عنوان پاسخ نهایی اعلام کرد.

ابهام اول: در صورت سؤال زمان خواندن از حافظه اصلی یعنی 100ms در واحد میلی ثانیه بیان شده است. اما درخواست سؤال مبنی بر محاسبه زمان خواندن از TLB در واحد نانوثانیه است که منجر به این می‌شود، اطلاعات سؤال با گزینه‌ها سازگاری نداشته باشد. که به نظر می‌رسد به هنگام حروفچینی سؤال 100ns به شکل 100ms تایپ شده است (خطای حروفچینی).

توجه: ما در هنگام حل سؤال زمان خواندن از حافظه اصلی را 100ns در واحد نانوثانیه در نظر می‌گیریم.

ابهام دوم: در صورت سؤال به وضوح مشخص نشده است که منظور از کارایی سیستم، چه نوع کارایی است. اینکه **کارایی زمان دسترسی سیستم** مدنظر طراح است؟ و یا **کارایی زمان ترجمه**

سیستم؟ اما طراح با فرض کارایی زمان دسترسی سیستم، سؤال را طرح کرده است (خطای طراح سؤال).

توجه: ما در هنگام حل سؤال، هر دو فرض را در نظر می‌گیریم.

در صورت سؤال مطرح شده است که سیستمی علاوه بر ذخیره جدول صفحه در حافظه اصلی، از جدول TLB نیز با نرخ miss برابر 20% استفاده می‌کند. اگر خواندن از حافظه اصلی 100ns زمان بردارد و درصد کارایی سیستم در صورت استفاده نکردن از جدول TLB برابر 80% باشد، خواندن از TLB چند نانوثانیه زمان لازم دارد؟

فرض اول: کارایی زمان دسترسی سیستم

مطابق رابطه زیر داریم:

زمان دسترسی سیستم بدون استفاده از TLB × کارایی زمان دسترسی سیستم = زمان دسترسی مؤثر سیستم با استفاده از TLB

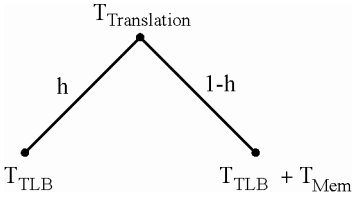
توجه: زمان دسترسی مؤثر سیستم با استفاده از جدول TLB از رابطه زیر محاسبه می‌گردد:

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

که پارامترهای آن به صورت زیر است:

$T_{Translation}$: میانگین زمان ترجمه

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



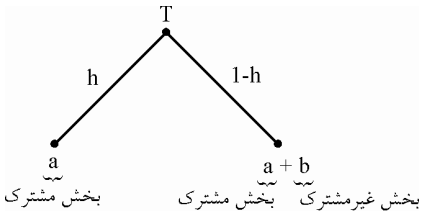
رابطه درخت فوق به صورت زیر خواهد بود:

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

که پس از ساده سازی داریم:

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

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



T = بخش مشترک + (1−h)× 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:

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

$T_{Destination}$: میانگین زمان دسترسی به مقصد مورد نظر

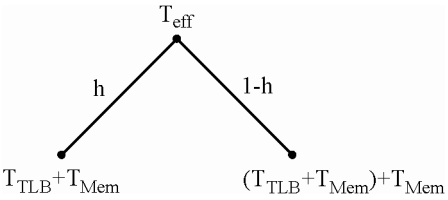
از آنجاییکه مطابق مفروضات مسأله، فقط حافظه اصلی به عنوان مقصد موردنظر، درنظر گرفته شده است، پس $T_{Destination}$ به صورت زیر محاسبه می‌گردد:

$$T_{Destination} = T_{Mem}$$

پس در نهایت زمان دسترسی مؤثر سیستم با استفاده از TLB مطابق آنچه گفتیم به صورت زیر محاسبه می‌گردد:

$$T_{eff} = T_{Access} = T_{TLB} + (1-h)T_{Mem} + T_{Mem}$$

توجه: در یک نگاه کلی‌تر، زمان دسترسی مؤثر سیستم با استفاده از جدول TLB را می‌توان از طریق درخت زیر نیز محاسبه نمود:



مطابق اتحاد درخت احتمال، رابطه زیر را خواهیم داشت:

$$T_{eff} = T_{Access} = T_{TLB} + T_{Mem} + (1-h) \times T_{Mem}$$

توجه: زمان دسترسی سیستم بدون استفاده از جدول TLB از رابطه زیر محاسبه می‌گردد:

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

که پارامترهای آن به صورت زیر است:

$T_{Translation}$: زمان ترجمه

$$T_{Translation} = T_{Mem}$$

$T_{Destination}$: زمان دسترسی به مقصد مورد نظر

$$T_{Destination} = T_{Mem}$$

پس در نهایت زمان دسترسی سیستم بدون استفاده از جدول TLB مطابق آنچه گفتیم به صورت زیر محاسبه می‌گردد:

$$T_{Access} = T_{Mem} + T_{Mem} = 2 \times T_{Mem}$$

در نهایت مطابق آنچه گفتیم داریم:

زمان دسترسی سیستم بدون استفاده از TLB × کارایی زمان دسترسی سیستم = زمان دسترسی مؤثر سیستم با استفاده از TLB

$$T_{TLB} + (1-h)T_{Mem} + T_{Mem} = 0.8 \times (2 \times T_{Mem})$$

$$T_{TLB} + 0.2 \times 100 + 100 = 0.8 \times (2 \times 100)$$

$$T_{TLB} + 20 + 100 = 160$$

$$T_{TLB} = 160 - 120 = 40 \text{ ns}$$

بنابراین اگر منظور از کارایی مطرح شده در صورت سؤال، کارایی زمان دسترسی سیستم باشد، آنگاه گزینه دوم پاسخ سؤال خواهد بود.

فرض دوم: کارایی زمان ترجمه سیستم

مطابق رابطه زیر داریم:

زمان ترجمه سیستم بدون استفاده از TLB × کارایی زمان ترجمه سیستم = زمان ترجمه موثر سیستم با استفاده از TLB

توجه: مطابق آنچه گفتیم، زمان ترجمه سیستم با استفاده از جدول TLB از رابطه زیر محاسبه می‌گردد:

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

توجه: همچنین، مطابق آنچه گفتیم، زمان ترجمه سیستم بدون استفاده از جدول TLB از رابطه زیر محاسبه می‌گردد:

$$T_{Translation} = T_{Mem}$$

در نهایت مطابق آنچه گفتیم داریم:

زمان ترجمه سیستم بدون استفاده از TLB × کارایی زمان ترجمه سیستم = زمان ترجمه موثر سیستم با استفاده از TLB

$$T_{TLB} + (1-h) \times T_{Mem} = 0.8 \times T_{Mem}$$

$$T_{TLB} + 0.2 \times 100 = 0.8 \times 100$$

$$T_{TLB} + 20 = 80$$

$$T_{TLB} = 60 \text{ ns}$$

بنابراین اگر منظور از کارایی مطرح شده در صورت سؤال، کارایی زمان ترجمه سیستم باشد، آنگاه گزینه چهارم پاسخ سؤال خواهد بود. همانطور که گفتیم سازمان سنجش آموزش کشور، در کلید اولیه و نهایی خود، گزینه دوم را به عنوان پاسخ اعلام کرده بود.

۱۰- گزینه (۳) صحیح است.

یک راه حل کلی جهت مقابله با تکه تکه شدن خارجی این است که اجازه دهیم یک فرآیند در قسمت‌های همجوار در حافظه قرار گیرد. یکی از روش‌هایی که از این ایده استفاده می‌کند، تکنیک صفحه‌بندی است. در این روش حافظه به بخش‌هایی با اندازه‌ی یکسان به نام قاب (Frame) تقسیم می‌شود. از طرفی برنامه‌ها نیز به قسمت‌های مساوی و هم‌اندازه با قاب‌ها تقسیم می‌شوند که به آن‌ها صفحه (Page) می‌گویند. حال هنگامی که برنامه‌ای به حافظه منتقل می‌شود باید تمام صفحاتش به داخل قاب‌های خالی آورده شوند. در این حالت اصلاً نیازی نیست صفحات مربوط به یک فرآیند در قاب‌های همجوار قرار گیرند.

مزیت عمده این روش از بین بردن تکه تکه شدن خارجی و به حداقل رساندن تکه تکه شدن داخلی می‌باشد، اما در عوض عملیات محاسبه آدرس‌ها و مدیریت این صفحات قدری هزینه‌بر و زمان‌گیر است.

توجه: برای پیاده‌سازی این روش به پشتیبانی سخت‌افزار نیاز است.

توجه: این روش از دید کاربر و برنامه‌نویس مخفی می‌ماند.

توجه: برای پیاده‌سازی این روش و مدیریت صفحه‌ها و از همه مهم‌تر تبدیل و نگاشت آدرس‌ها باید یک جدول صفحه به ازای هر فرآیند در نظر گرفت. در واقع جدول صفحه‌ی هر فرآیند دارای یک درایه به ازای هر صفحه می‌باشد که مشخص می‌کند هر صفحه از یک فرآیند در کدام قاب حافظه نگهداری می‌شود.

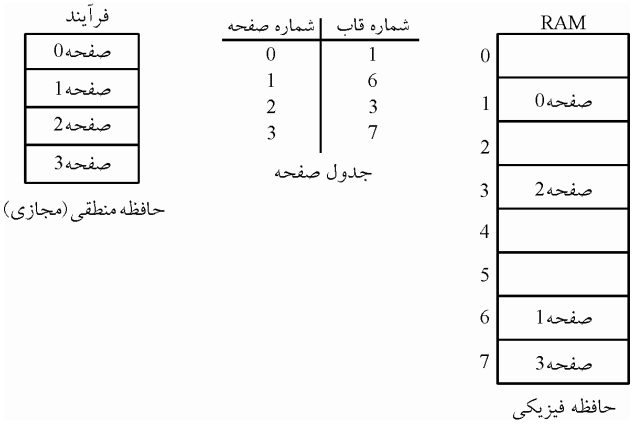
کارکرد اصلی جدول صفحه، تبدیل و نگاشت آدرس‌های مجازی به آدرس‌های فیزیکی می‌باشد. از نظر ریاضی جدول صفحه در واقع فقط یک تابع است که ورودی آن شماره صفحه مجازی و خروجی آن شماره قاب فیزیکی می‌باشد.

اما در این بین دو مسئله اساسی وجود دارد:

۱- نگاشت آدرس‌ها باید بسیار سریع صورت گیرد. هنگامی که جداول صفحه در حافظه اصلی باشند برای دسترسی به یک خانه حافظه، باید دو یا چند بار به حافظه سر زد. در اینصورت وقتی جداول به صورت چند سطحی پیاده‌سازی شوند، این تعداد بیشتر هم می‌شود و سرعت دسترسی به حافظه به شدت کاهش می‌یابد. در این حالت برای افزایش سرعت دسترسی به حافظه از تکنیک TLB استفاده می‌شود. TLB از حافظه‌های با سرعت بسیار بالا و گران قیمت ساخته شده است که در واقع مجموعه‌ای از رجیسترها موسوم به Associative Register هستند. وقتی فرآیندی جهت اجرا انتخاب شود، بخشی از درایه‌های جدول صفحه آن به TLB منتقل می‌شود. در این حالت وقتی CPU یک آدرس منطقی تولید می‌کند، شماره صفحه آن ابتدا در TLB جست‌وجو می‌شود، اگر شماره صفحه در TLB یافت شد که آدرس قاب متناظر به دست می‌آید، اما اگر شماره صفحه در TLB نباشد آن‌گاه طبق روال قبل به سراغ جدول صفحه در حافظه می‌رویم. نکته مهم در مورد TLB این است که عمل جست‌وجو در یک لحظه و همزمان در تمام سطرها صورت می‌گیرد. به

همین دلیل سرعت دسترسی به آن بسیار بالاست. این مساله برای کاربرنهـایی اهمیـت دارد. کـاربر نهایی زمان پاسخ کوتاه را در انجام کارها انتظار دارد.
۲- جدول صفحه می‌تواند بسیار بزرگ شود. سیستم صفحه‌بندی، با محدودیت اندازه قـاب مواجـه است، و اگر فرآیندی درون یک قاب جا نشود آنگاه باید در چند قـاب قـرار گیـرد. همچنـین ایـن روند برای جدول صفحه مربوط به یک فرآیند نیز وجود دارد و اگر جدول صفحه فرآینـدی درون یک قاب جا نشود آنگاه باید در چند قاب قرار گیرد. و همین مسئله منجر به ابداع جـداول صـفحه چند سطحی شد.
برای درک این ساختار، به مثال‌های زیر توجه کنید:

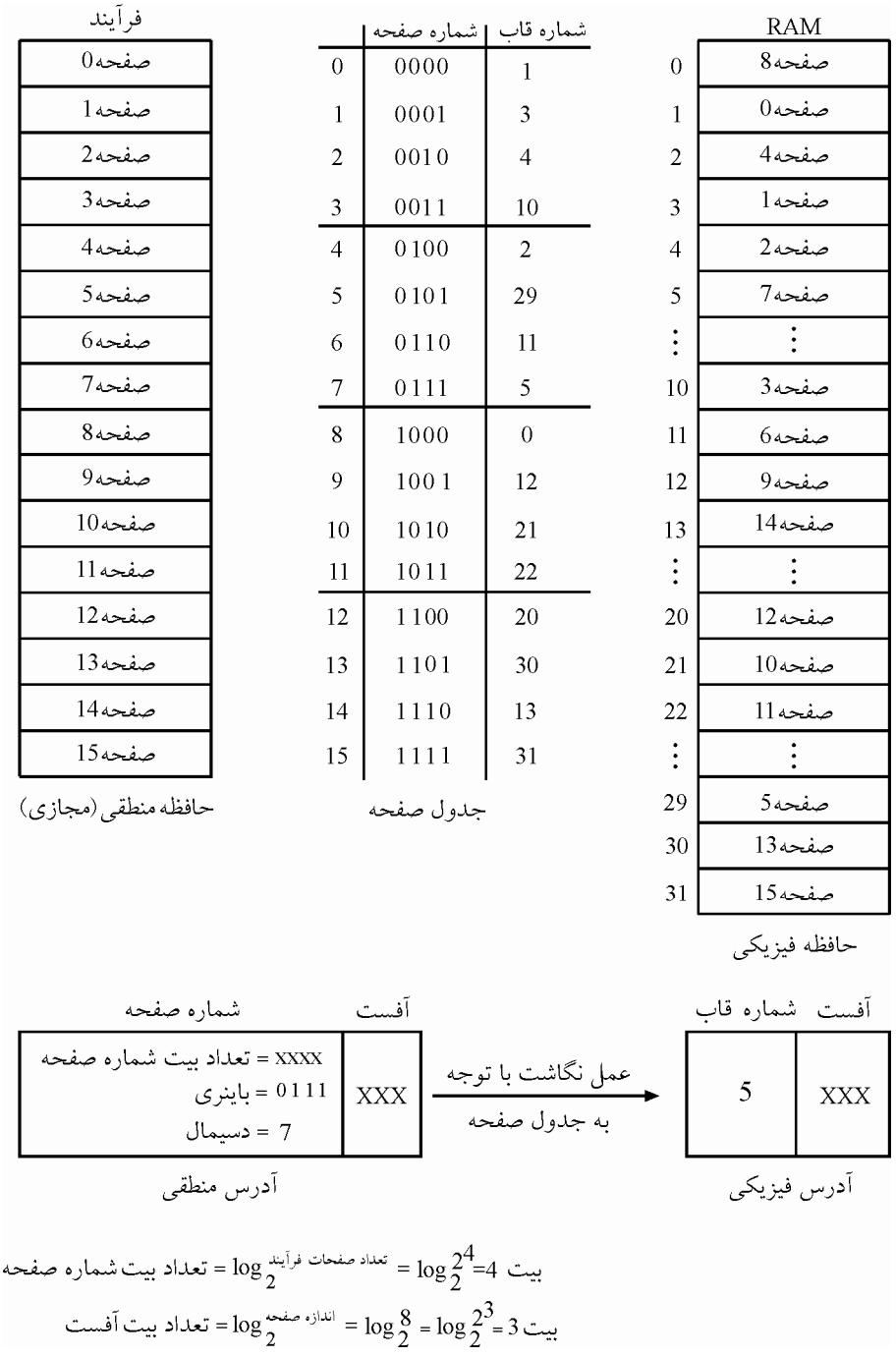
مثال اول: اگر فرآیندی که از چهار صفحه منطقی تشکیل شده است به صورت زیر در قاب‌های حافظه فیزیکی قرار گرفته باشد، آنگاه جدول صفحه به صورت زیر در خواهد آمد:



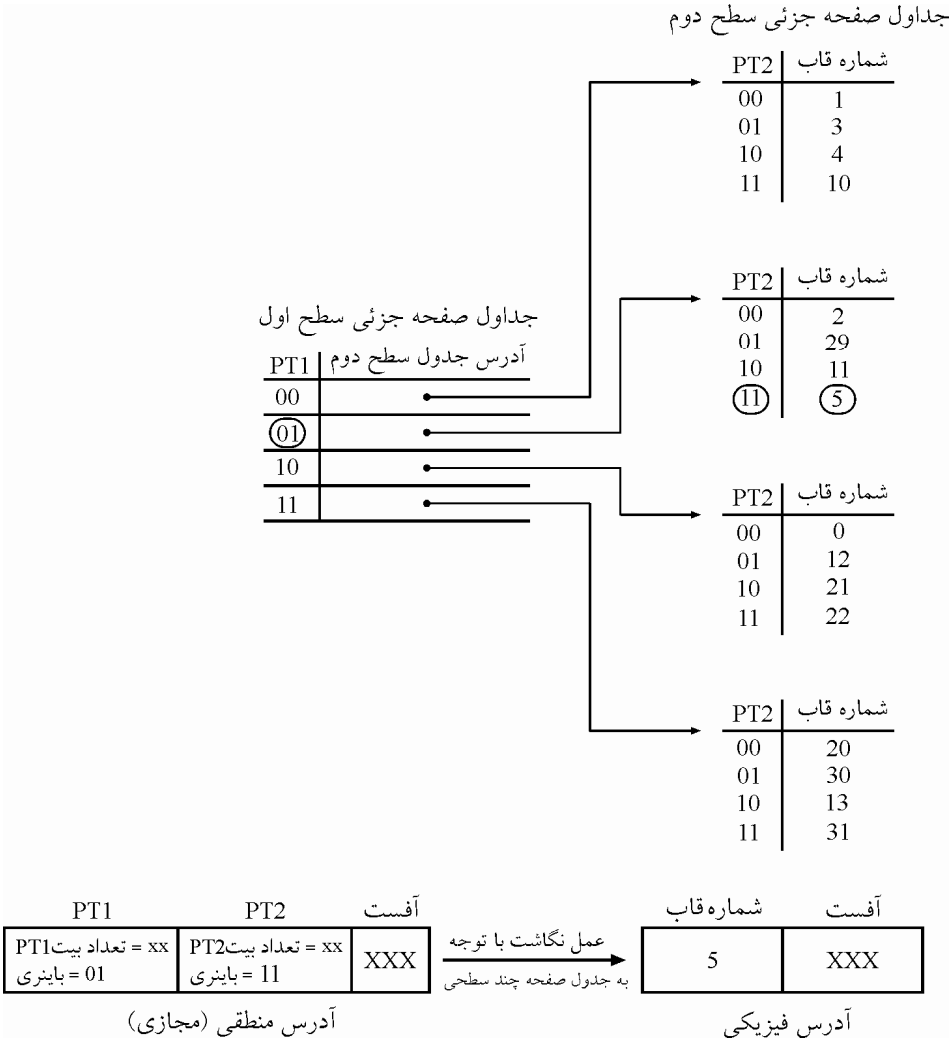
در راه‌حل فوق، از یک جدول صفحه تک سطحی، برای مدیریت صفحه‌بندی فرآیند استفاده شـده است. فرآیند فوق شامل چهار صفحه می‌باشد و واضح است که جدول صفحه نیز باید شامل چهار سطر باشد. برای مدیریت صفحه‌ها و نگاشت آدرس‌ها باید یک جدول صفحه به ازای هـر فرآینـد در نظر گرفت. در واقع جدول صفحه هر فرآیند دارای یک درایه به ازای هر صفحه مـی‌باشـد کـه مشخص می‌کند هر صفحه از یک فرآیند در کدام قاب حافظه نگهداری می‌شود.

مثال دوم: فرآیندی شامل 16 صفحه می‌باشد، اگر اندازه‌ی جـدول صـفحه دارای محـدودیت باشـد و حداکثر هر جدول صفحه بتواند دارای چهار سطر باشد، آنگاه جدول صفحه این فرآیند چنـد سـطحی خواهد بود؟ (اندازه‌ی صفحات 8 بایت است)

پاسخ: واضح است که اگر اندازه‌ی جدول صفحه دارای محدودیت نباشد، ایـن فرآینـد دارای یـک جدول صفحه تک سطحی با 16 سطر خواهد بود. شکل زیر جدول صفحه تک سطحی این فرآیند را نشان می‌دهد:


RAM
حافظه فیزیکی
شماره قاب
شماره صفحه
جدول صفحه
فرآیند
حافظه منطقی (مجازی)
آفست
شماره قاب
عمل نگاشت با توجه
به جدول صفحه
آدرس فیزیکی
شماره صفحه
آفست
= تعداد بیت شماره صفحه XXXX
= باینری 0111
= دسیمال 7
آدرس منطقی
بیت 4 = log2 24 = log2 تعداد صفحات فرآیند = تعداد بیت شماره صفحه
بیت 3 = log2 23 = log2 8 = log2 اندازه صفحه = تعداد بیت آفست

اما اگر اندازه‌ی جدول صفحه دارای محدودیت باشد و حداکثر بتواند دارای 4 سطر باشد، واضح است که در این حالت باید از راه‌حل جدول صفحه چند سطحی استفاده شود. بنابراین با 16 صفحه مواجه هستیم که هر 4 صفحه آن می‌تواند داخل یک جدول صفحه جزئی قرار بگیرد.
در شکل زیر واضح است که با توجه به شرایط مسأله یک جدول صفحه دو سطحی مسأله را حل می‌کند.



وقتی که یک آدرس مجازی به مدیر حافظه برسد، ابتدا فیلد PT1 این آدرس استخراج شده و به کمک آن درایه موردنظر در جدول سطح اول پیدا می‌شود. هر یک از درایه‌های این جدول، نماینده یک جدول سطح دوم می‌باشد. در واقع از مقادیر موجود در درایه‌های جدول سطح اول برای پیدا

کردن جدول سطح دوم مناسب استفاده می‌شود. اکنون از فیلد PT2 به عنوان اندیس جدول سطح دوم استفاده و به کمک آن، شماره فیزیکی قاب حافظه استخراج می‌شود و در نهایت بخش آفست نیز به این آدرس متصل می‌گردد.

توجه: در مثال فوق مشاهده می‌شود که برای دسترسی به هر قاب فقط به 2 جدول نیاز است. یکی جدول سطح اول و دیگری یکی از جداول سطح دوم. در واقع مزیت این روش از اینجا ناشی می‌شود که **فقط جداولی به حافظه آورده می‌شوند که مورد نیاز هستند.** دقت کنید این مفهوم با مفهوم حافظه مجازی متفاوت است. در مبحث حافظه مجازی مسأله این است که همه صفحات یک فرآیند به حافظه آورده نشوند. اما اینجا جداول صفحه به حافظه آورده نمی‌شوند.

توجه: به جداول موجود در سطوح مختلف جدول چند سطحی، **جداول جزئی** نیز گفته می‌شود. مجموع این جداول جزئی، جدول چند سطحی را ایجاد می‌کنند.

روابط زیر در جدول صفحه چند سطحی برقرار است:

f: تعداد صفحات فرآیند = 16

r: تعداد سطرهای جدول صفحه جزئی = 4

$$\text{تعداد جداول صفحه جزئی در سطح دوم} = \frac{\text{تعداد صفحات فرآیند}}{r} = \frac{f}{r} = \frac{16}{4} = 4$$

$$\text{تعداد جداول صفحه جزئی در سطح اول} = \frac{\text{تعداد جداول صفحه جزئی در سطح دوم}}{r} = \frac{4}{4} = 1$$

روال فوق گویای این مفهوم می‌باشد، که این تقسیم متوالی تا جایی ادامه پیدا می‌کند که خارج قسمت کوچکتر یا برابر یک شود. یعنی به یک جدول برسیم، به اندازه‌ی محدودیت.

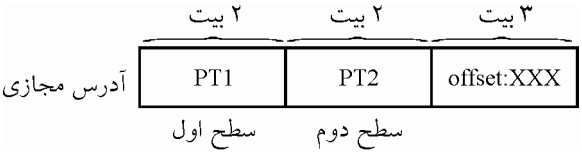
همچنین این نتیجه، تعداد سطوح جدول چند سطحی را نیز مشخص می‌کند. یعنی تعداد تقسیم از ابتدا تا به انتها با شرط مذکور.

توجه: در اینجا تعداد تقسیم متوالی برابر 2 است، بنابراین تعداد سطوح جدول چندسطحی برابر 2 است.

توجه: روال فوق را نیز می‌توان در مفهوم لگاریتم نیز جست‌وجو کرد، در واقع تقسیم فوق مفهوم لگاریتم را پیاده‌سازی می‌کند:

$$\text{تعداد سطوح جدول صفحه چند سطحی} = d = \lceil \log_r^f \rceil = \lceil \log_4^{16} \rceil = \left\lceil \log_{2^2}^{2^4} \right\rceil = 2$$

بنابراین در این مثال، آدرس‌های مجازی به سه بخش تقسیم می‌شوند:



اندازه فرآیند = اندازه صفحه × تعداد صفحات $=2^4 \times 2^3=2^7=128$ بایت

تعداد بیت آفست $= \log_2^{\text{اندازه صفحه}} = \log_2^8 = \log_2^{2^3} = 3$ بیت

$PT1 + PT2 = \log_2^{\text{تعداد صفحات فرآیند}} = \log_2^{2^4} = 4$

$PT2 = \log_2^r = \log_2^4 = \log_2^{2^2} = 2$

توجه: فرمول PT2 خیلی ساده است، جدول صفحه حداکثر می‌تواند چهار سطر داشته باشد، چهار سمبل داریم، برای آدرس‌دهی چهار سمبل به چند بیت نیاز داریم، واضح است که دو بیت، این همان مفهوم لگاریتم است.

$PT1 = (PT1+PT2) - PT2 = 2$ بیت

راه‌حل ساده‌تر (تجزیه): اندازه فرآیند را در اندازه r تجزیه کنید:

توجه: اندازه r، برابر تعداد سطرهای جداول صفحه است.

تعداد صفحات فرآیند $= 2^4 = 2^{\overset{\uparrow}{PT1}} \times 2^{\overset{\uparrow}{PT2}}$ (PT1 = ②، PT2 = ②)

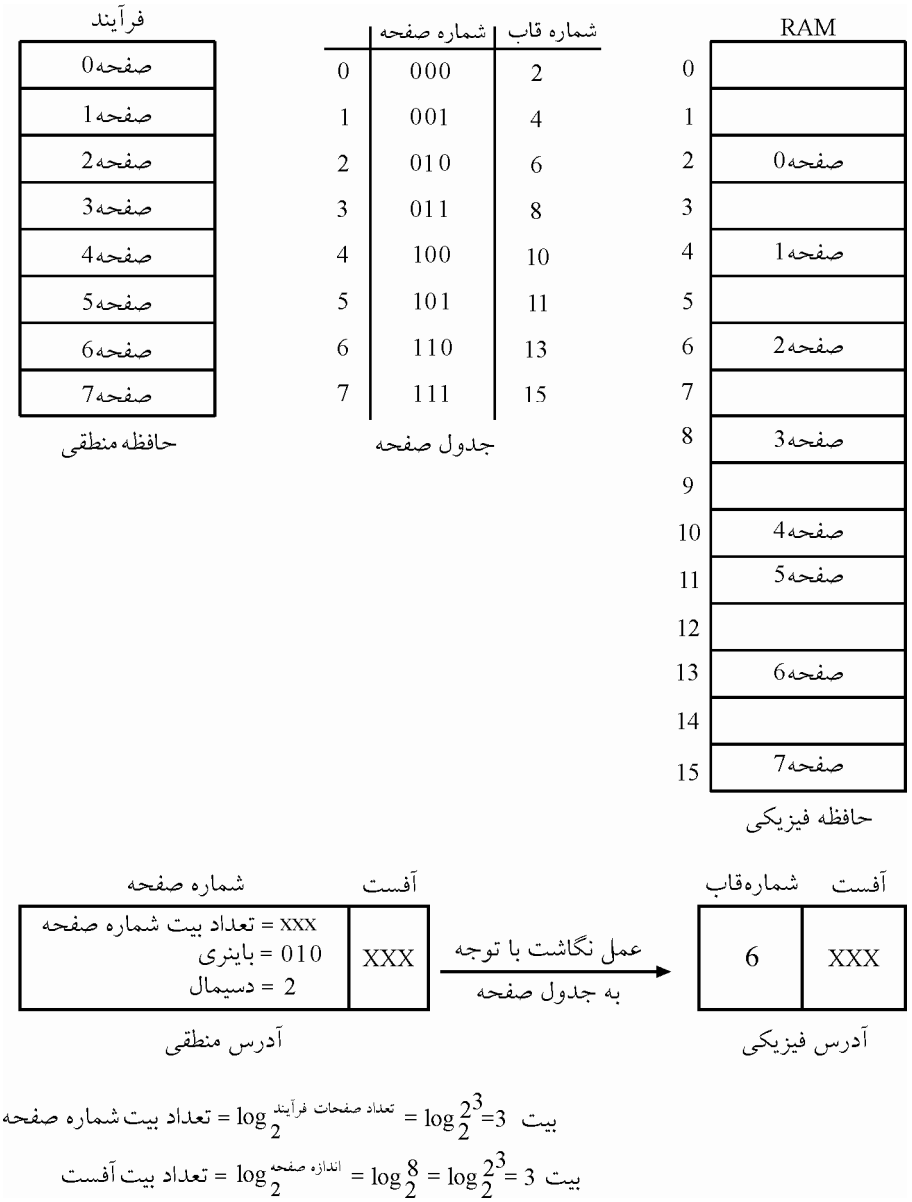
توجه: تجزیه را از راست به چپ و از اندیس n به 1 شروع کنید و کمترین مقدار در اندیس 1 قرار داده شود.

توجه: توان کمتر همواره PT1 است که در این مثال PT1 و PT2 برابر هستند.

توجه: تعداد عملوندها در رابطه بالا برابر تعداد سطوح جدول چند سطحی نیز می‌باشد، در رابطه فوق تعداد عملوندها برابر 2 است. بنابراین تعداد سطوح جدول چندسطحی نیز برابر 2 است، این نتیجه از آن‌جایی ناشی می‌شود که تجزیه فوق همان مفهوم لگاریتم را پیاده‌سازی می‌کند.

مثال سوم: فرآیندی شامل 8 صفحه می‌باشد، اگر اندازه‌ی جدول صفحه دارای محدودیت باشد و حداکثر هر جدول صفحه بتواند دارای 2 سطر باشد، آنگاه جدول صفحه این فرآیند چند سطحی خواهد بود؟ (اندازه صفحات 8 بایت است.)

پاسخ: واضح است که اگر اندازه‌ی جدول صفحه دارای محدودیت نباشد، این فرآیند دارای یک جدول صفحه تک‌سطحی با 8 سطر خواهد بود. شکل زیر جدول صفحه تک‌سطحی این فرآیند را نشان می‌دهد:

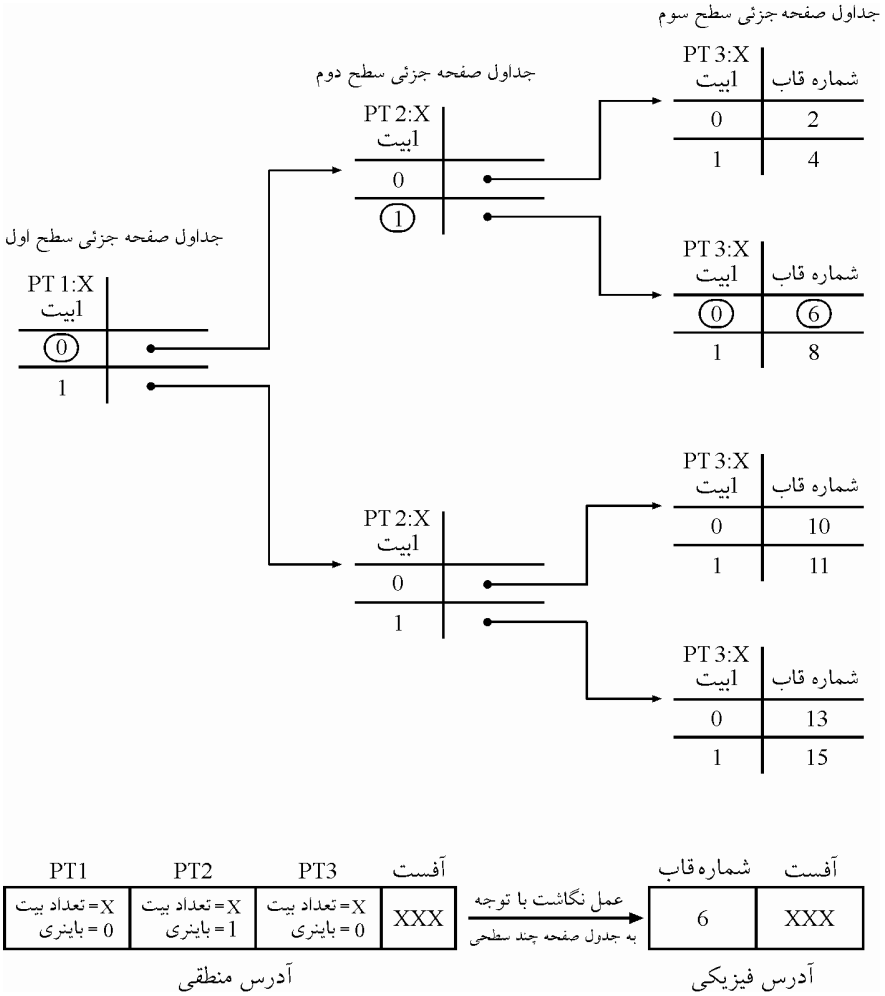


$$\text{تعداد بیت شماره صفحه} = \log_2^{\text{تعداد صفحات فرآیند}} = \log_2 \frac{2^3}{2} = 3 \text{ بیت}$$

$$\text{تعداد بیت آفست} = \log_2^{\text{اندازه صفحه}} = \log_2 \frac{8}{2} = \log_2 \frac{2^3}{2} = 3 \text{ بیت}$$

اما اگر اندازه جدول صفحه دارای محدودیت باشد و حداکثر بتوانـد دارای 2 سـطر باشـد، واضـح است که در این حالت باید از راه‌حل جدول صفحه چندسطحی استفاده شود. بنابراین با 8 صـفحه مواجه هستیم که هر 2 صفحه آن می‌تواند داخل یک جدول صفحه جزئی قرار بگیرد.

در شکل زیر واضح است که با توجه به شرایط مسأله یک جدول صفحه سه سطحی مسأله را حـل می‌کند.



روابط زیر در جدول صفحه چندسطحی برقرار است:

f: تعداد صفحات فرآیند = 8

r : تعداد سطرهای جدول سطحی جزئی = 2

روش تقسیم متوالی

$$\text{تعداد جداول صفحه جزئی در سطح سوم} = \frac{\text{تعداد صفحات فرآیند}}{r} = \frac{f}{r} = \frac{8}{2} = 4$$

$$\text{تعداد جداول صفحه جزئی در سطح دوم} = \frac{\text{تعداد جداول در سطح سوم}}{r} = \frac{4}{2} = 2$$

تعداد جداول صفحه جزئی در سطح اول = $\frac{\text{تعداد جداول در سطح دوم}}{r} = \frac{2}{2} = 1$

توجه: در اینجا تعداد تقسیم متوالی برابر 3 است، بنابراین تعداد سطوح جدول چندسطحی برابر 3 است.

روش لگاریتم

تعداد سطوح جدول صفحه چند سطحی = $d = \lceil \log_r^f \rceil = \lceil \log_2^8 \rceil = \left\lceil \log_2^{2^3} \right\rceil = 3$

روش تجزیه

تعداد صفحات فرآیند باید در اندازه r، تجزیه گردد.

تعداد صفحات فرآیند = $2^3 = 2^{\overset{\uparrow}{①}PT1} \times 2^{\overset{\uparrow}{①}PT2} \times 2^{\overset{\uparrow}{①}PT3}$

توجه: تعداد عملوندها برابر 3 است، بنابراین تعداد سطوح جدول چندسطحی نیز برابر 3 است.

توجه: 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 آن 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 پی‌درپی این دید را برای فرآیند ایجاد کرده است.

توجه: 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 هریک از آن‌ها بر روی دیسک است.

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

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 مابقی 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.

توجه: 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 اعتباری 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.

توجه: 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.

توجه: آدرسی که توسط فرآیند 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.

توجه: 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 شرط، چگونگی انتخاب یک صفحه برای ترک حافظه بسیار مهم است و تأثیر مستقیمی بر کارایی و تعداد وقفه‌های نقص صفحه در آینده دارد.

توجه: به جدول صفحه موجود در سطح اول از یک ساختار چندسطحی، جدول صفحه **ریشه** یا جدول صفحه بیرونی گفته می‌شود. جدول صفحه ریشه در حافظه اصلی و درون یکی از قاب‌های حافظه اصلی قرار دارد. صفحه مربوط به جدول صفحه ریشه هرگز نباید از حافظه اصلی خارج شود، مگر به طور کامل مورد استفاده نباشد. به بیان دیگر جدول صفحه ریشه، اغلب در حافظه اصلی قرار دارد و جداول صفحه بعدی (جداول صفحه جزئی) در صورت نیاز به حافظه اصلی منتقل می‌گردند و تا زمانیکه مورد نیاز و رجوع هستند نباید از حافظه اصلی خارج گردند، در غیر اینصورت در دیسک قرار می‌گیرند. برای یک فرآیند، جدول صفحه ریشه اغلب داخل حافظه اصلی است، اما سایر جداول جزئی در سطوح بعدی، در صورت نیاز و عدم وجود در حافظه اصلی پس از یک نقص صفحه حاصل از عدم وجود صفحه مربوط به جدول صفحه جزئی مورد نیاز به حافظه اصلی بارگذاری می‌گردند. دقت کنید که این نقص صفحه حاصل از عدم وجود صفحه مربوط به جدول صفحه جزئی مورد نیاز است. و نه نقص صفحه مربوط به صفحات یک

فرآیند. در روند تبدیل آدرس مجازی به فیزیکی پس از جور شدن همه جداول صفحه جزئی مورد نیاز و آوردن آنها به حافظه اصلی، ممکن است در آخرین سطح از ساختار چند سطحی، جلوی آدرس مجازی مورد نظر مقداری برای قاب آن نباشد، که این رویداد منجر به ایجاد نقص صفحه مربوط به صفحات فرآیند می‌گردد که مطابق روال مربوط اینبار باید صفحه مورد نظر در حافظه اصلی بارگذاری گردد. برای نمونه، در مثال دوم که به صورت جدول صفحه دو سطحی است، فرض کنید جلوی آدرس مجازی 7، شماره قاب 5 نباشد و به جای مقدار 5 مقدار NULL باشد، در اینصورت هرچند جداول صفحه جزئی مورد نیاز درون حافظه اصلی قرار دارند، اما اینبار صفحه مربوط به صفحه مورد نیاز یک فرآیند درون حافظه اصلی قرار ندارد، که این مساله منجر به نقص صفحه مربوط به صفحه مورد نظر یک فرآیند می‌گردد. اما اگر در روند تبدیل آدرس مجازی به فیزیکی پس از جور شدن همه جداول صفحه جزئی مورد نیاز و آوردن آنها به حافظه اصلی، در آخرین سطح از ساختار چند سطحی، جلوی آدرس مجازی مورد نظر مقداری برای قاب آن باشد، آنگاه این مقدار در سمت چپ مقدار آفست قرار می‌گیرد و آدرس فیزیکی خلق می‌گردد.
توجه: در طرح جدول‌های صفحه چندسطحی، چون ترجمه آدرس از جدول ریشه(بیرونی) شروع شده و به طرف سطوح بعدی حرکت می‌کند، این طرح به جدول صفحه نگاشت **پیشرو** نیز معروف است.

در صورت سوال مطرح شده است که در یک ساختار حافظه، سیستم صفحه‌بندی سه سطحی از آدرس مجازی مطابق فرمت زیر استفاده می‌کند، همچنین مطرح شده است که اگر نرخ برخورد هر سطح 90% باشد، احتمال برخورد تبدیل یک آدرس مجازی به فیزیکی چقدر است؟ البته با فرض اینکه احتمال برخورد هر سطح مستقل از احتمال برخورد در سطوح دیگر باشد.

P3	P2	P1	d

در روند تبدیل آدرس مجازی به فیزیکی، نرخ برخورد 90% در سطح اول، یعنی در 90% موارد جدول صفحه جزئی سطح دوم مورد نیاز درون حافظه اصلی قرار دارد یعنی در 90% موارد نقص صفحه حاصل از عدم وجود صفحه مربوط به جدول صفحه جزئی سطح دوم مورد نیاز رخ نمی‌دهد.

در روند تبدیل آدرس مجازی به فیزیکی، نرخ برخورد 90% در سطح دوم، یعنی در 90% موارد جدول صفحه جزئی سطح سوم مورد نیاز درون حافظه اصلی قرار دارد یعنی در 90% موارد نقص صفحه حاصل از عدم وجود صفحه مربوط به جدول صفحه جزئی سطح سوم مورد نیاز رخ نمی‌دهد.

در روند تبدیل آدرس مجازی به فیزیکی، نرخ برخورد 90% در سطح سوم، یعنی در 90% موارد صفحات مورد نیاز مربوط به یک فرآیند درون حافظه اصلی قرار دارد و برای آنها مقدار قاب نیز مشخص شده است. یعنی در 90% موارد نقص صفحه حاصل از عدم وجود صفحات مورد نیاز مربوط به یک فرآیند رخ نمی‌دهد.

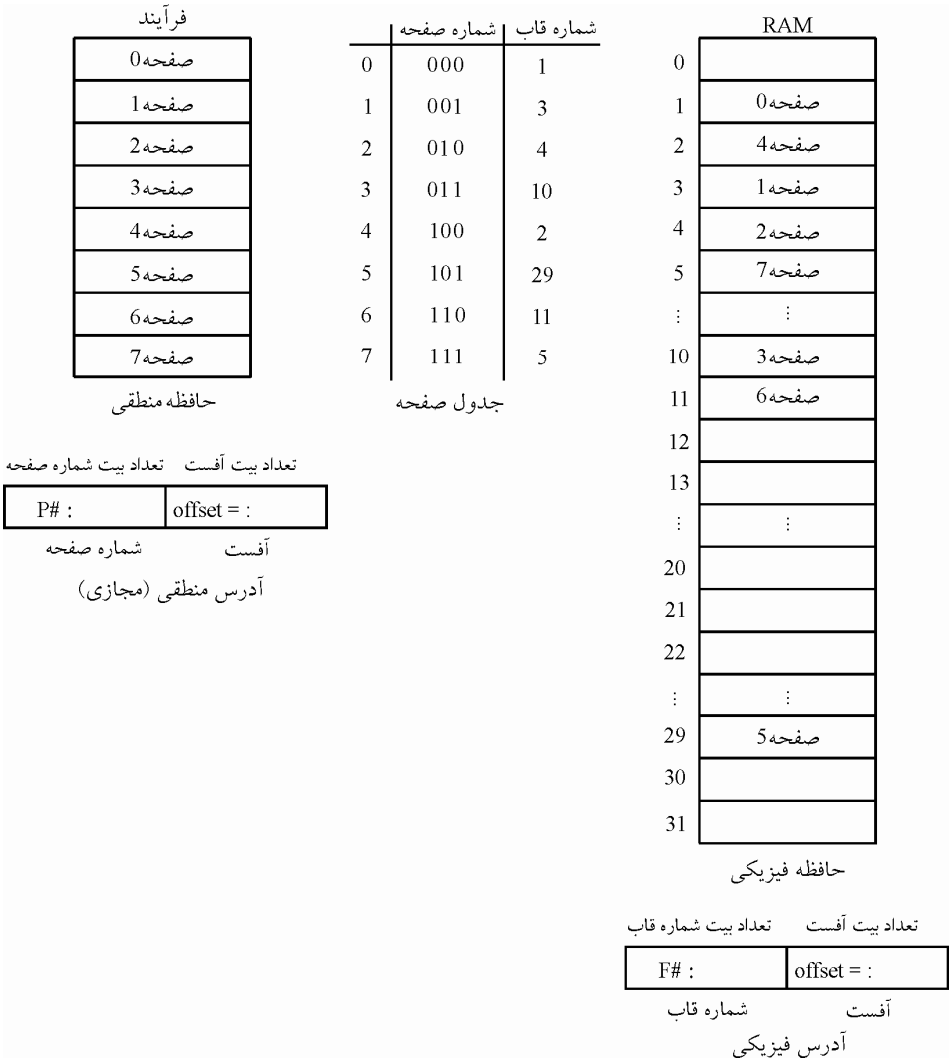
$0.9 \times 0.9 \times 0.9 = 0.729$ = احتمال برخورد تبدیل یک آدرس مجازی به فیزیکی

بنابراین گزینه سوم پاسخ سوال خواهد بود.

۱۱- گزینه (۳) صحیح است.

در صورت سوال مطرح شده است که اندازه فضای آدرس منطقی، 8 صفحه 1024 بایتی و اندازه حافظه فیزیکی 32 قاب است. همچنین خواسته شده است طول آدرس منطقی و آدرس فیزیکی محاسبه گردد.

به طور کلی روابط میان آدرس منطقی و آدرس فیزیکی در راه حل صفحه‌بندی به صورت زیر است:



$$\text{تعداد صفحات فرآیند (تعداد درایه‌های جدول صفحه)} = \frac{\text{اندازه فرآیند}}{\text{اندازه صفحه یا اندازه قاب}} = 8$$

$$\text{تعداد قاب‌های حافظه فیزیکی} = \frac{\text{اندازه حافظه فیزیکی}}{\text{اندازه صفحه یا اندازه قاب}} = 32$$

$$\text{تعداد بیت شماره صفحه} = b = \log_2^{\text{تعداد صفحات فرآیند}} = \log_2^8 = 3bit$$

$$\text{تعداد بیت شماره قاب} = b = \log_2^{\text{تعداد قاب‌های حافظه فیزیکی}} = \log_2^{32} = 5bit$$

$$\text{تعداد بیت آفست} = b = \log_2^{\text{اندازه صفحه یا اندازه قاب}} = \log_2^{1024} = 10bit$$

عرض جدول صفحه × تعداد صفحات فرآیند (تعداد درایه‌های جدول صفحه) = اندازه جدول صفحه

توجه: عرض جدول صفحه برابر حاصل جمع تعداد بیت‌های کنترلی و تعداد بیت‌های شماره قاب می‌باشد، دقت کنید که تعداد بیت‌های شماره صفحه جزو عرض جدول صفحه نمی‌باشد، بلکه شماره صفحه، اندیس هر سطر جدول صفحه می‌باشد، به صورت زیر:

تعداد بیت‌های کنترلی + تعداد بیت‌های شماره قاب = عرض جدول صفحه

در سوال مطرح شده 0 بیت مربوط به بیت‌های کنترلی و 5 بیت مربوط به تعداد بیت‌های شماره قاب می‌باشد.

پس: عرض جدول صفحه فوق برابر $5bit + 0bit = 5bit$ می‌باشد.

همانطور که گفتیم، اندازه جدول صفحه، از رابطه زیر محاسبه می‌گردد:

عرض جدول صفحه × تعداد صفحات فرآیند (تعداد درایه‌های جدول صفحه) = اندازه جدول صفحه

که مطابق رابطه فوق داریم:

$$\text{اندازه جدول صفحه} = 8 \times 5bit = 40bit = 5Byte$$

همچنین، اندازه آدرس‌های منطقی (مجازی) و فیزیکی به صورت زیر است:

$$\text{طول آدرس منطقی} = \text{تعداد بیت شماره صفحه} + \text{تعداد بیت آفست} = 3bit + 10bit = 13bit$$

$$\text{طول آدرس فیزیکی} = \text{تعداد بیت شماره قاب} + \text{تعداد بیت آفست} = 5bit + 10bit = 15bit$$

همچنین داریم:

$$\text{اندازه حافظه منطقی (فرآیند)} = 2^{\text{تعداد بیت شماره صفحه}} \times 2^{\text{تعداد بیت آفست}} = 2^{\text{تعداد بیت آدرس منطقی}}$$

$$\text{اندازه حافظه منطقی (فرآیند)} = 2^{13} = 2^3 \times 2^{10} = 2^{13} = 8KB$$

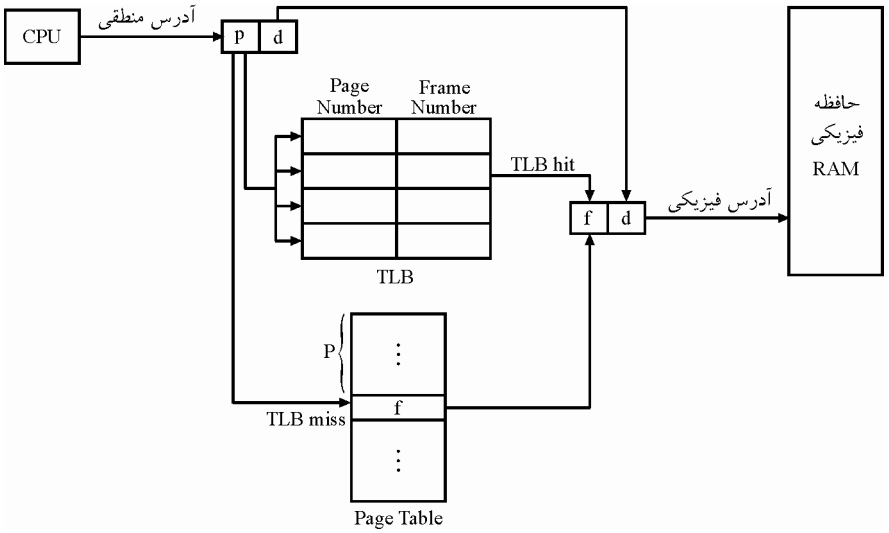
$$\text{اندازه حافظه فیزیکی (RAM)} = 2^{\text{تعداد بیت شماره قاب}} \times 2^{\text{تعداد بیت آفست}} = 2^{\text{تعداد بیت آدرس فیزیکی}}$$

$$\text{اندازه حافظه فیزیکی (RAM)} = 2^{15} = 2^5 \times 2^{10} = 2^{15} = 32\ KB$$

توجه: بنابراین اندازه آدرس منطقی (مجازی) و فیزیکی (حقیقی) الزاماً برابر نیست. بنابراین گزینه سوم پاسخ سوال خواهد بود.

۱۲- گزینه (۴) صحیح است.

بر اساس صورت سوال، در **گزینه اول** گفته شده‌است که اگر تعداد خانه‌های TLB بیشتر شود نرخ برخورد در TLB بیشتر می‌شود. این عبارت درست است. هنگامی که جداول صفحه در حافظه اصلی باشند برای دسترسی به یک خانه حافظه، باید دو یا چند بار به حافظه سر زد. در اینصورت وقتی جداول به صورت چند سطحی پیاده‌سازی شوند، این تعداد بیشتر هم می‌شود و سرعت دسترسی به حافظه به شدت کاهش می‌یابد. در این حالت برای افزایش سرعت دسترسی به حافظه از تکنیک TLB استفاده می‌شود. TLB از حافظه‌های با سرعت بسیار بالا و گران قیمت ساخته شده است که در واقع مجموعه‌ای از رجیسترها موسوم به Associative Register هستند. وقتی فرآیندی جهت اجرا انتخاب شود، بخشی از درایه‌های جدول صفحه آن به TLB منتقل می‌شود. در این حالت وقتی CPU یک آدرس منطقی تولید می‌کند، شماره صفحه آن ابتدا در TLB جست‌وجو می‌شود، اگر شماره صفحه در TLB یافت شد که آدرس قاب متناظر به دست می‌آید، اما اگر شماره صفحه در TLB نباشد آن‌گاه طبق روال قبل به سراغ جدول صفحه در حافظه می‌رویم. نکته مهم در مورد TLB این است که عمل جست‌وجو در یک لحظه و همزمان در تمام سطرها صورت می‌گیرد. به همین دلیل سرعت دسترسی به آن بسیار بالاست. این مساله برای کاربرنهایی اهمیت دارد. کاربر نهایی زمان پاسخ کوتاه را در انجام کارها انتظار دارد.



توجه: با توجه به ساختار TLB و گران بودن آن، فقط بخش کوچکی از جدول صفحه این شانس را پیدا می‌کند که به TLB وارد شود. به این ترتیب اگر هنگام تبدیل آدرس، شماره صفحه در

TLB موجود باشد اصطلاحاً HIT و در غیر این صورت MISS رخ داده است. «**بنابراین اگر تعداد خانه‌های TLB بیشتر شود، این رویداد منجر به این می‌شود که شماره صفحات بیشتری از یک فرآیند درون TLB یافت شود که این یعنی افزایش نرخ برخورد.**» پس «**اگر تعداد خانه‌های TLB بیشتر شود نرخ برخورد در TLB بیشتر می‌شود.**»

همچنین ضریب موفقیت یا نسبت اصابت به صورت زیر محاسبه می‌شود:

$$\text{HIT Ratio} = \frac{\text{HIT}}{\text{HIT} + \text{MISS}}$$

مثال: فرض کنید در یک سیستم، زمان دسترسی به حافظه 60 نانو ثانیه و زمان دسترسی به TLB، برابر 5 نانو ثانیه باشد. اگر احتمال وجود شماره صفحه در TLB برابر 80% باشد، زمان دسترسی به حافظه چقدر است؟

پاسخ: هنگامی که یک درخواست برای حافظه از راه می‌رسد، دو حالت ممکن است رخ دهد:

در حالت اول، شماره صفحه در TLB موجود است، که در این صورت یک بار در TLB و یک بار به حافظه رجوع می‌کنیم، یعنی جمعاً 60+5=65ns.

اما در حالت دوم، شماره صفحه در TLB پیدا نمی‌شود، در این صورت یک بار به TLB و دو بار به حافظه رجوع می‌کنیم (یک بار برای رجوع به جدول صفحه و پیدا کردن شماره صفحه و یک بار هم برای دستیابی به داده مورد نظر)، یعنی جمعاً 60+60+5=125ns.

لذا زمان کل دسترسی مؤثر به حافظه با توجه به ضریب موفقیت برابر است با:

$$0.8 \times 65 + 0.2 \times 125 = 77\text{ns}$$

بر اساس صورت سوال، در **گزینه دوم** گفته شده‌است که اگر اندازه صفحه‌ها بزرگتر شود نرخ برخورد در TLB بیشتر می‌شود. این عبارت درست است. اگر اندازه صفحه‌ها بزرگتر شود، تعداد صفحات و در نتیجه تعداد درایه‌های جدول صفحه کمتر می‌شود. یعنی فضای نمونه کمتر شده و در نتیجه احتمال و نرخ برخورد در TLB بیشتر می‌شود. وقتی فرآیندی جهت اجرا انتخاب می‌شود، بخشی از درایه‌های جدول صفحه آن به TLB منتقل می‌شود. حال اگر تعداد صفحات و تعداد درایه‌های جدول صفحه به دلیل افزایش اندازه صفحات، کمتر و کمتر شود، آنگاه تعداد درایه‌های بیشتر و بیشتری از جدول صفحه این شانس را خواهند داشت تا وارد فضای محدود TLB شوند و در بهترین حالت شاید همه وارد TLB شوند و این یعنی افزایش نرخ برخورد در TLB. پس «**اگر اندازه صفحه‌ها بزرگتر شود نرخ برخورد در TLB بیشتر می‌شود.**»

بر اساس صورت سوال، در **گزینه سوم** گفته شده‌است که برای صفحه‌های با اندازه ثابت، اگر طول آدرس فیزیکی بیشتر شود نرخ برخورد در TLB تغییر نمی‌کند. این عبارت درست است. نرخ برخورد در TLB تابعی از تعداد خانه‌های TLB و تعداد صفحات و در نتیجه تعداد درایه‌های

جدول صفحه است. یعنی مطابق آنچه در گزینه‌های اول و دوم گفتیم اگر تعداد خانه‌های TLB بیشتر شود نرخ برخورد در TLB بیشتر می‌شود. و اگر اندازه صفحه‌ها بزرگتر شود نرخ برخورد در TLB بیشتر می‌شود. افزایش یا کاهش نرخ برخورد در TLB رابطه‌ای با افزایش یا کاهش طول آدرس فیزیکی ندارد. و اصلا به هم مرتبط نیستند، پس «**برای صفحه‌های با اندازه ثابت، اگر طول آدرس فیزیکی بیشتر شود نرخ برخورد در TLB تغییر نمی‌کند.**»

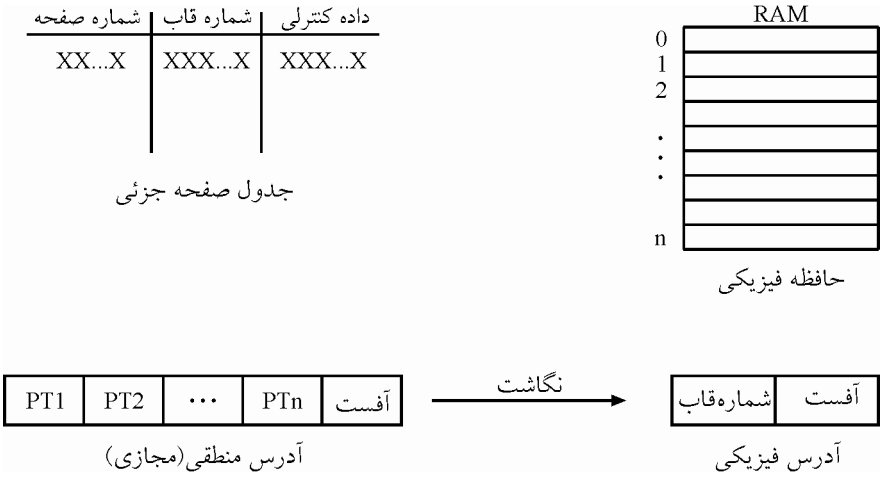
بر اساس صورت سوال، در **گزینه چهارم** گفته شده‌است که برای صفحه‌های با اندازه ثابت، با افزایش اندازه قطعه (segment)، نرخ برخورد در TLB بیشتر می‌شود. این عبارت نادرست است. در راه حل قطعه‌بندی همراه با صفحه‌بندی، قطعات تولید شده توسط برنامه‌نویس یا کامپایلر، توسط سیستم عامل صفحه‌بندی می‌شود. در این حالت به ازای هر فرآیند یک جدول قطعه ایجاد می‌شود که تعداد درایه‌های جدول قطعه، بستگی به اندازه فرآیند و تعداد قطعات حاصل دارد. همچنین به ازای هر قطعه (segment)، یک جدول صفحه ایجاد می‌شود که تعداد درایه‌های جدول صفحه، بستگی به اندازه قطعه و تعداد صفحات حاصل دارد. جهت افزایش سرعت نگاشت آدرس منطقی به فیزیکی می‌توان از TLB نیز استفاده نمود. TLB از حافظه‌های با سرعت بسیار بالا و گران قیمت ساخته شده است که در واقع مجموعه‌ای از رجیسترها موسوم به Associative Register هستند. در سیستم قطعه‌بندی همراه با صفحه‌بندی، وقتی فرآیندی جهت اجرا انتخاب شود، بخشی از درایه‌های جدول قطعه و جدول صفحه آن به TLB منتقل می‌شود. در این حالت وقتی CPU یک آدرس منطقی تولید می‌کند، شماره قطعه و شماره صفحه آن ابتدا در TLB جست‌وجو می‌شود، اگر شماره قطعه و شماره صفحه در TLB یافت شد که آدرس قاب متناظر در حافظه اصلی به دست می‌آید، اما اگر شماره قطعه و شماره صفحه در TLB نباشد، آن‌گاه طبق روال معمول به سراغ جدول قطعه و جدول صفحه در حافظه اصلی می‌رویم. نکته مهم در مورد TLB این است که عمل جست‌وجو در یک لحظه و همزمان در تمام سطرها صورت می‌گیرد. به همین دلیل سرعت دسترسی به آن بسیار بالاست. این مساله برای کاربرنهایی اهمیت دارد. کاربر نهایی زمان پاسخ کوتاه را در انجام کارها انتظار دارد. همانطور که گفتیم به ازای هر قطعه (segment)، یک جدول صفحه ایجاد می‌شود که تعداد درایه‌های جدول صفحه، بستگی به اندازه قطعه و تعداد صفحات حاصل دارد. اگر اندازه قطعه (segment)، بزرگتر شود تعداد صفحات و در نتیجه تعداد درایه‌های جدول صفحه مربوط به قطعه مورد نظر بیشتر می‌شود. یعنی فضای نمونه بیشتر شده و در نتیجه احتمال و نرخ برخورد در TLB کمتر می‌شود. وقتی فرآیندی جهت اجرا انتخاب شود، بخشی از درایه‌های جدول قطعه و جدول صفحه آن به TLB منتقل می‌شود. حال اگر تعداد صفحات و تعداد درایه‌های جدول صفحه قطعه مورد نظر به دلیل افزایش اندازه قطعات، بیشتر و بیشتر شود، آنگاه تعداد درایه‌های کمتری از جدول صفحه قطعه مورد نظر این شانس را خواهند داشت تا وارد فضای محدود TLB شوند و این یعنی کاهش نرخ برخورد در

TLB. پس «**برای صفحه‌های با اندازه ثابت، با افزایش اندازه قطعه (segment)، نرخ برخورد در TLB کمتر می‌شود و نه بیشتر.**»

۱۳- گزینه (۱) صحیح است.

در اینجا برای جدول صفحه جزئی محدودیتی به اندازه یک قاب (صفحه) داریم.

بنابراین اندازه جدول صفحه جزئی برابر اندازه قاب (صفحه) می‌باشد. بنابراین برای محاسبه تعداد سطرهای جدول صفحه جزئی، کافی است، اندازه قاب که برابر اندازه جدول صفحه جزئی است بر اندازه عرض جدول صفحه جزئی تقسیم گردد. به شکل زیر توجه کنید:



توجه: عرض جدول صفحه همواره برابر حاصل جمع تعداد بیت‌های کنترلی و تعداد بیت‌های شماره قاب است، دقت کنید که تعداد بیت‌های شماره صفحه جزو عرض جدول صفحه نمی‌باشد، بلکه شماره صفحه، اندیس هر سطر جدول صفحه می‌باشد.

بنابراین داریم:

تعداد بیت‌های کنترلی + تعداد بیت‌های شماره قاب= عرض جدول صفحه جزئی

$4B$ = عرض جدول صفحه جزئی

توجه: مطابق فرض سؤال، هر مدخل جدول صفحه (عرض جدول صفحه جزئی) 4 بایت در نظر گرفته شده است.

$$\text{تعداد سطرهای جدول صفحه جزئی} = \frac{\text{اندازه قاب}}{\text{عرض جدول صفحه}} = \frac{2^3 \times 2^{10}B}{2^2 B} = 2^{11} = 2048$$

$2^{46}B$ =اندازه فرآیند (فضای آدرس مجازی)

$$\text{اندازه صفحه = اندازه قاب} = 8KB = 8192B = 2^3 \times 2^{10}B = 2^{13}B$$

$$\text{f : تعداد صفحات فرآیند} = \frac{\text{اندازه فرآیند}}{\text{اندازه صفحه}} = \frac{2^{46}}{2^{13}} = 2^{33}$$

$$\text{r : تعداد سطرهای جدول صفحه جزئی} = 2048 = 2^{11}$$

حال اطلاعات کافی برای محاسبه تعداد سطوح جدول صفحه چند سطحی را در اختیار داریم:

روش تجزیه

تعداد صفحات فرآیند باید در اندازه r(2^{11}) تجزیه گردد:

$$\text{تعداد صفحات فرآیند} = 2^{33} = 2^{\overset{\uparrow}{\textcircled{11}}\text{PT1}} \times 2^{\overset{\uparrow}{\textcircled{11}}\text{PT2}} \times 2^{\overset{\uparrow}{\textcircled{11}}\text{PT3}}$$

توجه: تعداد عملوندها برابر 3 است، بنابراین تعداد سطوح جدول چند سطحی نیز برابر 3 است.

روش لگاریتم

$$\text{تعداد سطوح جدول چند سطحی} = d = \left\lceil \log_r^f \right\rceil = \left\lceil \log_{2^{11}}^{2^{33}} \right\rceil = 3$$

روش تقسیم متوالی

$$\text{تعداد جداول صفحه جزئی در سطح سوم} = \frac{\text{تعداد صفحات فرآیند}}{r} = \frac{f}{r} = \frac{2^{33}}{2^{11}} = 2^{22}$$

$$\text{تعداد جداول صفحه جزئی در سطح دوم} = \frac{\text{تعداد جداول صفحه جزئی در سطح سوم}}{r} = \frac{2^{22}}{2^{11}} = 2^{11}$$

$$\text{تعداد جداول صفحه جزئی در سطح اول} = \frac{\text{تعداد جداول صفحه جزئی در سطح دوم}}{r} = \frac{2^{11}}{2^{11}} = 1$$

توجه: سطح اول، یک جدول به حساب می‌آید، که 2^{11} سطر دارد.

توجه: تعداد تقسیم متوالی برابر 3 است، بنابراین تعداد سطوح جدول صفحه چند سطحی برابر 3 است.

$$\text{تعداد بیت آفست} = \log_2^{\text{اندازه صفحه}} = \log_2^{2^{13}} = 13 \text{ بیت}$$

بنابراین شکل آدرس منطقی (مجازی) به صورت زیر خواهد بود:

PT1	PT2	PT3	آفست
11 بیت	11 بیت	11 بیت	13 بیت
33 بیت			

مطابق آنچه گفتیم سه دسترسی به جداول صفحه جزئی سطح اول، دوم و سوم برای ترجمه آدرس و یک دسترسی به داده اصلی (مقصد مورد نظر) لازم است، که مجموع آن شامل چهار دسترسی به حافظه می‌گردد. بنابراین برای خواندن یک کلمه 32 بیتی نیاز به چهار دسترسی به حافظه است.

۱۴- گزینه (۱) صحیح است.

در تکنیک صفحه‌بندی، برای هر فرآیند یک جدول صفحه 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 مقرون به صرفه نیست.

برای حل این مشکل از جداول صفحه معکوس استفاده می‌کنیم. در این حالت 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.

با استفاده از تکنیک جدول صفحه معکوس، مقدار زیادی در حافظه صرفه‌جویی می‌شود اما تبدیل آدرس مجازی به فیزیکی سخت‌تر و زمان‌گیرتر می‌شود.

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 وارونه را برای یافتن درایه (PID, P#) جستجو کند.

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:

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

بنابراین گزینه اول پاسخ سوال است.

همچنین اندازه حافظه منطقی (فرآیند) مطابق رابطه زیر قابل محاسبه است:

$$\text{اندازه حافظه منطقی (فرآیند)} = 2^{\text{تعداد بیت آدرس منطقی}} = 2^{\text{تعداد بیت شماره صفحه}} \times 2^{\text{تعداد بیت آفست}}$$

$$\text{اندازه حافظه منطقی (فرآیند)} = 2^{34} = 2^{23} \times 2^{11} = 2^{34}\ \text{Byte} = 2^4 \times 2^{30} = 16\ \text{GB}$$

۱۵- گزینه ( ) صحیح است.

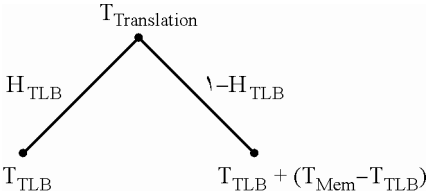
مطابق فرض سوال دو سیستم S_1 و S_2 همزمان با ارسال درخواست به TLB، آنرا به حافظه اصلی نیز ارسال می‌کنند تا در صورت عدم یافتن آدرس در TLB، زمان پاسخ کاهش یابد. به عبارت دیگر نحوه عملکرد دو سیستم S_1 و S_2 **موازی** در نظرگرفته شده‌است، به صورت زیر:

$\mathbf{T_{Translation}}$: میانگین زمان ترجمه در حالت **موازی**

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

برای محاسبه $T_{Translation}$ میانگین T_{TLB} 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.

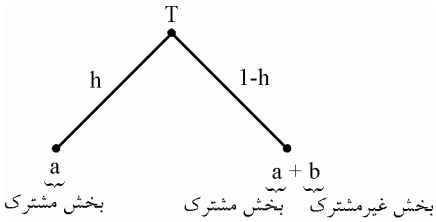
$$T_{Translation} = T_1 = T_{TLB} + (1\text{-}H_{TLB}) \times (T_{Mem} - T_{TLB})$$

$$T_{Translation} = T_1 = 120 + (1\text{-}0.6) \times (T_{Mem} - 120) = 120 + 0.4 \times T_{Mem} - 48 = 72 + 0.4 \times T_{Mem}$$

$$T_{Translation} = T_2 = T_{TLB} + (1\text{-}H_{TLB}) \times (T_{Mem} - T_{TLB})$$

$$T_{Translation} = T_2 = 150 + (1\text{-}0.8) \times (T_{Mem} - 150) = 150 + 0.2 \times T_{Mem} - 30 = 120 + 0.2 \times T_{Mem}$$

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 $T_{Translation}$ خواهیم داشت:

$T_{Translation} = T_1 = T_{TLB} + (1\text{-}H_{TLB}) \times (T_{Mem} - T_{TLB})$

$T_{Translation} = T_1 = 120 + (1\text{-}0.6) \times (T_{Mem} - 120) = 120 + 0.4 \times T_{Mem} - 48 = 72 + 0.4 \times T_{Mem}$

$T_{Translation} = T_2 = T_{TLB} + (1\text{-}H_{TLB}) \times (T_{Mem} - T_{TLB})$

$T_{Translation} = T_2 = 150 + (1\text{-}0.8) \times (T_{Mem} - 150) = 150 + 0.2 \times T_{Mem} - 30 = 120 + 0.2 \times T_{Mem}$

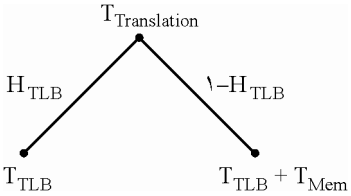
همچنین مطابق فـرض سـوال سیسـتم S_3 پـس از دریافـت پاسـخ از TLB و عـدم یـافتن آدرس، درخواست را به حافظه اصلی ارسال می‌کند. به عبارت دیگـر نحـوه عملکـرد سیسـتم S_3 **ترتیبـی** (سری) در نظرگرفته شده‌است، به صورت زیر:

$\mathbf{T_{Translation}}$: میانگین زمان ترجمه در حالت **ترتیبی**

توجه: مطابق فرض سوال در **بخـش ترجمـه** یعنـی $T_{Translation}$، دو عنصـر T_{TLB} و T_{Mem} بـرای ترجمه وجود دارد، بنابراین باید میانگین آنها محاسبه گردد.

برای محاسبه $T_{Translation}$ میانگین T_{TLB} و T_{Mem} را محاسبه می‌کنیم که همـان $T_{Translation}$ اسـت. 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.

$T_{Translation} = T_3 = T_{TLB} + (1\text{-}H_{TLB}) \times T_{Mem}$

$T_{Translation} = T_3 = 120 + (1\text{-}0.7) \times T_{Mem} = 120 + 0.3 \times T_{Mem}$

خواسته سوال این است که **کمترین** و **بیشترین** میانگین زمان پاسخ این سـه سیسـتم بـه ترتیـب از راست به چپ چند نانوثانیه است؟

که مطابق آنچه پیشتر گفتیم، داریم:

$T_{Translation} = T_1 = 120 + (1\text{-}0.6) \times (T_{Mem} - 120) = 120 + 0.4 \times T_{Mem} - 48 = 72 + 0.4 \times T_{Mem}$

$T_{Translation} = T_2 = 150 + (1\text{-}0.8) \times (T_{Mem} - 150) = 150 + 0.2 \times T_{Mem} - 30 = 120 + 0.2 \times T_{Mem}$

$T_{Translation} = T_3 = 120 + (1\text{-}0.7) \times T_{Mem} = 120 + 0.3 \times T_{Mem}$

متاسفانه مقدار T_{Mem} در صورت سوال توسط طراح محترم بیان نشده است، که باعث شده ادامه حل مساله امکان‌پذیر نباشد، اما اگر مقدار T_{Mem} را برابر 600 ns در نظر بگیریم، آنگاه نتایج زیر را خواهیم داشت:

$T_{Translation} = T_1 = 72 + 0.4 \times T_{Mem} = 72 + 240 = 312$ ns

$T_{Translation} = T_2 = 120 + 0.2 \times T_{Mem} = 120 + 120 = 240$ ns

$T_{Translation} = T_3 = 120 + 0.3 \times T_{Mem} = 120 + 180 = 300$ ns

اگر مقدار T_{Mem} از 600 ns بیشتر شود، آنگاه **بیشترین** میانگین زمان پاسخ که مربوط به T_1 و برابر 312 است، از 312 بیشتر می‌شود که در گزینه‌ها بیشتر از مقدار 312 وجود ندارد. بنابراین در این حالت **کمترین** و **بیشترین** میانگین زمان پاسخ این سه سیستم به ترتیب برابر 312 و 240 نانوثانیه است که در هیچیک از گزینه‌ها وجود ندارد.

توجه: منظور طرح از زمان پاسخ، محاسبه زمان ترجمه یعنی $T_{Translation}$ در شرایط مطرح شده است، و نه محاسبه زمان دسترسی به یک مقصد مورد نظر، بنابراین در محاسبات فوق از زمان مقصد یعنی $T_{Destination}$ استفاده نکردیم.

توجه: سازمان سنجش آموزش کشور در کلید اولیه خود ابتدا گزینه دوم را به عنوان پاسخ اعلام نمود، سپس در کلید نهایی نظر خود را عوض کرد و کلاً تست را حذف نمود، که عمل درستی را انجام داده است که البته عمل درست‌تر آن است که سؤال از ابتدا، درست طرح شود.