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

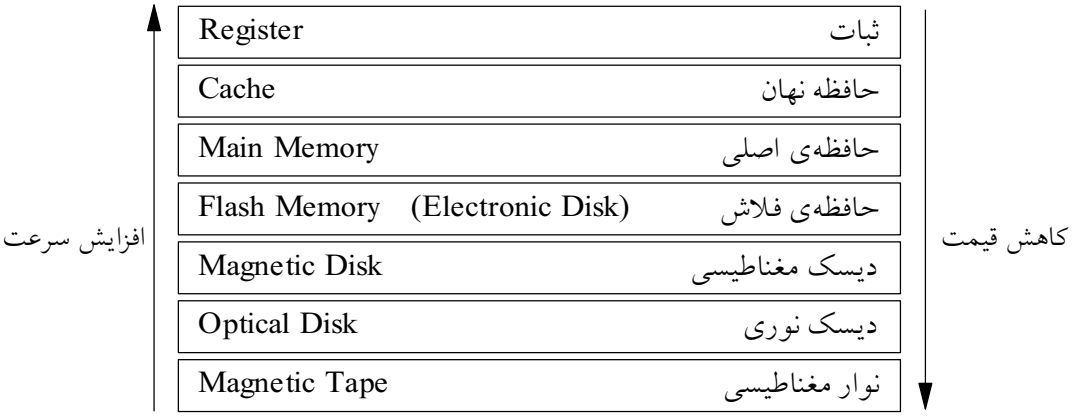
۴ مدیریت حافظه (قطعه‌بندی و صفحه‌بندی)

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:

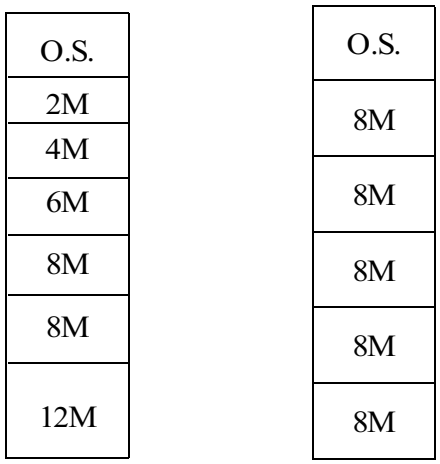
الف) حافظه به بخش‌های **مساوی** تقسیم می‌شود و در اختیار فرایندها قرار می‌گیرد. این شیوه دو نقص عمده دارد:

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.

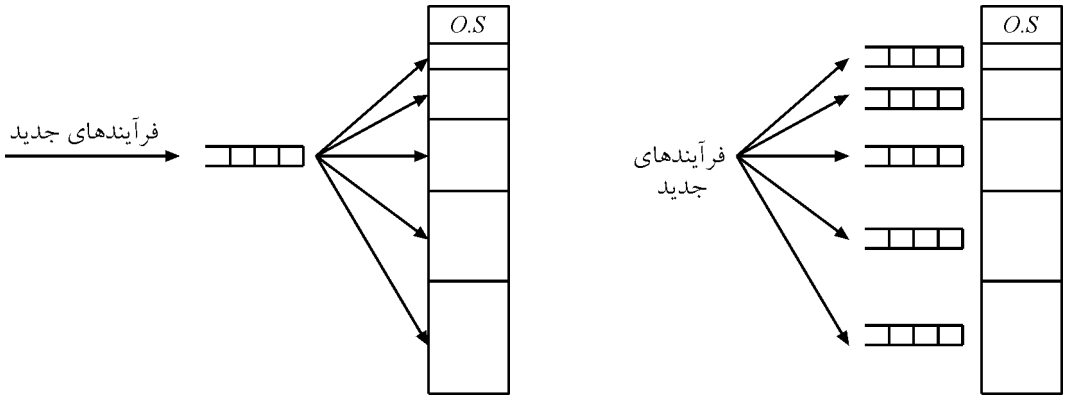
شکل زیر، یک مثال از این دو روش را نشان می‌دهد.



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:

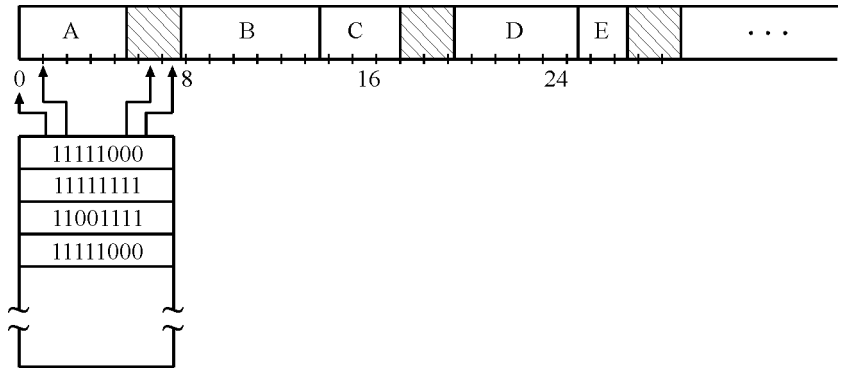
۱ـ روش Bitmap (نقشه بیتی)

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.

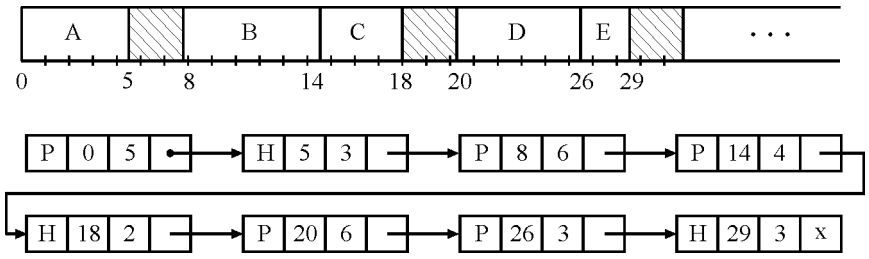
شکل زیر این روش را نشان می‌دهد:



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).

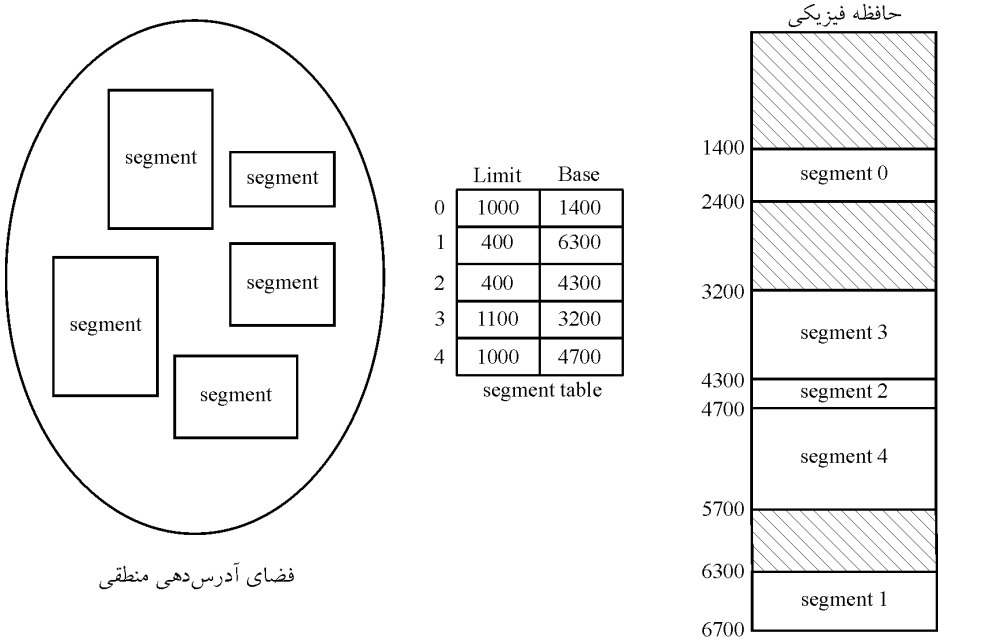
از آنجا که طول قطعات با هم برابر نیستند، روش قطعه‌بندی همانند روش بخش‌بندی پویا است. در واقع در این روش، یک فرآیند به چندین قطعه تقسیم می‌شود که کماکان برای اجرای فرآیند، همه قطعات باید به داخل حافظه آورده شوند، اما این قطعات لزوماً نباید در حافظه همجوار باشند.

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:

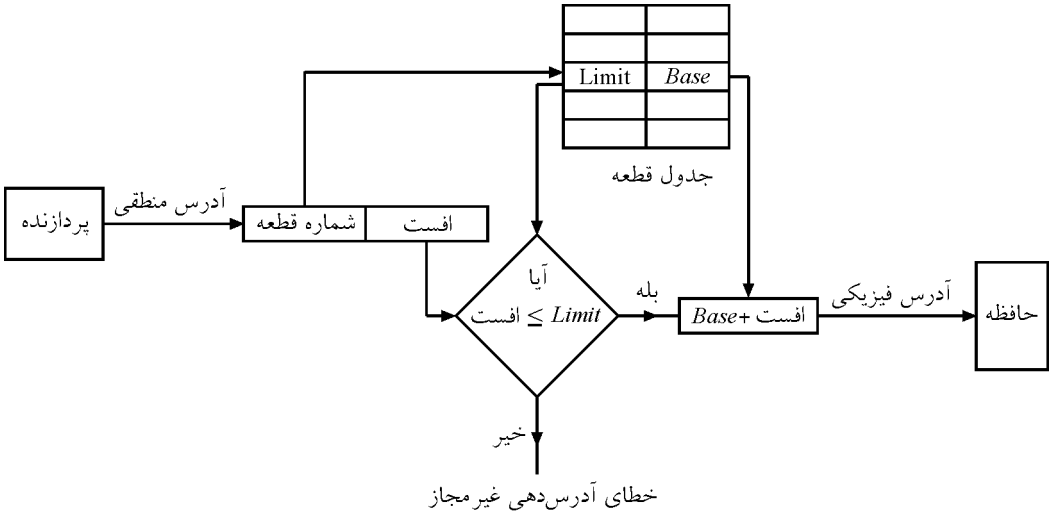
	n بیت	m بیت
: آدرس منطقی	شماره قطعه	آفست

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

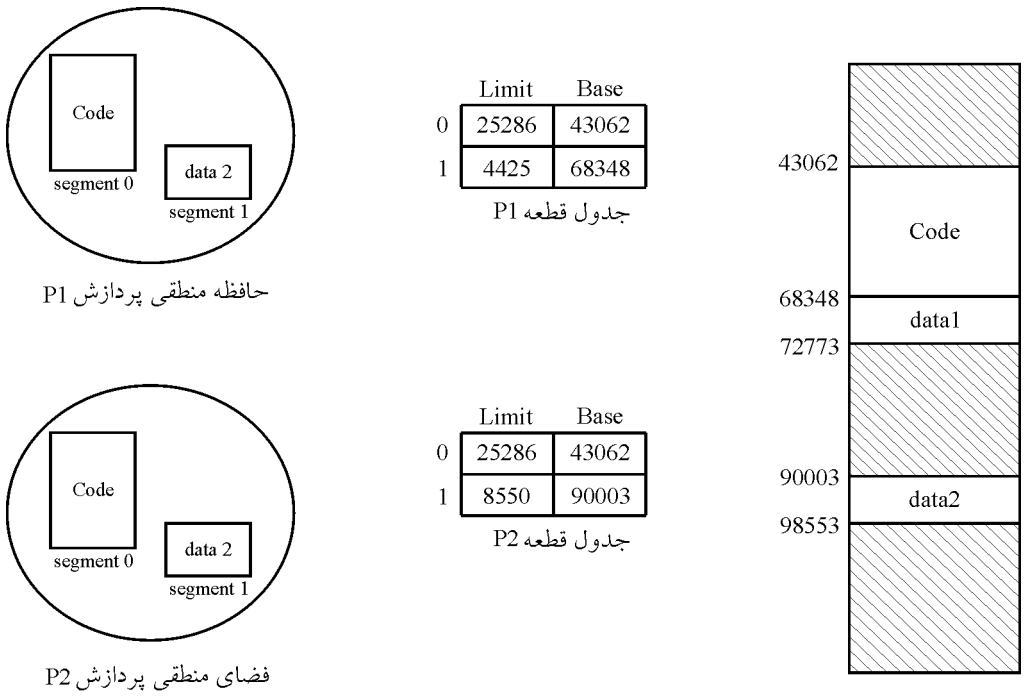
نکته: تکنیک استفاده از TLB همانند صفحه‌بندی می‌تواند در قطعه‌بندی نیز جهت افزایش سرعت دسترسی به حافظه استفاده شود.

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.

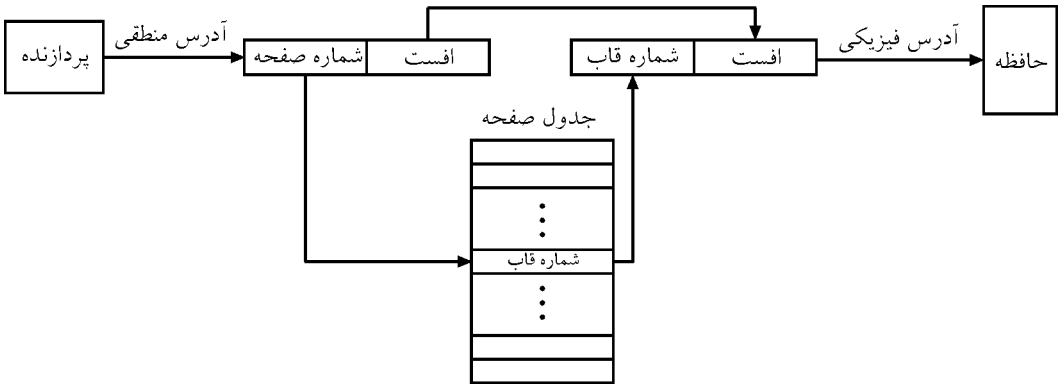
	n بیت	m بیت
آدرس منطقی :	شماره صفحه	آفست

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.

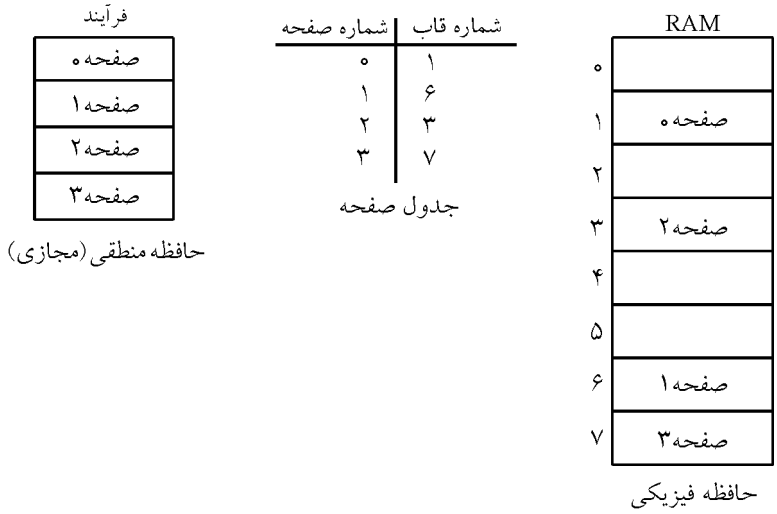
ساده‌ترین و ابتدایی‌ترین راه برای پیاده‌سازی جدول صفحه، استفاده از ثبات‌های سخت‌افزاری پرسرعت می‌باشد. در واقع در این روش به ازای هر درایه جدول صفحه به یک ثبات نیاز داریم. با این کار سرعت نگاشت و تبدیل آدرس‌ها بسیار بالا می‌رود. اما این روش هنگامی کاربرد دارد که تعداد صفحات به طرز بسیار معقولی کم باشند. 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:

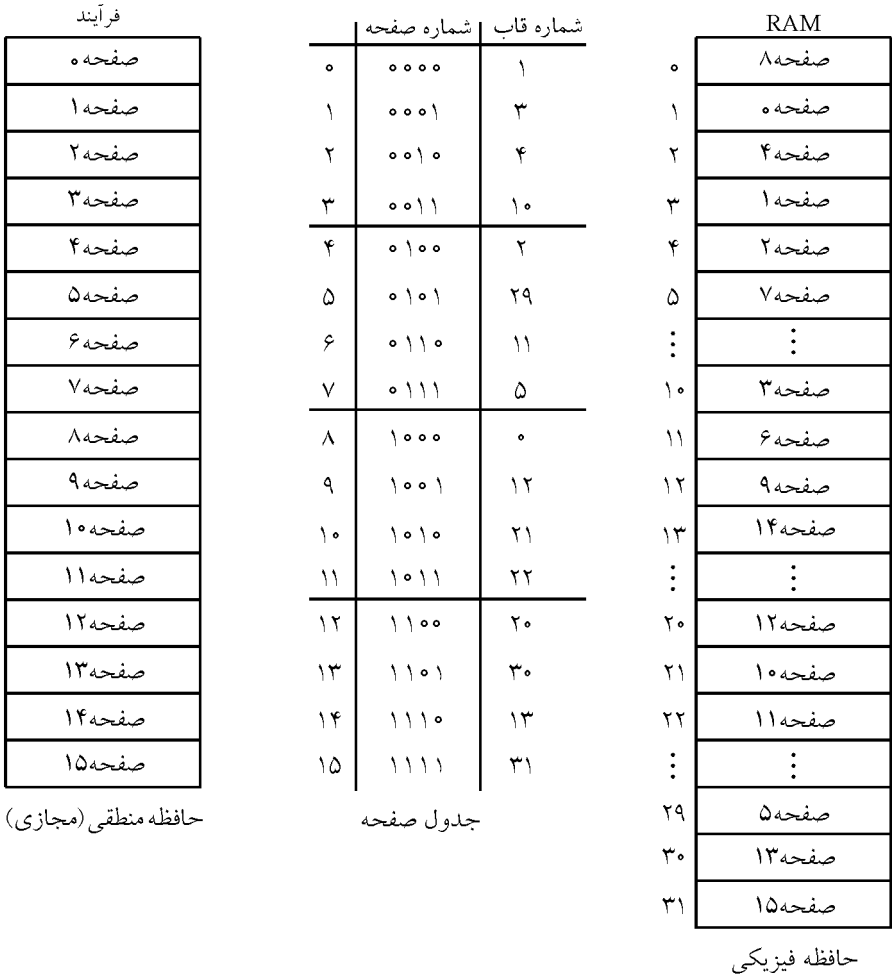
مثال: اگر فرآیندی که از ۴ صفحه منطقی تشکیل شده است به صورت زیر در قاب‌های حافظه فیزیکی قرار گرفته باشد، آنگاه جدول صفحه به صورت زیر درخواهد آمد:



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:

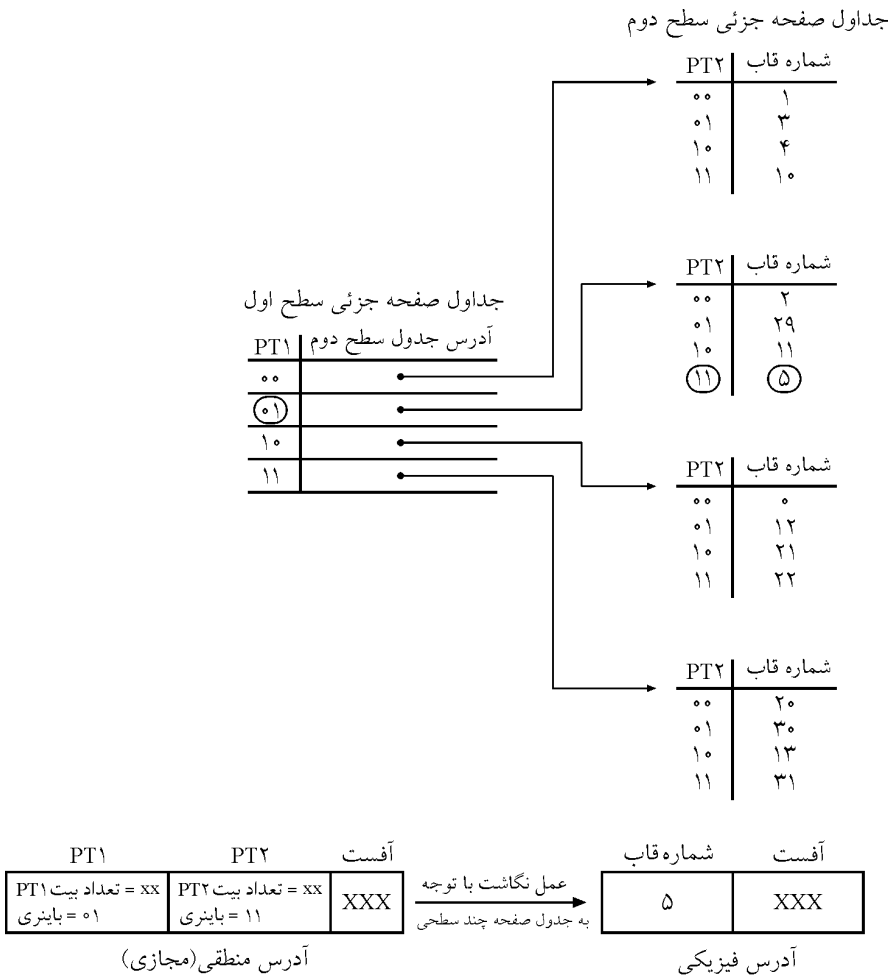


$$\text{بیت ۴} = \log_2^{2^4} = \log_2^{\text{تعداد صفحات فرآیند}} = \text{تعداد بیت شماره صفحه}$$

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

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

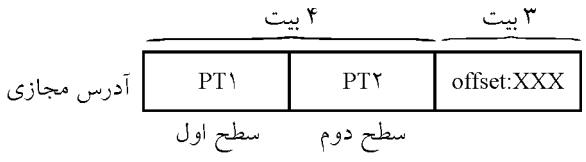
$$\text{تعداد جداول صفحه جزئی در سطح دوم} = \frac{\text{تعداد صفحات فرآیند}}{r} = \frac{f}{r} = \frac{۱۶}{۴} = ۴$$

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

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 $= ۲^{۴} = ۲^{(۲)} \times ۲^{(۲)}$

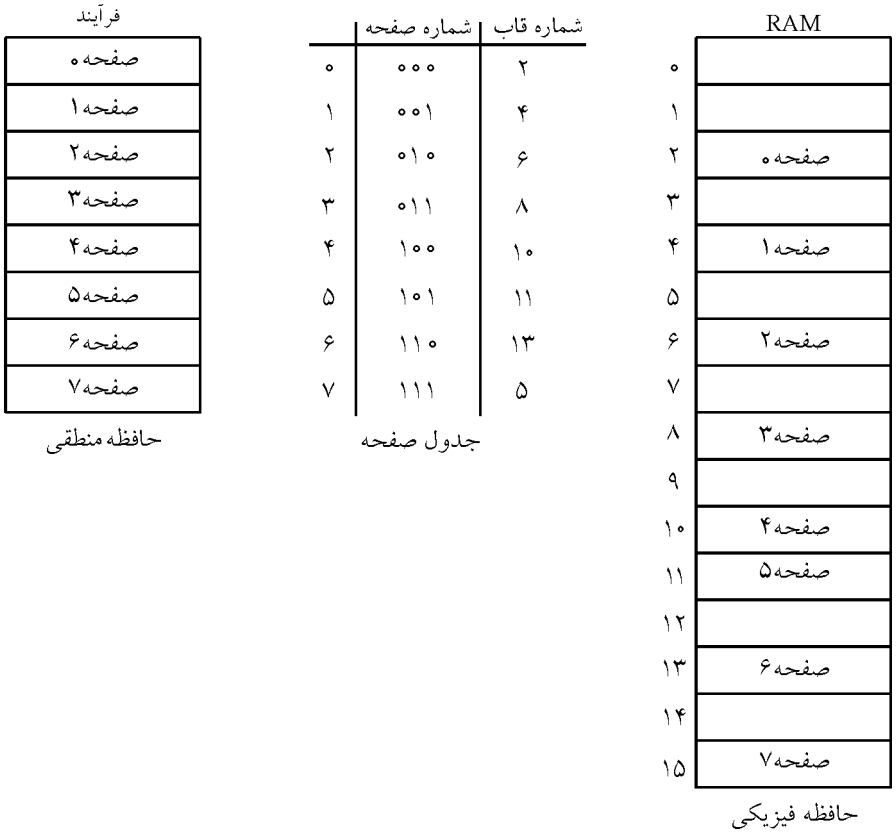
توجه: تجزیه را از چپ به راست و از اندیس n به ۱ شروع کنید و کمترین مقدار در اندیس ۱ قرار داده شود.

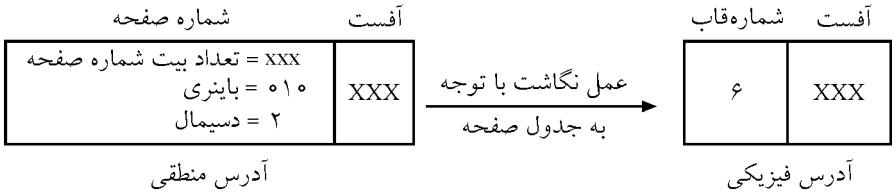
نکته مهم: توان کمتر همواره PT1 است. که در این مثال PT1 و PT2 برابر هستند.

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:



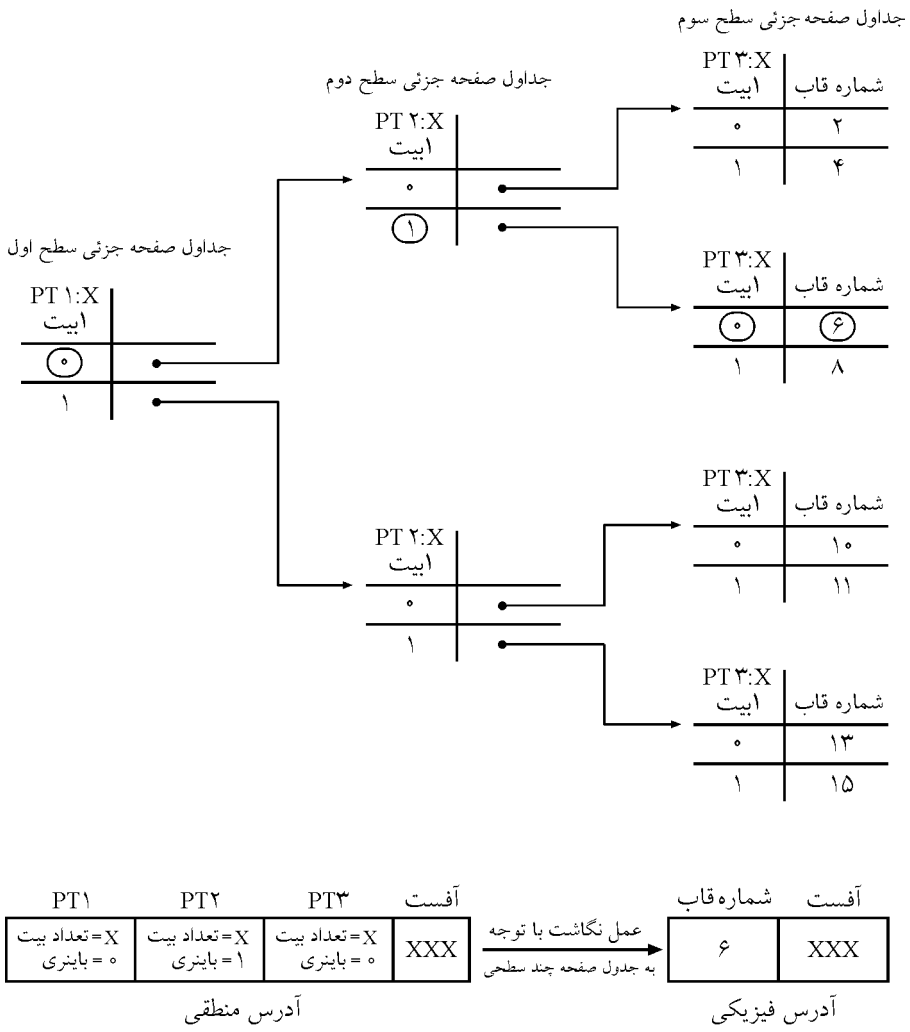


$$\text{بیت ۳} = \log_{۲}^{۲^{۳}} = \log_{۲}^{\text{تعداد صفحات فرآیند}} = \text{تعداد بیت شماره صفحه}$$

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

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:

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

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

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

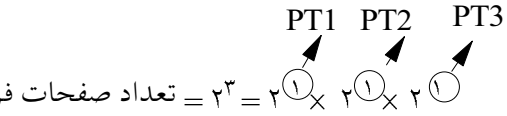
توجه: در اینجا تعداد تقسیم متوالی برابر ۳ است، بنابراین تعداد سطوح جدول چند سطحی برابر ۳ است.

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.



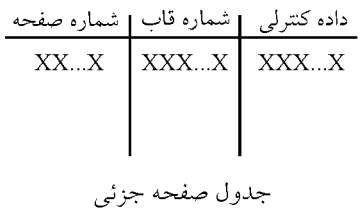
توجه: تعداد عملوندها برابر ۳ است، بنابراین تعداد سطوح جدول چند سطحی نیز برابر ۳ است.

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؟

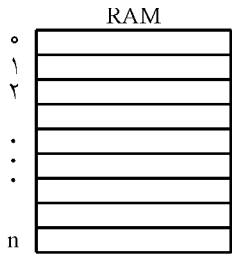
پاسخ: در این جا برای جدول صفحه جزئی محدودیتی به اندازه یک قاب داریم.

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.

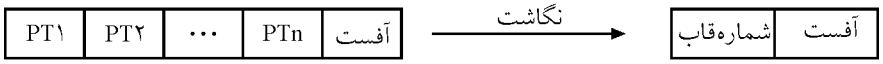
به شکل زیر توجه کنید:



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.

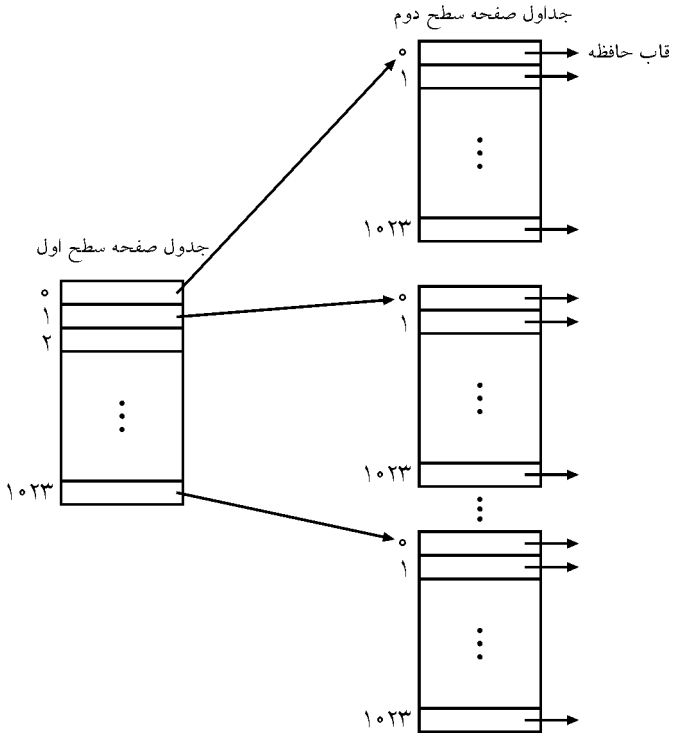
مثال: فرض کنید در یک سیستم آدرس‌های مجازی، ۳۲ بیتی هستند. در این کامپیوتر آدرس‌های مجازی به سه بخش تقسیم شده‌اند. دو فیلد ۱۰ بیتی به نام‌های PT1 و PT2 و یک آفست ۱۲ بیتی به صورت زیر:

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:

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

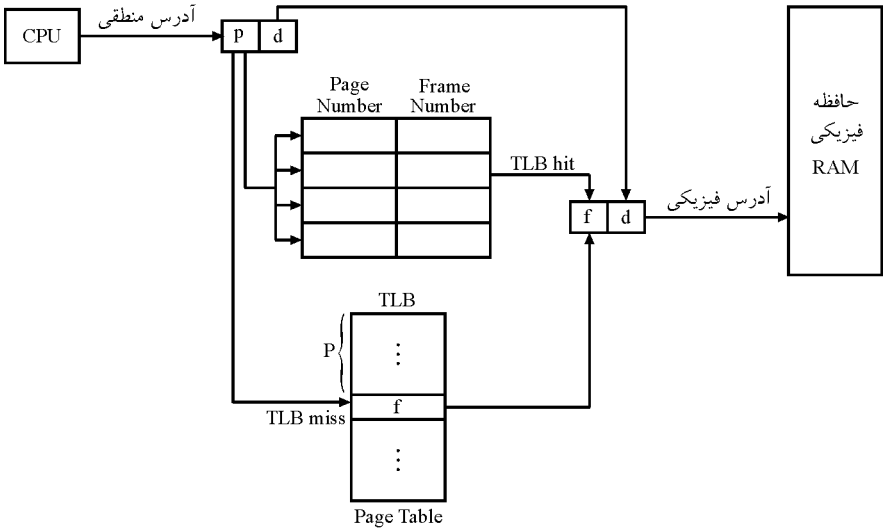
$$\text{تعداد جداول صفحه جزئی در سطح اول} = \frac{\text{تعداد جداول صفحه جزئی در سطح دوم}}{r} = \frac{۲^{۱۰}}{۲^{۱۰}} = ۱$$

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).