فصل ششم

مدیریت حافظه حقیقی

۶-۱ - مدیریت حافظه

سازمان و مدیریت حافظه اصلی در یک سیستم کامپیوتری از مهم‌ترین عوامل در طراحی سیستم عامل می‌باشند. منظور از حافظه اصلی، حافظه‌ایی است که پردازنده برای دست‌یابی به دستورالعمل‌ها و داده‌ها مستقیماً به آن رجوع می‌کند.

منظور از **سازمان حافظه** این است که تعیین می‌گردد:

ـ آیا تنها یک کاربر در حافظه اصلی قرار می‌گیرد و یا این‌که چند کاربر هم‌زمان از حافظه استفاده می‌کند؟

ـ در صورتی که چند برنامه کاربردی در آن واحد از حافظه اصلی استفاده نمایند، آیا به تمام آن‌ها مقدار یکسانی از حافظه تعلق می‌گیرد و یا این‌که حافظه اصلی به نواحی به اندازه های مختلف به نام بخش تقسیم می‌گردد و این بخش‌ها به کاربران اختصاص خواهند یافت؟

ـ آیا حافظه به گونه‌ای تقسیم بندی می‌گردد که بخش‌های ثابت و بدون تغییر ایجاد گردند؟

و یا این‌که این تقسیم بندی حافظه به صورت پویاتری انجام می‌پذیرد و به سیستم کامپیوتری امکان می‌دهد تا بر حسب نیازهای کارهای کاربران تغییراتی را در بخش‌ها اعمال نماید؟

ـ آیا لازم است که کارهای کاربر به گونه‌ای طراحی شوند که در ناحیه‌ای خاص اجرا گردند و یا این‌که این امکان وجود دارد که کارها در هر مکانی که بتوانند در آن گنجانده شوند، اجرا گردند؟

ـ آیا لازم است که هر کاری در بلوک پیوسته‌ای از مکان‌های حافظه قرار گیرد یا این‌که کار می‌تواند در بلوک‌های جداگانه توزیع گردد و در هر شکاف و فضای خالی موجود در حافظه قرار گیرد؟

روش‌های مدیریت حافظه تعیین می‌کنند که:

ـ چه زمانی یک برنامه جدید برای قرار گرفتن در حافظه انتخاب می‌گردد؟

ـ آیا این انتخاب زمانی رخ می‌دهد که سیستم برنامه جدید را تقاضا کند و یا این‌که سعی بر این خواهد بود که تقاضای سیستم پیش بینی شوند؟

ـ در کجای حافظه اصلی برنامه بعدی برای اجرا باید قرار گیرد؟

ـ آیا این جایگذاری برنامه در حافظه به نحوی است که تا حد امکان در شکاف های موجود در حافظه صورت گیرد، به طوری که فضای هدر رفته را کاهش دهد و یا برنامه‌ها به نحوی جایگذاری می‌گردند که زمان اجرا کاهش یابد؟

ـ در صورتی که لازم باشد برنامه جدیدی در حافظه قرار گیرد و اگر در حال حاضر حافظه اصلی به طور کامل پر باشد، کدام‌یک از برنامه‌های موجود در حافظه باید جای خود را به برنامه جدید بدهد؟

ـ روش تعیین برنامه‌ای که باید از حافظه خارج گردد چیست؟

به طور کلی برای مدیریت حافظه می‌توان ۴ وظیفه در نظر گرفت:

۱ـ ثبت وضعیت هر یک از مکان‌های حافظه ـ به این معنی که مشخص شود در هر لحظه چه مکانی از حافظه آزاد و یا اشغال است ؟

۲ـ تعیین سیاست و خط‌مشی تخصیص حافظه ـ به این معنی که مدیریت حافظه تصمیم بگیرد که حافظه به چه کسی، به چه مقدار، در چه زمانی و در کجا اختصاص یابد. در صورتی که فرآیندها بخواهند به طور مشترک و هم‌زمان از حافظه استفاده کنند، مدیریت حافظه باید تعیین کند که به تقاضای کدام فرآیند برای تخصیص حافظه باید جواب داده شود.

۳ـ تکنیک تخصیص حافظه اصلی ـ هر زمان که تصمیم به تخصیص حافظه گرفته شود، مکان خاصی از حافظه باید انتخاب گردد و اطلاعات وضعیتی درباره حافظه اختصاص یافته اصلاح گردند.

۴ـ تکنیک آزاد کردن حافظه اصلی ـ برای آزاد کردن حافظه اختصاص یافته باید سیاستی وجود داشته باشد. یک فرآیند خود می‌تواند حافظه‌ای را که پیش‌تر به آن اختصاص یافته است آزاد اعلام نماید و یا این‌که مدیریت حافظه خود مبادرت به آزاد سازی حافظه نماید.

۶-۲- پیوند آدرس (Address Binding)

آدرس های برنامه منبع(Source Program)، معمولاً به فرم سمبولیک می‌باشند، وظیفه مترجم‌هاست که این آدرس‌های سمبولیک را به آدرس های قابل جابه‌جایی تبدیل نمایند. ویرایشگرهای پیوندی یا بارکننده‌ها (Linkage Editor or loader) به نوبه خود، این آدرس های قابل جابه‌جایی را به آدرس‌های مطلق (Absolute) پیوند می‌دهند. هر پیوند، نگاشتی از یک فضای آدرس به فضای آدرس دیگر می‌باشد.

به طور کلی پیوندها در مسیر زیر در زمان‌های گوناگون صورت می‌پذیرند.

۱ـ زمان کمپایل:

اگر در زمان ترجمه معلوم گردد که فرآیند در چه بخشی از حافظه قرار خواهد گرفت (آدرس مبنا در زمان ترجمه مشخص می‌شود) در این صورت کد مطلق می‌تواند تولید شود. هر گونه تغییری در آدرس مبنای برنامه، مستلزم ترجمه دوباره برنامه می‌باشد. برنامه‌های با پسوند **Com** در سیستم عامل **MS-DOS** از آدرس های مطلق پیوند یافته در زمان ترجمه تشکیل شده اند.

۲ـ زمان بارگذاری :

در این حالت مترجم، کد قابل جابه‌جایی تولید می‌نماید و پیوند دادن نهایی تا زمان بارگذاری به تعویق انداخته می‌شود. چنان‌چه آدرس شروع عوض شود، فقط کافی است که کد کاربر را دوباره بارگذاری نماییم.

۳ـ زمان اجرا:

چنان‌چه فرآیندی در زمان اجرایش بتواند تغییر مکان دهد، آن‌گاه پیوند آدرس ها تا زمان اجرا به تعویق می افتد. در این حالت سخت افزار خاصی برای حمایت از این طرح باید موجود باشد. در این شیوه ثبات مبنا را ثبات جابه‌جایی (Relocation) می‌نامند. مقدار ثبات مبنا به هر آدرسی که توسط فرآیند کاربر ایجاد می‌گردد(در زمانی که به حافظه فرستاده می‌شود) اضافه خواهد شد. به عنوان مثال اگر مبنا در آدرس ۱۴۰۰۰ حافظه باشد، در آن صورت سعی کاربر برای دست‌یابی به مکان صفر منجر به دست‌یابی به آدرس ۱۴۰۰۰ خواهد شد و دست‌یابی به آدرس X منجر به رجوع به آدرس (X+ ۱۴۰۰۰) می‌گردد.

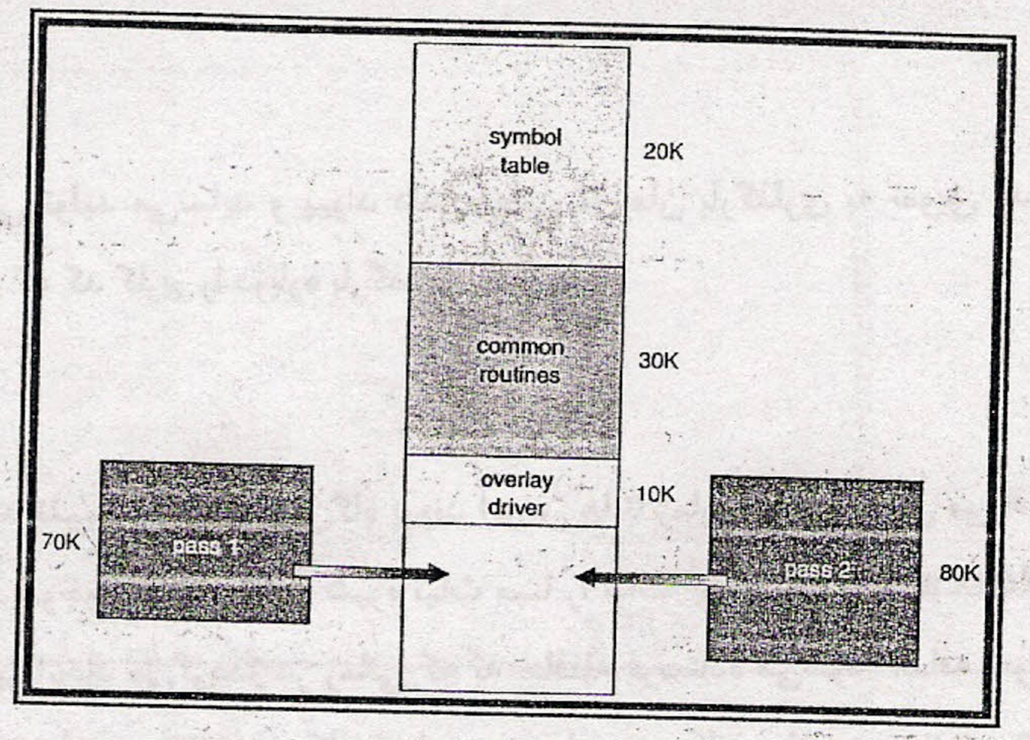
۶-۳- فضای آدرس فیزیکی و منطقی

آدرس منطقی همان آدرس تولید شده توسط پردازنده می‌باشد (به عبارت ساده تر برنامه کاربر همواره با آدرس های منطقی سرو کار دارد) در حالی که آدرس فیزیکی آدرس قابل رویت توسط واحد حافظه است.

آدرس فیزیکی و منطقی در پیوندهای زمان ترجمه و زمان بارگذاری یکسانند،اما پیوند آدرس در زمان اجرا دارای فضای آدرس فیزیکی و منطقی متفاوتی می‌باشد. مجموعه آدرس های منطقی به‌نام فضای آدرس منطقی و مجموعه آدرس های فیزیکی به نام فضای آدرس فیزیکی نامیده می‌شود.

۶-۴ - جایگذاشت ها (Overlays)

به منظور آن‌که فرآیندی محدود به اندازه حافظه فیزیکی نگردد و بتواند بزرگ‌تر از حافظه تخصیص یافته به آن، اجرا شود. تکنیکی به‌نام **جایگذاشت** به‌کار می‌رود. ایده جایگذاشتن آن است که در هر زمانی که داده و یا کدی از برنامه مورد نیاز می‌باشد در حافظه بار گردد. هنگامی‌که دستورالعمل‌های دیگری مورد نیاز است، این دستورات در فضایی که دستورات قدیمی غیرقابل استفاده اشغال نموده بودند، بار می‌گردد. (بخشی از حافظه در زمان‌های مختلف توسط کدهای گوناگون مورد استفاده قرار می‌گیرد)(شکل ۶-۱). این روش نیاز به پشتیبانی خاص از سوی سیستم عامل ندارد، بلکه کاملاً توسط کاربر می‌تواند پیاده سازی گردد. این کار با استفاده از ساختارهای فایل ساده‌ای انجام پذیر است. بدین ترتیب که محتویات این فایل به حافظه آورده می‌شود و سپس انتقال به حافظه مورد نظر صورت می‌پذیرد و دستوراتی که هم اکنون خوانده شده‌اند اجرا می‌گردند. سیستم عامل تنها متوجه خواهد شد که I/O بیش‌تری به کار گرفته می‌شود. برنامه نویس از سوی دیگر باید ساختار جایگذاری را به شایستگی طراحی و برنامه ریزی نماید. روش‌های خود کار برای اجرای برنامه‌های بزرگ در حافظه فیزیکی محدود، مطمئناً قابل قبول تر هستند.



شکل ۶-۱

۶-۵ - مدیریت حافظه یک پارچه

تخصیص حافظه به صورت یک پارچه یک شیوه مدیریت حافظه ساده است که نیاز به پشتیبانی سخت افزار خاص ندارد. در چنین سیستم‌هایی، عملکرد چند برنامگی وجود ندارد و یک تناظر یک به یک بین کاربر و فرآیند وجود دارد. حافظه در ظاهر به ۳ ناحیه متوالی و پیوسته تقسیم می‌شود.

بخشی از حافظه به طور ثابت به سیستم عامل اختصاص دارد. کلیه فضای باقیمانده در دسترس و آزاد است و به یک فرآیند اختصاص داده خواهد شد. پردازش مورد نظر در واقع بخشی از حافظه اختصاص داده شده را مورد استفاده قرار می‌دهد و ناحیه‌ای را بی استفاده باقی می‌گذارد.

در این شیوه پردازش در هنگام زمان‌بندی، کل حافظه را به خود اختصاص می‌دهد. هنگامی‌که فرآیند انجام شد کل حافظه به وضعیت آزاد بازگردانده می‌شود. در این صورت برنامه‌ها از لحاظ اندازه محدود به مقدار حافظه اصلی هستند، اما این امکان وجود دارد که با استفاده از جایگذاری، برنامه‌های بزرگ‌تر از اندازه حافظه اصلی را اجرا نمود.

در این روش مکانیزم حفاظتی ساده و ابتدایی به کار گرفته می‌شود تا اطمینان حاصل گردد که برنامه کاربران با سیستم عامل تداخل نمی‌نماید. این مکانیزم می‌تواند متشکل از یک ثبات مرزی و یک حالت کاربر و سیستم در ارتباط با پردازنده باشد. در صورتی که سعی به دست‌یابی به فضای حفاظت شده (سیستم‌عامل) انجام گیرد، وقفه‌ای رخ می‌دهد و کنترل به سیستم عامل منتقل می‌شود.

۶-۶ - روش‌های مدیریت حافظه چند برنامگی

وظیفه اصلی مدیریت حافظه انتقال برنامه به داخل حافظه جهت اجرا توسط پردازنده می‌باشد. در تمام سیستم‌های عامل جدید چند برنامگی انجام این وظیفه همراه با طرح حافظه مجازی صورت می‌پذیرد. حافظه مجازی به نوبه خود به دو روش اساسی **قطعه‌بندی** و **صفحه‌بندی** مبتنی است. روش‌های صفحه‌بندی و قطعه‌بندی ساده که در سیستم‌های جدید به‌کار گرفته نمی‌شوند به منظور روشن‌تر ساختن بحث در مورد حافظه مجازی می‌توانند مورد مطالعه قرار گیرند و هم‌چنین یکی دیگر از روش‌های مدیریتی حافظه به‌نام بخش‌بندی که در بسیاری از سیستم‌های عامل امروزه از رده خارج شده‌اند مورد بحث قرار می‌گیرد.

۶-۶-۱- روش بخش بندی ایستا (Static)

در این روش سیستم عامل بخش ثابتی از حافظه اصلی را اشغال می‌نماید و بقیه حافظه می‌تواند به دو صورت مختلف بخش‌بندی گردد:

۱ـ امکان اول استفاده از بخش‌هایی با اندازه مساوی که در این حالت هر فرآیندی که اندازه آن کمتر یا مساوی اندازه بخش باشد می‌تواند به داخل هر بخش موجود بار شود. اگر تمام بخش‌ها پر باشند و هیچ یک از فرآیندهای مقیم در حالت آماده یا اجرا نباشد، سیستم عامل می‌تواند فرآیندی از هر یک از بخش‌ها را به خارج مبادله و فرآیند دیگری را بار نماید تا کاری برای پردازنده ایجاد گردد.

در این حالت به دو مشکل بر می خوریم:

الف ـ ممکن است برنامه بزرگ‌تر از آن باشد که در یک بخش جای گیرد. در این‌صورت **روش جایگذاری** می‌تواند استفاده شود.

ب ـ استفاده از حافظه اصلی ناکارآمد می‌شود. هر برنامه ای صرف نظر از این‌که چقدر کوچک باشد، بخش کامل را اشغال می‌کند. فضای به هدر رفته را **تکه تکه شدن داخلی (Internal Fragmentation)** می‌گویند.

۲ـ امکان دوم استفاده از بخش‌هایی با اندازه نامساوی که در این حالت هر دو مشکل مطرح شده می‌تواند کاهش یابند. در این روش تخصیص هر فرآیند می‌تواند به کوچک‌ترین بخشی باشد که در آن جای می‌گیرد. در این حالت یک صف زمان‌بندی برای هر بخش لازم است تا فرآیندهای مبادله شده مربوط به آن بخش را نگهداری نماید. در این رویکرد تکه تکه شدن داخلی به حداقل می‌رسد. رویکرد دیگری که از دیدگاه سیستم بهینه‌تر می‌باشد به کارگیری یک صف منفرد برای تمام فرآیندهاست. هنگامی‌که وقت بار شدن یک فرآیند به داخل حافظه اصلی برسد، کوچک‌ترین بخش برای نگهداری فرآیند انتخاب خواهد شد. در این رویکرد پیوند آدرس ها در زمان بارگیری صورت می‌پذیرد.

به طورکلی طرحهای بخش‌بندی ایستا نسبتاً ساده هستند و حداقل به نرم افزار سیستم عامل نیاز دارد، اما از **معایب** زیر برخوردار می‌باشند.

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

۲ـ کارهای کوچک باعث تکه تکه شدن داخلی بیش‌تر می‌شود.

۶-۶-۲ – روش بخش بندی پویا (Dynamic)

در این روش، سیستم عامل جدولی شامل بخش‌های حافظه آزاد و اشغال را نگه می دارد.

در ابتدا کل حافظه همانند یک بلوک بزرگ آزاد در نظر گرفته می‌شود. وقتی فرآیندی درخواست حافظه می‌نماید فضایی به اندازه کافی بزرگ را به آن تخصیص می‌دهیم و باقیمانده فضا را برای درخواست‌های آتی فرآیند آزاد نگه می داریم.

در حالت کلی، در هر لحظه یک مجموعه از فضاهای با اندازه‌های متغیر، به صورت پراکنده در سرتاسر حافظه، وجود دارند بدین معنی که با بار شدن برنامه‌ها و در پایان آزاد شدن فضاهای اشغال شده، حافظه به مرور به قطعاتی تقسیم می‌شود که ممکن است این قطعات به قدری کوچک باشند که مورد استفاده هیچ برنامه ایی قرار نگیرد.

این نوع تکه تکه شدن حافظه را **تکه تکه شدن خارجی(External Fragmentation)**می‌نامند که باعث اتلاف حافظه می‌شود.

سیستم عامل بخش مدیریت حافظه باید از وضعیت حافظه شامل محل شروع، طول یک بخش اشغال شده و برنامه‌هایی که این بخش‌ها را در اختیار دارند، مطلع باشد. بدین منظور می‌توان از دو ساختار استفاده نمود.

الف ـ روش Bitmap

در این روش حافظه به واحدهای تخصیص تقسیم می‌شود (یک کلمه و یا یک بلوک). متناظر با هر واحد تخصیص یک بیت در یک بردار از بیت‌ها گرفته می‌شود که در صورت آزاد بودن واحد، مقدارش صفر است و درصورت اشغال بودن آن مقدارش یک است.

ب ـ روش لیست پیوندی (Linked List)

روش دیگری جهت ردگیری و ثبت حافظه، نگهداری یک لیست پیوندی از قطعات تخصیص یافته و آزاد می‌باشند که در آن محل شروع، طول قطعه، وضعیت پر و خالی بودن قطعه و اشاره گر به مدخل بعدی مشخص می‌گردد.

یک روش برای مقابله با تکه تکه شدن خارجی، **فشرده سازی (Compaction)** می‌باشد. در این حالت سیستم عامل فرآیندها را طوری انتقال می‌دهد تا کنار هم دیگر قرار گیرند و تمام حافظه آزاد به صورت یکپارچه در یک بلوک واقع شود و بدین ترتیب به احتمال زیاد فرآیند جدید به اندازه کافی فضای آزاد برای بار شدن خواهد داشت. فشرده سازی نیاز به پیوند آدرس‌های فرآیندها در زمان اجرا دارد تا مراجعات به حافظه در برنامه‌ها به درستی صورت گیرد و ضمناً این روش وقت زیادی از پردازنده را برای عمل فشرده سازی تلف می‌نماید. بنابراین بهتر است که سیستم عامل در تصمیم‌گیری برای تخصیص حافظه به فرآیندها با دقت بیش‌تری عمل کند.

۶-۶-۲-۱- روش‌های تخصیص حافظه به فرآیند

الف ـ روش First Fit:

لیست مربوط به فضاهای آزاد حافظه، برحسب مکان فضای آزاد در حافظه، ذخیره و مرتب می‌گردد. هنگامی‌که نیاز به اختصاص حافظه باشد، شروع جستجو برای یافتن اولین فضای آزاد که به اندازه نیاز اعلام شده باشد، از ابتدای لیست صورت می‌گیرد(اولین فضای آزادی که بتواند بخش درخواستی را در خود بگنجاند، انتخاب خواهد شد).

اگر جستجو از محلی در لیست شروع شود که آخرین بار تخصیص از آن محل صورت گرفته است، آن‌گاه روش جدیدی به نام Next Fit حاصل می‌گردد.

ب ـ روش Best Fit:

در این روش، مدیریت حافظه با گرفتن یک تقاضای حافظه، کوچک‌ترین فضای آزادی را که بتواند نیاز اعلام شده را برآورده سازد، تخصیص می‌دهد.

در حقیقت این الگوریتم فضاهای بزرگ را که ممکن است بعداً به آن نیاز داشته باشیم، تقسیم نمی‌نماید، بلکه به دنبال فضایی می‌گردد که اندازه آن به اندازه فرآیند بسیار نزدیک باشد. این روش به‌دلیل ایجاد قطعات خیلی کوچک فضای آزاد، گاهی اوقات باعث افزایش مشکل تکه تکه شدن خارجی می‌شود.

ج ـ روش Worst Fit

برای این‌که مشکل به‌وجود آمدن فضاهای بسیار کوچک در حافظه برطرف گردد این الگوریتم پیشنهاد شده است. این الگوریتم بزرگ‌ترین فضای موجود را انتخاب می‌نماید. بدین دلیل که اگر آن فضا تقسیم شود، فضای به‌وجود آمده باز هم قابل استفاده باشد.

د ـ روش Quick Fit

در این الگوریتم برای هر دسته از فرآیندها، با اندازه های متداول یک لیست جداگانه تهیه می‌شود برای مثال، فرض کنید این الگوریتم دارای جدولی با n درایه باشد. درایه اول این جدول اشاره گری است که به ابتدای لیستی شامل فضاهای ۴ کیلو بایتی اشاره می‌کند.

اشاره گر داریه دوم به فضاهای ۸ کیلو بایتی اشاره می‌کند و به همین ترتیب ادامه می‌یابد. هدف از انجام این کار افزایش سرعت جستجو و بهبود کارآیی مدیریت حافظه است.

هـ ـ روش Buddy (رفاقتی): در این روش همواره بخش‌های آزاد حافظه به صورت قطعاتی با اندازه‌هایی که همه توانی از ۲ هستند در لیست‌های جداگانه قرار می‌گیرند. برای شروع، تمامی فضای موجود برای تخصیص، به عنوان بلوک واحدی به اندازه 2^U در نظر گرفته می‌شود. اگر درخواستی به اندازه P مطرح شود به طوری که $2^{u-1} \prec p \leq 2^u$ باشد، تمامی بلوک تخصیص می‌یابد. درغیر این صورت، بلوک به دو رفیق به اندازه مساوی 2^{u-1} تقسیم می‌گردد. اگر $2^{u-2} \prec p \leq 2^{u-1}$ باشد، درخواست به رفیق با آدرس کوچک‌تر تخصیص می‌یابد. در غیر این صورت رفیق با آدرس کوچک‌تر دوباره به دو قسمت مساوی تقسیم می‌شود. این فرآیند ادامه می‌یابد تا این‌که کوچک‌ترین بلوک، بزرگ‌تر و یا مساوی P گردد و به درخواست تخصیص یابد. این روش از **تکه تکه شدن داخلی** برخوردار می‌باشد.

۶-۶-۳ – مدیریت حافظه صفحه‌بندی ساده (Paging)

راه حل دیگر تکه تکه شدن خارجی آن است که اجازه دهیم فضای آدرس منطقی فرآیند **غیر همجوار** باشد. در مدیریت حافظه صفحه‌بندی شده هر فضای آدرسی متناظر با یک فرآیند، به بخش‌های مساوی به‌نام **صفحه** تقسیم می‌گردد. هم‌چنین حافظه فیزیکی نیز خود به نواحی به اندازه یکسان با صفحه به‌نام **قاب** یا **Frame** شکسته می‌شود. وقتی فرآیندی باید اجرا گردد، کلیه صفحاتش که از دیدگاه برنامه کاربر پیوسته می‌باشد، به داخل قاب‌های آزاد حافظه (که لزوماً یکپارچه و پیوسته نمی‌باشند) بارگذاری می‌شوند. صفحات فرآیندها برروی حافظه کمکی یا پشتیبان قرار دارند و از بلوک‌هایی به اندازه ثابت همانند قاب‌های حافظه اصلی تقسیم می‌شوند(**Slot**). درون برنامه، هر آدرس منطقی متشکل از شماره صفحه و انحراف (**Offset**) در داخل صفحه است این آدرس با **آدرس نسبی** یکسان می‌باشد.

از دیدگاه سخت افزار، برای انجام عمل نگاشت فضای آدرس به حافظه فیزیکی، باید یک جدول به‌نام **جدول صفحه** برای هر فرآیند در نظر گرفت. جدول صفحه یک فرآیند، شامل یک مدخل برای هر صفحه آن فرآیند است، به‌نحوی که شماره صفحه به راحتی بتواند به عنوان شاخص آن جدول عمل کند. هر مدخل جدول صفحه شامل شماره قابی در حافظه است که صفحه مربوطه را نگهداری می‌کند. سیستم عامل فهرستی از قاب‌های آزاد حافظه اصلی که در حال حاضر اشغال نشده را نیز نگهداری می‌نماید(شکل ۶-۲).

برای ساده تر شدن طرح صفحه‌بندی، تاکید می‌کنیم که اندازه صفحه و در نتیجه اندازه قاب باید توانی از ۲ باشد. در این صورت آدرس نسبی که نسبت به ابتدای برنامه تعریف شده و آدرس منطقی که به صورت شماره صفحه و انحراف بیان گردیده است یکسان می‌باشند.

پیاده سازی سخت افزاری ترجمه آدرس مجازی در زمان اجرا، به‌شکل ذیل می‌تواند صورت گیرد:

یک آدرس مجازی **n+m** بیتی را در نظر بگیرید **n** بیت سمت چپ شماره صفحه و **m** بیت سمت راست انحراف را نشان می‌دهد.

مراحل زیر برای ترجمه آدرس لازم هستند:

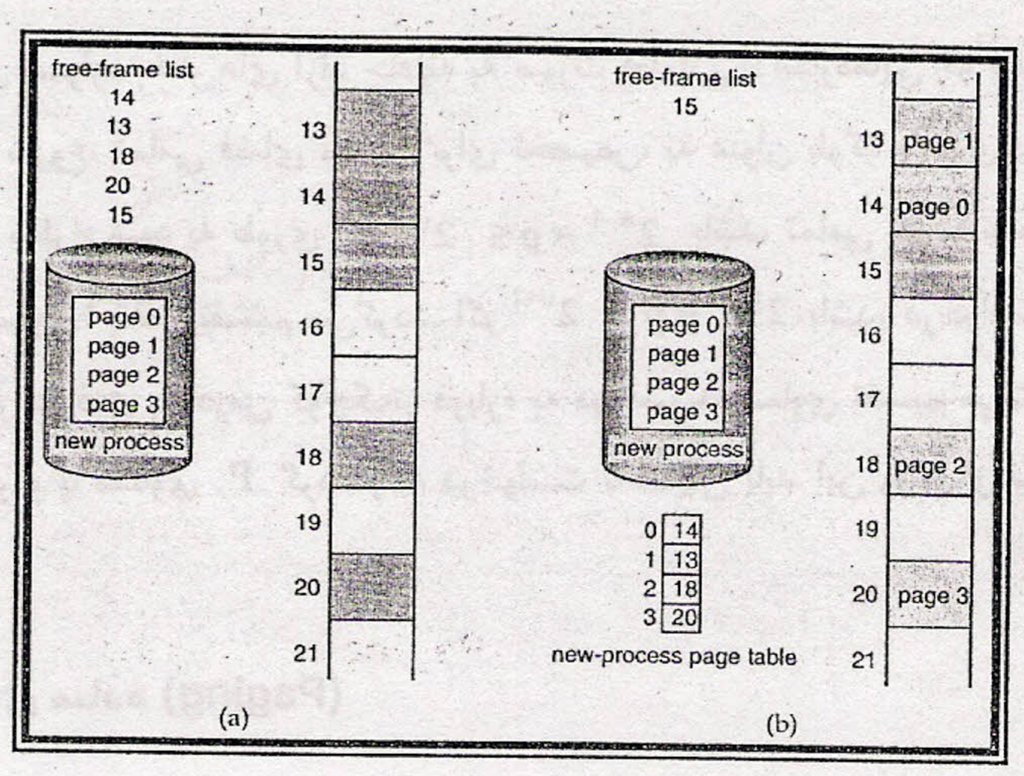
ـ استخراج شماره صفحه (**n** بیت سمت چپ آدرس منطقی)

ـ به کارگیری شماره صفحه به عنوان شاخص به جدول صفحه برای استخراج شماره قاب (**f**) .

ـ برای به‌دست آوردن آدرس فیزیکی شماره قاب را به‌جای شماره صفحه در قالب آدرس منطقی قرار می‌دهیم. **m** بیت سمت راست را که انحراف را نشان می‌دهد بدون تغییر باقی می‌ماند.

وقتی مدل صفحه‌بندی را به کار میبریم، تکه تکه شدن خارجی نداریم. اما ممکن است قدری تکه تکه شدن داخلی وجود داشته باشد. اگر اندازه فرآیند ضریبی از اندازه صفحه نباشد، آن‌گاه آخرین قاب تخصیص یافته ممکن است کاملاً پر نشود و تکه تکه شدن داخلی

رخ دهد. اگر اندازه فرآیند مستقل از اندازه صفحه باشد، انتظار داریم که تکه تکه شدن داخلی به طور متوسط **نیم صفحه به ازای هر فرآیند** باشد. این بررسی، پیشنهاد می‌نماید که اندازه صفحه کوچک، مطلوب‌تر است. اما، در برگیرنده سر بار قابل توجه ائی در تعداد داریه‌های جدول صفحه می‌شود. همین طور، ورودی/خروجی دیسک وقتی حجم داده انتقالی زیاد باشد، بازده بیش‌تری خواهد داشت.



شکل ۶-۲

۶-۶-۳-۱- ساختمان جدول صفحه

پیاده سازی سخت افزاری جدول صفحه، به روش‌های گوناگون می‌تواند انجام گیرد. در حالت ساده می‌تواند به صورت مجموعه‌ای از رجیسترهای اختصاصی پیاده سازی شود. وقتی توزیع کننده فرآیندی را آغاز می‌نماید این رجیسترها را همانند بقیه رجیسترها بارگذاری می‌نماید. این روش به شرطی که جدول صفحه به اندازه قابل توجهی کوچک باشد، ارضا کننده است. اما اکثر کامپیوترهای امروزی جداول صفحه بسیار بزرگی دارند، برای این‌گونه ماشین‌ها، به جای این روش، **جدول در حافظه اصلی** نگهداری می‌شود و یک رجیستر مبنای جدول صفحه (PTBR) به جدول صفحه اشاره می‌نماید. در هنگام تغییر جدول صفحه (به هنگام برگرداندن ـ متن)، فقط تغییر این رجیستر کافی به‌نظر می‌رسد. مشکلی که در این روش به چشم می خورد، زمان دست‌یابی به مکان حافظه کاربراست. اگر بخواهیم مکانی در صفحه i را دسترسی نماییم، ابتدا باید شماره قاب را در جدول صفحه به‌دست آوریم و سپس به محل مورد نظر در حافظه دسترسی نماییم. بنابراین سرعت دسترسی به حافظه با ضریب ۲ کاهش می‌یابد. راه حل این مشکل، استفاده از سخت‌افزار **ثبات‌های انجمنی (Associative) یا TLB (Translation Look Aside Buffer)** می‌باشد.

این رجیسترها مجموعه‌ای از حافظه های با سرعت بالا هستند که از دو بخش تشکیل یافته‌اند، یک کلید و یک مقدار (شماره قاب) در هنگام درخواست کلیدخاصی، کلیه کلیدها هم‌زمان چک می‌گردند، چنان‌چه کلید مورد نظر به‌دست آید مقدار متناظر آن به‌عنوان خروجی حاصل می‌شود (شکل ۶-۳).

زمانی که فرآیندی برای اجرا انتخاب می‌شود بخشی از درایه های جدول صفحه آن فرآیند به رجیسترهای انجمنی منتقل می‌شود. وقتی آدرس منطقی تولید می‌گردد، شماره صفحه آن با کلید رجیسترهای انجمنی مقایسه و شماره قاب حاصل می‌شود. اگر شماره صفحه در رجیسترهای انجمنی یافت نشود، آن‌گاه رجوع به حافظه برای جدول صفحه بایستی صورت پذیرد.

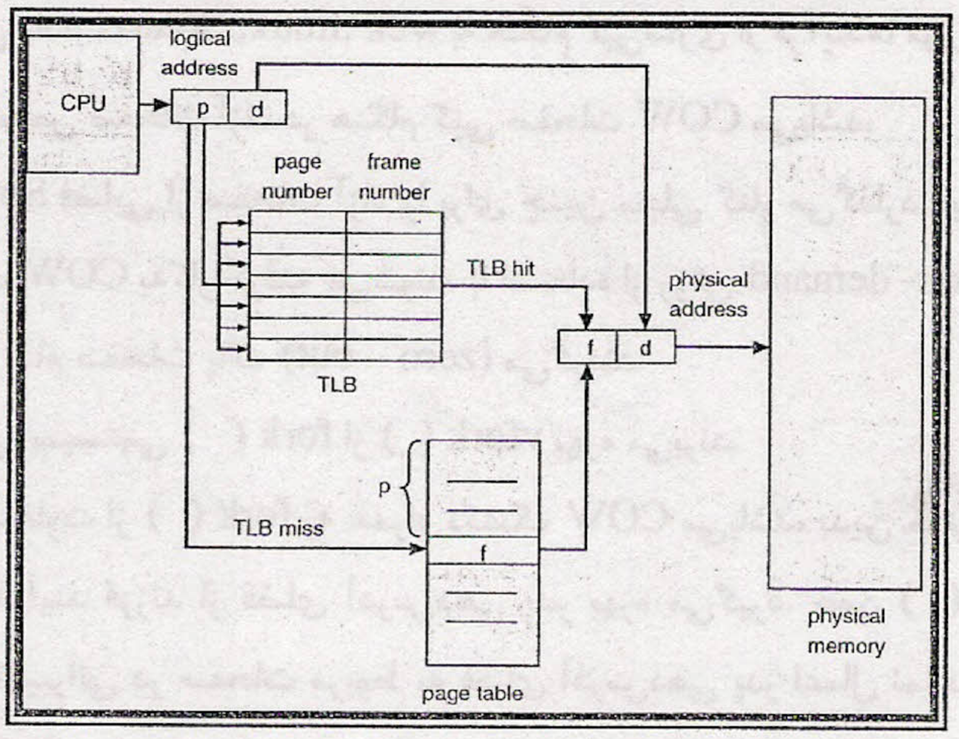
بنابراین در هنگام تبدیل آدرس، اگر شماره صفحه و شماره قاب در رجیستر انجمنی موجود باشد وضعیت برخورد (HIT) رخ می‌دهد در غیر این‌صورت وضعیت عدم برخورد (MISS) پدید می‌آید که نسبت این در وضعیت به‌نام **ضریب توفیق یا نسبت اصابت** (HIT RATIO) از رابطه زیر محاسبه می‌شود:

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

اگر زمان دسترسی به حافظه t_{mem} ۶۰ نانو ثانیه و زمان دسترسی به رجیسترهای انجمنی t_{ar} برابر ۵ نانو ثانیه و احتمال وجود شماره صفحه آدرس در رجیستر انجمنی برابر ۷۵ درصد باشد. آن‌گاه زمان دسترسی به حافظه نگاشته شده در موقعی که شماره قاب در رجیسترهای انجمنی یافت شود، ۶۵ نانوثانیه خواهدبود. در غیر این‌صورت، دسترسی به جدول صفحه در حافظه ۶۰ نانوثانیه و دسترسی به مکان مورد نظر در حافظه نیز ۶۰ نانو ثانیه و ۵ نانو ثانیه هم زمان تلف شده برای جستجو در رجیسترهای انجمنی خواهد بود که مجموعاً ۱۲۵ نانو ثانیه به طول می‌انجامد. لذا زمان **دسترسی موثر حافظه** به شکل زیر محاسبه می‌شود:

$$EAT = t_{ar} + HR * t_{mem} + (1 - HR) * 2 * t_{mem}$$

EAT= ۵ +۰/۷۵ * ۶۰+ ۰/۲۵ * ۲ *۶۰=۸۰ نانوثانیه



شکل ۶-۳

۶-۶-۳-۲- حفاظت در صفحه‌بندی

برای پیاده سازی حفاظت، می‌توان از چندین بیت در کنار هر سطر جدول صفحه استفاده نمود. این بیت‌ها قابل خواندن - نوشتن و اجرا بودن محتویات صفحه را مشخص می‌کنند. تلاش در نوشتن یک صفحه فقط خواندنی سبب می‌شود که سخت افزار به سیستم عامل وقفه دهد(**Trap یا تله** رخ می‌دهد)

۶-۶-۳-۳- اشتراک در صفحه‌بندی

اشتراک در صفحه‌بندی به مفهوم استفاده چند برنامه از صفحات مشترک و استفاده بهتر از حافظه است. دو یا چند فرآیند هنگامی می‌تواند کد یکسانی را هم‌زمان اجرا نمایند که از خاصیت تغییرناپذیری و چند مدخلی برخوردار باشد بدین معنی که در هنگام اجرا، کد مشترک غیرقابل تغییر و فقط خواندنی باشد(**Reentrant**). فقط یک کپی از کد مشترک لازم است در حافظه فیزیکی قرار گیرد و هر جدول صفحه فرآیند به کپی فیزیکی یکسان آن کد نگاشته می‌شوند، اما صفحات مربوط به فضاهای داده مربوط به فرآیندها در قاب‌های گوناگون نگاشته می‌شود.

تکنیک copy on write (cow)

این تکنیک ساخت سریعی از یک پردازش را ممکن می‌سازد و تعداد صفحات جدید تخصیص یافته به پردازش را به حداقل می‌رساند.
همان‌طور که می‌دانید فراخوانی سیستمی ( )fork پردازش فرزند را که یک کپی از پردازش پدر می‌باشد ایجاد می‌نماید. دستور مرسوم ( ) fork با ایجاد یک کپی از فضای آدرس‌دهی پدر برای فرزند (کپی صفحات متعلق به پدر) کار را شروع می‌کند. ایجاد یک کپی از فضای آدرس‌دهی پدر ضرورتی ندارد، زیرا بسیاری از فرآیندهای فرزند بلافاصله بعد از ( )fork فراخوانی سیستمی ( )exec را صدا می‌زنند.
به‌جای این عمل از روش شناخته شده‌ای به‌نام (COW) می‌توانیم استفاده نماییم. این روش در شروع کار صفحات موجود را بین پدر و فرزند به اشتراک می‌گذارد. سیستم این صفحات را به‌عنوان صفحات COW علامت‌گذاری می‌نماید، بدین معنی که چنان‌چه هر کدام از دو فرآیند در داخل صفحه اقدام به نوشتن کند، بلافاصله سیستم یک کپی از آن صفحه مشترک را ایجاد خواهد کرد.
برای مثال فرض کنید فرآیند فرزند بخواهد تغییری در صفحه مربوط به بخش Stack ایجاد نماید، سیستم چون این صفحه را به‌عنوان صفحه COW می‌شناسد بنابراین یک کپی از آن ایجاد شده و به فضای آدرس فرآیند فرزند نگاشت می‌شود. بنابراین فرآیند فرزند کپی را تغییر می‌دهد و صفحه متعلق به فرآیند پدر بدون تغییر باقی می‌ماند.
این تکنیک در بسیاری از سیستم‌های عامل مانند w2k ,linux, Solaris به هنگام کپی‌سازی از فرآیندها مورد بهره‌برداری قرار می‌گیرد.
مطلب دیگری که حایز اهمیت است، تخصیص صفحات آزاد در هنگام کپی صفحات COW می‌باشد.
بسیاری از سیستم‌های عامل مانند Solaris فضایی از صفحات آزاد را برای چنین محلی کنار می‌گذارد. این صفحات آزاد برای افزایش Stack, heap درخواست کپی در تکنیک COW به‌کار گرفته می‌شوند. با استفاده از روش Zero – fill – on- demand این صفحات تخصیص داده می‌شوند. قبل از تخصیص تمام صفحات پاک (zero – out) می‌گردند.
بعضی از نسخه‌های unix به‌جای فراخوانی سیستمی ( ) fork از ( ) vfork بهره می‌برند.
( )vfork (virtual Memory fork) متفاوت از ( ) fork به همراه تکنیک COW می‌باشد، بدین معنی که با اجرای این فراخوانی سیستمی فرآیند پدر معلق می‌گردد و فرآیند فرزند از فضای آدرس‌دهی پدر بهره می‌گیرد. **چون ( ) vfork از تکنیک COW استفاده نمی‌کند** چنان‌چه فرآیند فرزند تغییراتی در صفحات مربوط به فضای آدرس‌دهی پدر اعمال نماید، این تغییرات توسط فرآیند پدر قابل رویت می‌باشد. بنابراین ( ) vfork را با احتیاط باید به‌کار گرفت بدین معنی که مطمئن باشیم که فرآیند فرزند فضای آدرس‌دهی پدر را تغییر نمی‌دهد.
( )Vfork باید زمانی استفاده شود که بلافاصله بعد از آن دستور ( )exec در فرآیند فرزند فراخوانی گردد (زیرا کپی از صفحات گرفته نمی‌شود) این روش کاملاً کارآمد می‌باشد و گاهی برای پیاده‌سازی Interface مربوط به خط فرمان shell در سیستم unix مورد استفاده قرار می‌گیرد.

۶-۶-۳-۴ - جداول صفحه چند سطحی

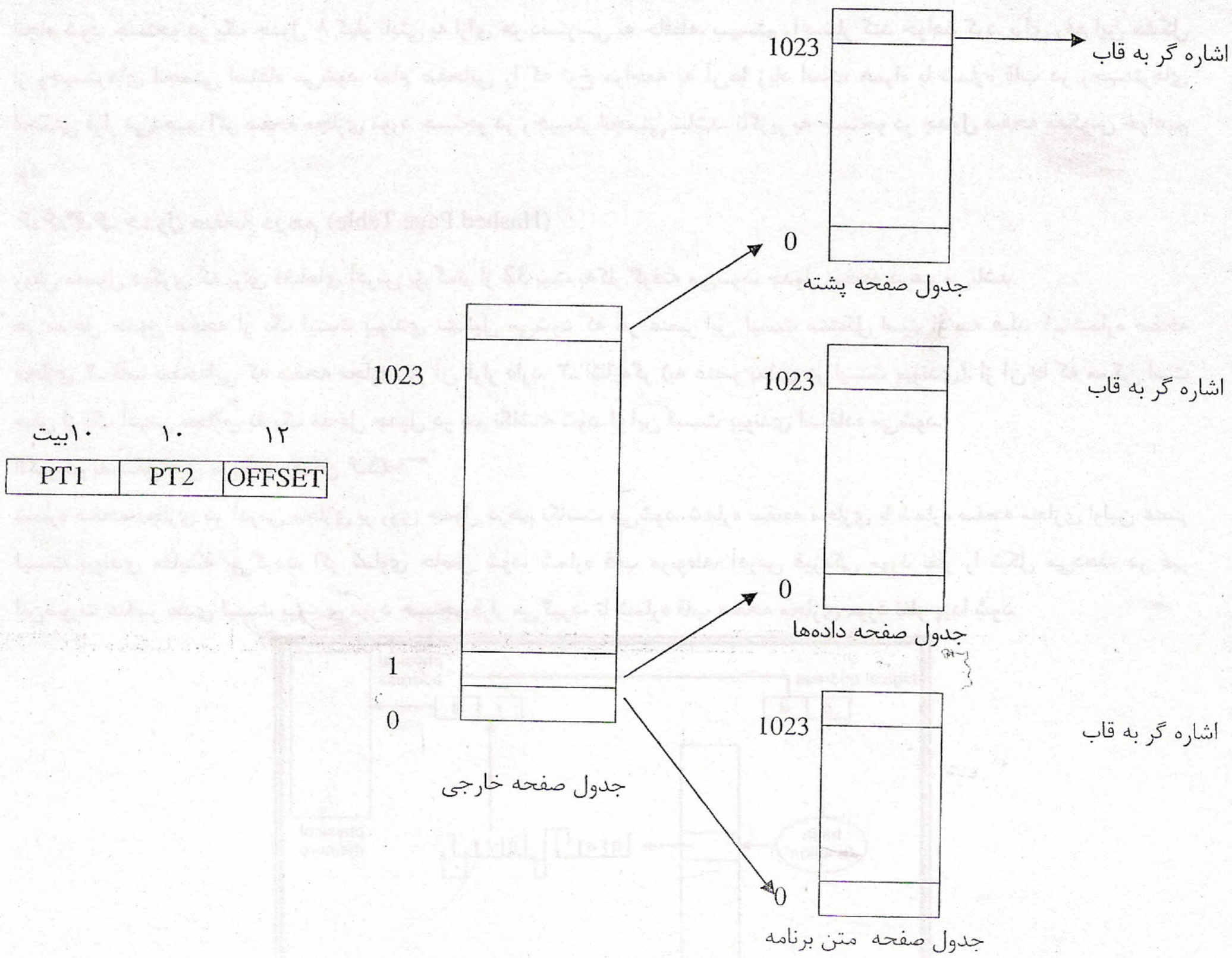
به منظور رفع مشکل داشتن جداول صفحه بزرگ در حافظه، بسیاری از سیستم‌ها از جداول صفحه چند سطحی استفاده می نمایند.
فرض کنید آدرس مجازی ۳۲ بیتی در کامپیوتری به ۳ قسمت متشکل از فیلد ۱۰ بیتی PT1 فیلد ۱۰ بیتی PT2 و فیلد ۱۲ بیتی انحراف (Offset) تقسیم شده باشد.
چون فیلد انحراف ۱۲ بیتی است، اندازه صفحات 2^{12} می‌باشند و فرآیند حداکثر از 2^{20} عدد صفحه می‌تواند برخوردار باشد. در این روش فقط جداولی که به آن‌ها احتیاج است در حافظه قرار می‌گیرند.

برای مثال فرض کنید فرآیندی به ۱۲ مگا بایت حافظه احتیاج دارد. ۴ مگابایت پایینی حافظه برای متن برنامه، ۴ مگابایت بعدی برای داده‌های مورد نیاز و ۴ مگا بایت بالائی برای پشته در نظر گرفته شده است و بقیه قسمت‌ها بی مصرف باشند.

در شکل ۶-۴ در این مثال یک جدول صفحه دو سطحی مورد استفاده قرار می‌گیرد. جدول سمت چپ، بالاترین سطح از جدول صفحه است که به **جدول صفحه خارجی** (جدول سطح اول) موسوم می‌باشد و از ۱۰۲۴ درایه تشکیل شده است. این درایه ها با فیلد ۱۰ بیتی PT1 در ارتباط هستند. هنگامی‌که آدرس مجازی گرفته می‌شود، واحد سخت افزار مدیریت حافظه ابتدا فیلد ۱۰ بیتی PT1 را استخراج کرده و مقدار آن را به عنوان اندیسی بر روی جدول صفحه خارجی مورد استفاده قرار می‌دهد. محتوای هر درایه از جدول صفحه خارجی (بالاترین سطح) که با اندیس PT1 مشخص می‌شود، شماره قاب صفحه برای یک جدول سطح ۲ خواهد بود.

در جدول صفحه خارجی، درایه شماره 0 به جدول صفحه متن برنامه، درایه شماره ۱ به جدول صفحه داده‌ها و درایه شماره ۱۰۲۳ به جدول صفحه پشته اشاره می‌کند و بقیه درایه ها این جدول بدون استفاده باقی می‌ماند.

فیلد PT2 به عنوان اندیس جدول صفحه سطح ۲ انتخاب شده به‌کار می‌رود تا شماره قاب خود صفحه مورد جستجو، پیدا شود.



شکل ۶-۴

۶-۶-۳-۵- جدول صفحه معکوس

در جداول صفحه که تاکنون درباره آن‌ها توضیح داده‌ایم باید به ازای هر صفحه مجازی، یک درایه وجود داشته باشد. بنابراین با فضای آدرس 2^{64} بایتی و صفحات 4K، اندازه جدول صفحه بیش از 10^{15} بایت خواهد بود. اختصاص یک میلیون گیگا بایت حافظه فقط برای جدول صفحه غیرممکن می‌باشد.

یک راه حل، استفاده از **جدول صفحه معکوس (Inverted Page Table)** است. به جای این‌که به ازای هر صفحه مجازی یک درایه در جدول صفحه داشته باشیم، **به ازای هر قاب در حافظه اصلی یک درایه در جدول صفحه معکوس وجود دارد.** برای مثال اگر آدرس مجازی ۶۴ بیتی باشد و صفحات ۴ کیلو بایتی فرض شود و ۳۲ مگا بایت حافظه اصلی در اختیار داشته باشیم، جدول صفحه معکوس دارای ۸۱۹۲ درایه خواهد بود. هر یک از این درایه ها مشخص می‌کند که کدام فرآیند و صفحه مجازی آن در قاب حافظه اصلی قرار گرفته است.

وقتی فرآیند K به صفحه مجازی P مراجعه می‌کند، سرتاسر جدول صفحه معکوس را برای یافتن درایه (K,P) باید جستجو نمود تا آدرس فیزیکی آن در حافظه پیدا شود. این جستجو باید نه تنها در هنگام وقوع نقص صفحه، بلکه به ازای هر دسترسی به حافظه نیز انجام شود. جستجو در یک جدول ۸ کیلو بایتی به ازای هر دسترسی به حافظه، سیستم را بسیار کند خواهد کرد. برای رفع این مشکل از رجیسترهای انجمنی استفاه می‌شود. تمام صفحاتی را که نرخ مراجعه به آن‌ها زیاد است، همراه با شماره قاب در رجیسترهای انجمنی قرار می‌دهیم. اگر صفحه مجازی مورد جستجو در رجیستر انجمنی نباشد، ناگزیر به جستجو در **جدول صفحه معکوس** خواهیم بود.

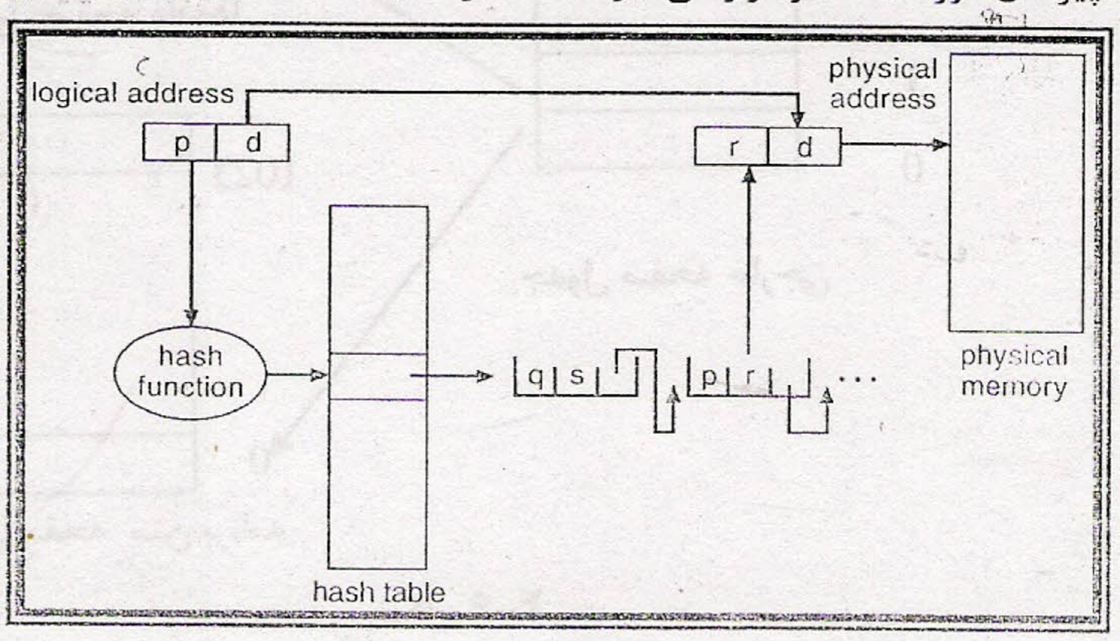
۶-۶-۳-۶- جدول صفحه درهم (Hashed Page Table)

روش معمول دیگری که برای فضاهای آدرس بزرگ‌تر از 32 بیت به‌کار گرفته می‌شود، جدول صفحه درهم می‌باشد.

هر مدخل جدول صفحه از یک لیست پیوندی تشکیل می‌شود که هر عنصر این لیست متشکل است از سه فیلد ۱- شماره صفحه مجازی ۲- قاب صفحه‌ایی که صفحه مجازی در آن قرار دارد. ۳- اشاره‌گر (به عنصر بعدی در لیست پیوندی). از آن‌جا که ممکن است بیش از یک آدرس مجازی به یک مدخل جدول در هم نگاشته شود. از این لیست پیوندی استفاده می‌شود.

الگوریتم به شکل ذیل می‌باشد (شکل ۶-۵):

شماره صفحه مجازی در آدرس مجازی بر روی جدول درهم نگاشت می‌شود. شماره صفحه مجازی با شماره صفحه مجازی اولین عنصر لیست پیوندی مقایسه می‌گردد، اگر تساوی حاصل شود، شماره قاب مربوطه، آدرس فیزیکی مورد نظر را شکل می‌دهد، در غیر این‌صورت عناصر بعدی لیست پیوندی مورد جستجو قرار می‌گیرد، تا شماره قاب صفحه مجازی مورد نظر پیدا شود.



شکل ۶-۵

۶-۶-۴- مدیریت حافظه قطعه‌بندی ساده

یک قطعه را می‌توان به عنوان مجموعه منطقی از اطلاعات تعریف کرد؛ به‌عنوان مثال، زیرروال، آرایه ناحیه داده‌ای می‌توانند به‌عنوان **قطعه** محسوب گردند. بنابراین فضای آدرس هر فرآیند در واقع متشکل از اجتماعی از قطعات می‌باشد. در نتیجه قطعه‌بندی روشی است که برای مدیریت این قطعات به کار می‌رود.

قطعات فرآیند دارای اندازه یکسان نمی‌باشند اگر چه حداکثری برای طول قطعه در نظر گرفته می‌شود. همانند صفحه‌بندی، آدرس منطقی در قطعه‌بندی شامل دو بخش شماره قطعه و انحراف است. در قطعه‌بندی ساده به‌خاطر عدم به کارگیری حافظه مجازی، لازم است کلیه قطعات برنامه برای اجرا در حافظه بار گردند. قطعه‌بندی تکه تکه شدن داخلی را حذف می‌کند و همانند بخش بندی پویا، دارای اشکال **تکه تکه شدن خارجی** است. تفاوت آن با بخش بندی پویا در این است که یک برنامه می‌تواند بیش از یک بخش را اشغال نماید و لزومی ندارد این بخش‌ها پیوسته در حافظه قرار گیرند. در حالی که **صفحه‌بندی از دید برنامه ساز مخفی** است.

قطعه‌بندی قابل رویت و عامل تسهیل سازماندهی برنامه‌ها و داده‌ها می‌باشد.

به طور طبیعی برنامه کاربر، ترجمه می‌شود و مترجم به طور خودکار قطعاتی را که منعکس کننده برنامه ورودی است بنا می‌سازد و بارکننده، تمام قطعات تولید شده را گرفته و شماره قطعه را به آن‌ها نسبت می‌دهد.

طرح قطعه‌بندی ساده همانند صفحه‌بندی، از یک جدول قطعه برای هر فرآیند و جدولی از بلوک‌های آزاد در حافظه اصلی استفاده می‌کند. هر مدخل جدول قطعه، طول قطعه و آدرس شروع قطعه مورد نظر در حافظه اصلی را مشخص می‌نماید. .

زمانی‌که فرآیندی به حالت اجرا می‌رود آدرس جدول قطعه فرآیند در ثبات خاصی که مورد استفاده سخت‌افزار حافظه است، بار می‌شود.

مراحل زیر برای ترجمه آدرس منطقی n+m بیتی (که در آن n بیت سمت چپ شماره قطعه و m بیت سمت راست شماره انحراف هستند) به آدرس فیزیکی توسط سخت افزار صورت می‌پذیرد.

- استخراج شماره قطعه از n بیت سمت چپ آدرس منطقی
- استفاده از شماره قطعه به‌عنوان شاخص به جدول قطعه فرآیند برای یافتن آدرس فیزیکی شروع آن قطعه (درصورت وجود رجیسترهای انجمنی همانند صفحه‌بندی عمل می‌شود.)
- مقایسه انحراف موجود در m بیت سمت راست با طول قطعه، اگر انحراف بزرگ‌تر یا مساوی طول قطعه باشد، آدرس معتبر نیست (کنترل با ایجاد تله به سیستم عامل واگذار می‌شود).
- مجموع آدرس فیزیکی شروع قطعه و انحراف، آدرس فیزیکی مورد نظر را تولید می‌نماید.

توجه داشته باشید که در مواردی که برنامه شامل تعداد زیادی قطعه باشد، نمی‌توان جدول قطعه را در رجیسترها نگه داشت، بنابراین باید در حافظه نگه داشته شود.

راه حل طبیعی آن است که یک مجموعه رجیسترهای انجمنی برای نگهداری درایه های جدول قطعه که اخیراً مورد استفاده واقع شده‌اند، به کار گرفته شود. مجموعه کوچکی از رجیسترهای انجمنی می‌تواند کلاً زمان لازم برای دست‌یابی حافظه را کمتر از ۱۵ درصد کندتر از دست‌یابی مستقیم به حافظه کاهش دهد.

۶-۶-۴-۱ حفاظت در مدیریت قطعه‌بندی

دست‌یابی به هر قطعه ای باید تحت کنترل باشد. برای مثال جدولی با محتوای مقادیر ثابت، باید محدود به دست‌یابی از نوع خواندن گردد در حالی که دست‌یابی به یک قطعه از نوع فضای فرآیند می‌تواند به صورت خواندن و نوشتن باشد. یک قطعه متعلق به یک رویه محض به دلیل خرابی ناگهانی ان نباید امکان نوشته شدن داشته باشد و ضمناً به این سبب که امکان به سرقت رفتن از رویه می‌رود، دست‌یابی از نوع خواندن به این قطعه نیز غیرمجاز خواهد بود. بنابراین دست‌یابی به چنین قطعه ای باید فقط از نوع اجرایی باشد.

در یک محیط قطعه‌بندی شده مشترک، دست‌یابی از نوع خواندن، نوشتن و یا اجرا می‌تواند براساس کاربرد قطعه مشترک به کاربران قطعه اعطا گردد.

۶-۶-۴-۲- اشتراک در مدیریت قطعه‌بندی

اشتراک، یکی از مزایای مدیریت قطعه‌بندی می‌باشد. هر فرآیند، در زمان اجرا جدول قطعه منحصر به فرد خود را برای تبدیل آدرس منطقی به آدرس فیزیکی به کمک سخت افزار به‌کار می برد. زمانی که درایه های جداول قطعه متعلق به دو فرآیند، به مکان‌های فیزیکی یکسانی اشاره نمایند، می‌گویند آن قطعات به اشتراک گذاشته شده اند. بنابراین هر اطلاعی که به صورت یک قطعه تعریف شده باشد می‌تواند به اشتراک گذاشته شود، زیرا اشتراک در سطح قطعه رخ می‌دهد.

برای مثال بسته های برنامه‌های فرعی رایج، می‌تواند مابین بسیاری از کاربران به اشتراک گذاشته شوند، اگر به صورت قطعات مشترک فقط خواندنی قابل تعریف باشند.

۶-۶-۵- قطعه‌بندی با صفحه‌بندی

در جایی که میانگین اندازه قطعه، بزرگ باشد تکه تکه شدن خارجی مساله می‌شود. از طرفی دیگر، زمان جستجو برای تخصیص قطعه، با استفاده از روش First Fit یا Best Fit می‌تواند طولانی باشد. بنابراین ممکن است به علت تکه تکه شدن خارجی، حافظه از دست بدهیم و یا به علت جستجوهای طولانی زمان تلف شود و یا هر در مورد رخ دهد.

راه حل این مشکل، صفحه‌بندی قطعات است بدین معنی که قطعات تولید شده توسط مترجم، صفحه‌بندی می‌شوند و به ازا، هر فرآیند یک جدول قطعه و به ازای هر قطعه یک جدول صفحه جداگانه ایجاد می‌گردد. تعداد درایه های هر جدول صفحه بستگی به اندازه قطعه مربوطه دارد. همانند صفحه‌بندی، آخرین صفحه هر قطعه معمولاً پر نمی‌گردد و به ازای **هر قطعه به طور میانگین نصف صفحه تکه تکه شدن داخلی خواهیم داشت.**

برای مثال سیستم مالتیکس با آدرس منطقی ۳۴ بیتی را در نظر بگیرید که در آن شماره قطعه ۱۸ بیتی و انحراف ۱۶ بیتی است.

به‌کارگیری مدیریت حافظه قطعه‌بندی در این سیستم به‌خاطر اندازه بزرگ قطعه که می‌تواند تا 2^{16} بایت باشد، مشکلاتی را که در فوق متذکر شده ایم، در بر می‌گیرد. بنابراین سیستم از مدیریت حافظه قطعه‌بندی صفحه‌بندی استفاده می‌نماید. نگرش این مدیریت در این سیستم بدین گونه است که انحراف قطعه به دو بخش ۶ بیت شماره صفحه و ۱۰ بیت انحراف صفحه تجزیه می‌شود. جدول صفحه برای هر قطعه می‌تواند حداکثر از 2^{6} درایه برخوردار باشد و هر فرآیند حداکثر می‌تواند تا 2^{18} قطعه را در بر گیرد.

انحراف از صفحه (۱۰ بیت)	شماره صفحه (۶ بیت)	شماره قطعه (۱۸ بیت)