فصل چهارم

مدیریت حافظه (قطعه بندی و صفحه بندی)

برای برنامه نویسان، حافظه ایده آل بی نهایت بـزرگ، سـریع، ارزان و غیرفـرار اسـت (یعنـی بـا رفـتن بـرق اطلاعات آن از بین نمی رود) ولی تکنولوژی امروزی چنین حافظه ای را در اختیار ما قرار نمی دهد. لذا اکثر کامپیوتر های امروزی از یک سلسله مراتب حافظه استفاده می کنند. یکی حافظه اصلی یا RAM می باشد که سریع است ولی گران بوده و با رفتن برق اطلاعات آن از بین نمی رود ولی کند است. بخشی از سیسـتم عامل که سلسله مراتب حافظه را اداره می کند مدیر حافظه (Memory Manager) نامیـده مـی شـود. ایـن بخش باید بداند که کدام قسمت حافظه خالی و کدامیک استفاده شده است. همچنین هنگامی کـه چنـدین برنامههمزمان می خواهند با هم اجراء شوند بایستی همگی به حافظه آورده شوند و این وظیفه سیستم عامل است که از تداخل آنها در حافظه جلوگیری کند. از طرف دیگر اگر مقدار RAM برای اجرای برنامه ها کافی نباشد، سیستم عامل می تواند دیسک به عنوان یک حافظه مجازی برای این منظور اسـتفاده کنـد کـه ایـن موضوع را در فصل بعدی شرح می دهیم.

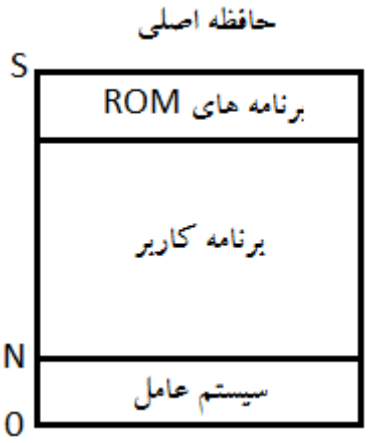
در این فصل طرحهای گوناگون مدیریت حافظه را از ساده ترین حالات شروع می کنیم تا به حالات کامـل و پیچیده تر برسیم. لازم به ذکر است همگام با پیشرفت های سخت افزاری که حافظـه را ارزان و حجـم آن را زیادکرده است، برنامه ها نیز بزرگتر می شوند و طبق قانون پارکینسون (parkinson) «برنامه ها آنقدر رشـد می کنند که کل حافظه را پر می سازند.»

سیستم مدیریت حافظه به طور کلی می تواند به دو دسته تقسیم گردد. دسته اول آنهائی که پردازش هـا را بین حافظه اصلی و دیسک جا به جا می کنند (تکنیک های مبادله و صفحه بندی) و دسته دوم آنهایی کـه از تکنیک های مبادله و صفحه بندی استفاده نمی کنند. نوع دوم سیستم های ساده تری بـوده و مـا بحـث خود را از آنها آغاز می کنیم. موضوع مبادله و صفحه بندی زمانی مطرح می گردد کـه حافظـه اصـلی جـای کافی برای نگهداری تمام پردازش ها را به طور همزمان ندارد.

۱- تگ برنامگی ساده

ساده ترین طرح آن است که در هر لحظه فقط یک برنامه در حال اجراء باشد و هنگامی که ایـن برنامـه بـه حافظه آورده می شود کل حافظه را در اختیار خود می گیرد. اگر حافظه RAM به اندازه کافی در دسـترس نباشد برنامه اجراء نمی شود. سیستم عامل DOS اولیه اینگونه بوده است. طرح حافظه د این سیستم عامـل اولیه به صورت شکل زیر می باشد:

یعنی خود سیستم عامل بخش اولیه حافظه را اشغال کرده و بخش بالایی حافظه نیز برای برنامه های موجود در ROM (از جمله BIOS سیستم) و موارد دیگر استفاده می شود و بقیه حافظه در اختیار برنامه ها قرار می گیرد. مثلاً در کامپیوترهای اولیه DOS که حداکثر یک مگابایت حافظه داشتند حافظه باقی مانده برای برنامه ها حدود 540 کیلوبایت و یا حتی کمتر بود.



۲- تک برنامه‌ای با سیستم oecrlay

در این سیستم مدیریت پردازش می تواند بزرگتر از حافظه اصلی باشد. در شیوه overlay (جایگذاشت) برنامه به بخشهای مختلفی تقسیم شده و تنها آن داده و دستوراعمل هایی را در حافظه قرار می دهیم که در هر زمان مفروض مورد نیاز هستند و بقیه بخشها در دیسک باقی می ماند. هنگامی که به بخش دیگری از آن برنامه نیاز داریم، قسمتی که موردنیاز نیست از حافظه خارج شده و بخش مورد نیاز به حافظه آورده می شود. Overlay به حمایت سخت افزاری ویژه ای نیاز ندارد. این تکنیک را خود برنامه نویس می بایست در برنامه پیاده سازی کند.

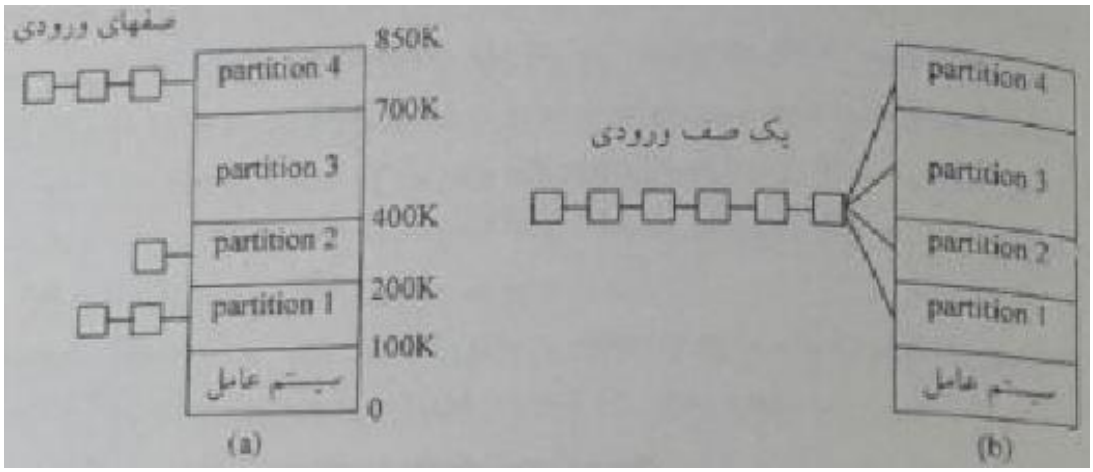
تکنیک overlay فقط در میکروکامپیوترهای اولیه و تحت سیستم عاملهای ساده ای مثل DOS بسیاری از برنامه های exe به علتی بزرگی نیاز به یک فایل دیگر با پسوند ovl نیز داشتند.

۳- چندبرنامگی با بخش بندی ثابت حافظه

همانطور که می دانید سیستم عامل های امروزی همگی چندبرنامگی هستند. پس باید بتوان چند برنامه را همزمان در حافظه بار کرد به گونه ای هیچگونه تداخلی باهم نداشته باشند.

ساده ترین روش برای چندبرنامگی این است که حافظه را نسبت به N قسمت تقسیم کنیم. اندازه هر قسمت می تواند با بخشهای دیگر متفاوت باشد این کار می تواند در هنگام شروع کار سیستم به صورت دستی توسط اپراتور انجام شود. وقتی که یک کار (Job) وارد می شود در یک صف ورودی قرار می گیرد تا در

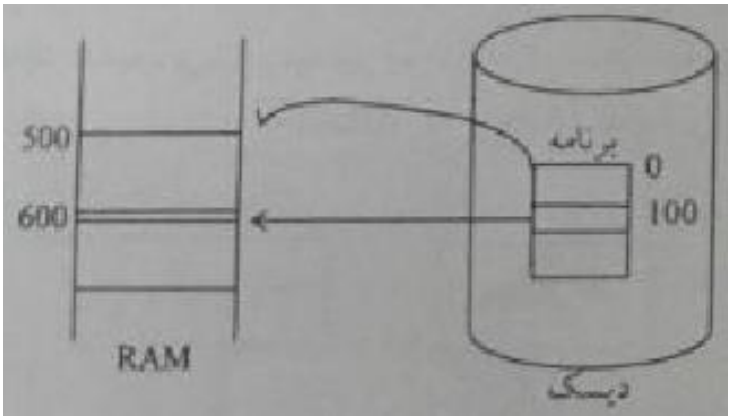
کوچکترین بخشی که مناسب آن است قرار داده شود. البته ممکن است آن بخش دقیقاً هم اندازه برنامه نبوده و بدین ترتیب مقداری از فضای آن از بین برود. در این روش می توان برای هر پارتیشن از یک صف مجزا استفاده کرد و یا اینکه فقط یک صف برای تمام پارتیشن ها داشت (شکل زیر):



این روش با پارتیشن های ثابت که توسط اپراتور تعریف می شد. در سیستم عامل IBM OS/360 بر روی مین فیرمهای IBM استفاده می شد. این سیستم به راحتی پیاده سازی می شود ولی در سیستم عاملهای امروزی از آن استفاده نمیشود چرا که باعث اتلاف زیاد حافظه می گردد.

تبدیل آدرس ها

در اکثر سیستم ها برنامه کاربر می تواند در هر قسمت حافظه فیزیکی قرار بگیرد بنابراین متغیری ک مثلاً در آدرس 100 از ابتدای یک برنامه قرار گرفته است ممکن است هنگام اجراء در آدرس واقعی 600 حافظه قرار بگیرد.



هر پیوند (bind) در واقع نگاشتی (تبدیلی) از یک فضای آدرس به فضای آدرس دیگر است.

پیوند آدرس های حافظه می تواند به یکی از سه صورت زیر باشد:

۱- زمان کامپایل: اگر در موقع کامپایل معلوم باشد که برنامه در کجای حافظه قرار خواهد گرت، در این صورت کد مطلق می تواند تولید شود یعنی آدرس های ذکر شده در برنامه هنگام بار شدن و یا هنگام اجراء تغییر نخواهد کرد و تصویر آینه وار برنامه در دیسک عیناً به حافظه آورده شده و اجرا می گردد و مثلاً آدرس 100 ذکر شده در برنامه همان آدرس 100 مطلق حافظه RAM می باشد. برنامه های COM تحت سیستم عامل DOS اینگونه هستند.

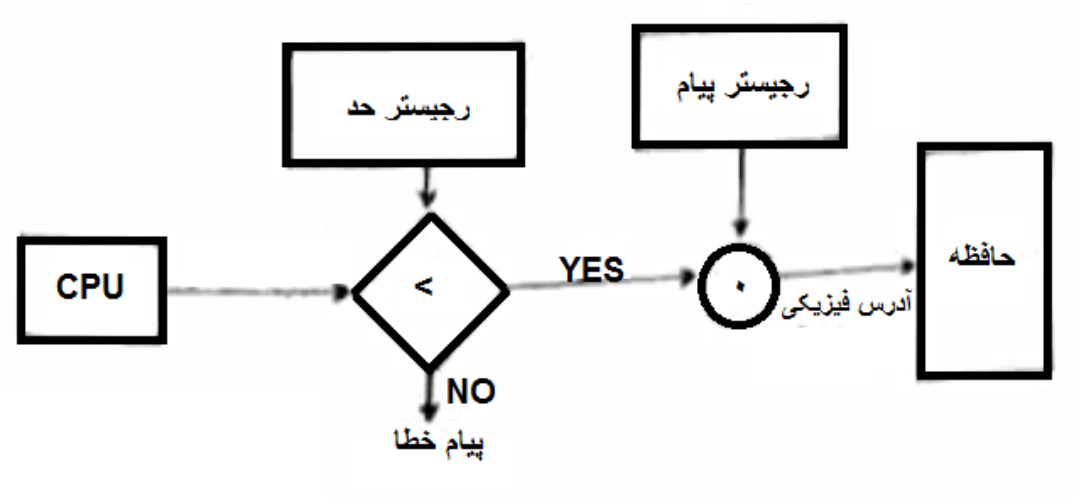
۲- زمان بارگذاری اگر در زمان کامپایل معلوم نباشد که برنامه در کجای حافظه قرار خواهد گرفت آنگاه کامپایلر بایستی که قابل جابجائی (Relocacable) تولید کند. مثلاً در برنامه های EXE تحت DOS. برنامه یک لیست در ابتدای فایل EXE می سازد. در آن لیست مکانهایی در برنامه exe. که حاوی آدرسهای جابه-جا شونده هستند مشخص می گردد. مثلاً اگر در آن مکان مشخص شده آدرس 50 وجود دارد و برنامه هنگام بار شدن از آدرس 600 در حافظه قرار داده شود آدرس 50 تبدیل به آدرس 650 می شود و اگر در آدرس 800 بار شودآدرس مذکور 850 می شود.

۳- زمان اجراء: اگر پردازش در حین زمان اجرایش یتواند در حافظه جابه‌جا شود، آن گاه پیوند دادن بایستی از زمان اجرا به تأخیر انداخته شود. برای این حالت نیاز به سخت افزار خاصی وجود دارد.

آدرس فیزیکی و منطقی (logical and physical address)

آدرس تولید شده توسط CPU را آدرس منطقی می گویند. در حالیکه آدرس مشاهده توسط واحد حافظه (یعنی آنچه که در رجیستر آدرس حافظه بار می شود) را آدرس فیزیکی می نامند. در حالی که پیوند آدرسهای حافظه در زمان اجراء باشد. به آدرس منطقی، آدرس مجازی (Virtual Address) نیز گفته می شود.

زمان اجرای آدرسهای مجازی به فیزیکی توسط واحد سخت افزاری مدیریت حافظه (Memory Management Unit) در پردازنده 8088 این واحد به طور مجزا و مشخص وجود ندارد ولی در 386 به بعد واحد مجزائی به نام MMU در ریزپردازنده ها وجود دارد یک طرح ساده از سیستم MMU که اجازه می دهد برنامه حین اجراء بتواند در حافظه جابه‌جا شود (پیوند آدرس در زمان اجراء) به صورت شکل زیر از دو ثبات پایه (Base) و حد (Limit) استفاده می کند. این سیستم برنامه ها را از تداخل یکدیگر محافظت می کند.



مثلاً فرض کنید اندازه یک پردازش 500 باشد حال اگر این پردازش در آدرس 2000 حافظه بار شود هنگام اجرا این عدد2000 در ثبات پایه و عدد 500 در ثبات حد قرار داده می شود. هنگامی که مثلاً با دستور Call 100 برنامه بخواهد به آدرس منطقی 100 دسترسی پیدا کند ابتدا چون 500>100 است این دستور قبول شده و سپس عدد 2000 با 100 جمع شده و آدرس 2100 به سمت حافظه فرستاده می شود اگر برنامه مثلاً با دستور Call 600 بخواهد به آدرسی بیرون از محدوده خود دسترسی پیدا کند سخت افزار جلوی کار آن را می گیرد. در سیستم عامل های multitasking هنگام تعویض متن رجیسترهای حد و پایه با مقادیر پردازش در حال اجرای جدید پر می شوند.

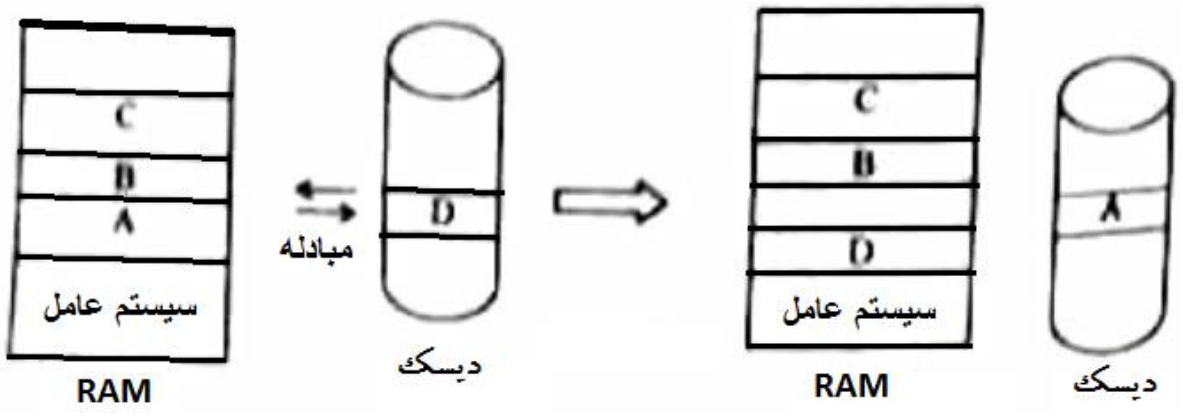
نکته ۱ : اصل محلی بودن مراجعات (Localy of References) در عمل بیشتر آدرس دهی های یک برنامه نزدیک به هم و احتمالاً به دنبال همدیگر هستند. این موضوع به نام «اصل محلی بودن مراجعات» معروف است.

نکته ۲ : اغلب سیستم ها حداقل دارای دو مد اجرائی مختلف (privilege level) هستند. مثلاً سیستم عامل در سطح 0 که دارای اولویت بالاتری است اجرا می شود و برنامه های کاربران در سطح یک که اولویت کمتری دارد اجراء می گردند. در برنامه های کاربر هرگونه تلاش برای دستیابی به محلهائی که وجود ندارند یا آدرسهای موجود در سیستم عامل، منجر به وقفه می گردد. یعنی سیستم محافظتی رجیستر حد که در شکل قبلی ترسیم شده برای برنامه های سطح کاربر فعال است. ولی برنامه های سیستمی که در سطح صفر اجراء می شوند آزادند که به تمامی حافظه دسترسی داشته باشند. لذا قسمت چک کردن رجیستر حد در این مد غیر فعال می گردد.

تذکر: دستورات ممتاز (Privileged) دستوراتی هستند که برنامه های سیستمی استفاده شده و می توانند وضعت سیستم را تغییر دهند.

۴- مبادله Seapping

بعضی وقتها حافظه اصلی به اندازه ای نیست که بتواند تمامی برنامه هـای در حـال اجـراء را در خـود جـای دهد. در این موارد می توان از تکنیک مبادله استفاده کرد. در سیستم مبادله هر پروسسی «به طـور کامـل» به حافظه اصلی آورده می شود، در آنجا برای مدتی اجرا می گردد و سپس دوبـاره بـه دیسـکبرگردانده مـی شود (تکنیک دیگری به نام حافظه مجازی وجود دارد که در آن می توان فقط بخضی از برنامه را از دیسـک به حافظه آورد. این تکنیک را در فصل بعد شرح می دهیم). شکل زیر این موضوع را نشان می دهد:



ایراد اساسی این روش کند بودن آن سات. فرض کنید برنامه کاربر 100K باشد و سرعت انتقال دیسـک نیـر یک مگابایت در ثانیه باشد. بنابراین انتقال برنامه 100K از یا به حافظه 100 میلی ثانیه طول می کشد. اگـر زمان جستجوی شیار و سکتور موردنظر را جمعاً حدود 8 میلی ثانیه فرض کنیم، کل زمان مبادلـه -100)*2 216 = (8 ملی ثانیه می شود که زمان نسبتاً زیادی است. از طرف دیگر کوانتوم زمـانی بایسـتی بـه مراتـب بزرگتر از 216 میلی ثانیه باشد، در چنین سیسستمی درصد به کارگیری CPU برابر است با:

$$\text{درصد بکارگیری CPU} = \frac{\text{زمان } CPU}{\text{زمان CPU} + 2 \times \text{زمان مبادله}} \times 100$$

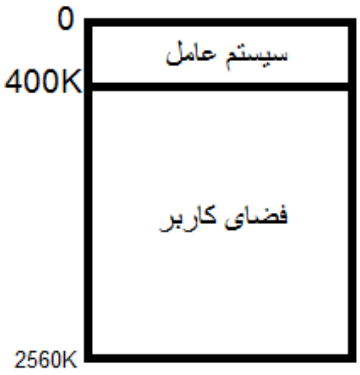
ایراد دیگر آن است که بر اثر عملیات مبادله ممکن است حفره های متعددی در حافظه بوجود آید که شکل فوق این موضوع را نشان می دهد. البته برای رفع این مشکل می توان در زمانهای معـین حافسـه را فشـرده سازی کنیم (Memory Vompaction). یعنی مثلاً بررسی ها را تا حد امکان به پائین شیفت دهـیم تـا یـک فضای بزرگ به اندازه مجموع حفره های کوچک پدید آید اما این کار نیز وقت CPU را هدر می دهد. بـرای مثال در یک کامپیوتر که 32 مگابایت حافظه دارد و می تواند شانزده بایت را در یک میکروثانیه کپـی کنـد. جهت فشرده سازی همه حافظه دو ثانیه از وقت CPU را می گیرد که زمان نسبتاً زیادی است. از طرف دیگر برای فشرده سازی کد برنامه ها باید جابه‌جا پذیر (Relocatable) باشد.

امروزه از تکنیک مبادله به صورت فوق به ندرت استفاده می شود و به جای آن از روش های مبادله کاملتری مثل حافظه مجازی که بعداً بیان خواهیم کرد استفاده می گردد.

۵- تخصیص چندبخشی به صورت همجوار

مثال زیر این روش را نشان می دهد.

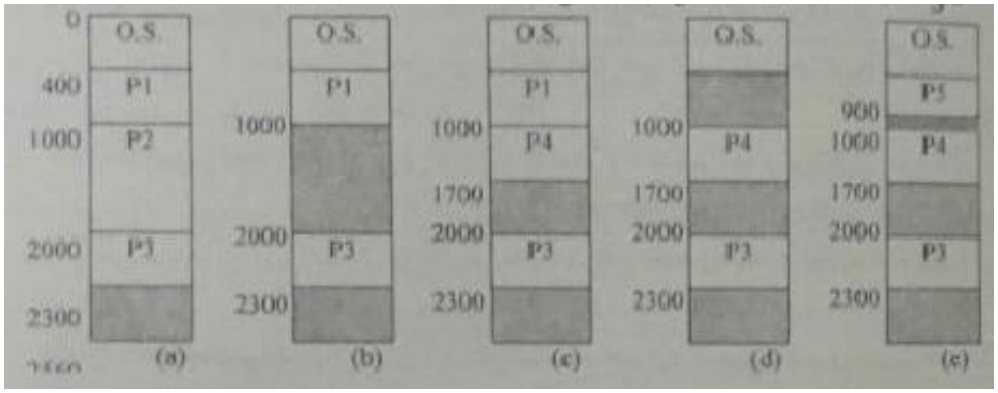
حافظه ای با اندازه 2560K داریم که 400K اول آن را سیستم عامل استافده کرده است پس 2160K حافظه آزاد داریم. فرض کنید 5 پردازش با مشخصات زیر را بخواهیم اجرا کنیم:



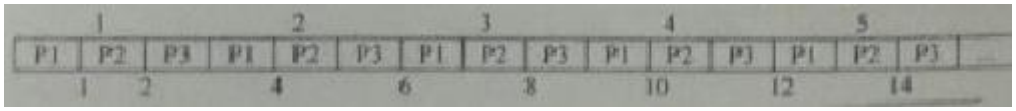
Job query		
پردازش	مقدار حافظه موردنیاز	زمان موردنیاز
P1	600K	10
P2	1000K	5
P3	300K	20
P4	700K	8
P5	500K	15

فرض کنید زمانبندی کار از نوع FCFS باشد یعنی برنامه ها به ترتیب ذکر شده از دیسک به RAM بار شوند. همچنین فرض کنید زمانبندی CPU از نوع نوبت گردشی با کوانتوم ۱ باشد. یعنی برنامه های موجود در حافظه ی RAM هر کدام ۱ واحد زمانی CPU را در اختیار می گیرند و هر بار ثباتهای پایه و حد با مقادیر مناسب هر پردازش پر می شوند.

ابتدا برنامه های P1 و P2 و P3 در حافظه بار شده و شروع به اجرا می شوند. توجه کنید به علت نبودن حافظه کافی P4 و P5 در حافظه بار نمی شوند (شکل a)



با کوانتوم زمانی ۱ پردازش P2 در زمان 14 تمام شده و حافظه خود را رها می کند (شکل b) و پردازش P4 در حافظه قرار داده می شود (شکل c)



سپس به همین ترتیب در زمان 28 پردازش P1 تمام شده و به جای آن پردازش P5 وارد حافظه مـی شـود. پس در این روش در هر زمان نسبتی از اندازه هـای بلوکهـای آزاد و صـف ورودی داریـم. پـردازش هـا وارد حافظه شده و تا هنگامی که حافظه مورد نیاز پردازش های بعدی در دسترس نباشد، سیستم عامل صبر می کند تا بلوک آزادی به اندازه کافی پدید آید. البته سیستم عامل می تواند در این حالت صف Job بـه دنبـال پردازشی بگردد که در یکی از بلوک های آزاد بتواند اجراء شود.

سؤال مهمی که در اینجا مطرح می شود این است که اگر چندین حفره آزاد وجود داشـته باشـد و پـردازش موردنظر بتواند در هر یک از آنها اجراء شود، سیستم عامل پردازش مذکور را در کدام حفره باید اجراء کنـد؟ این عمل که به عنوان روشهای تخصیص حافظه (به صورت همجوار) مطرح می شود. مـی توانـد بـه صـورت یکی از روشهای زیر انجام گیرد:

روشهای تخصص حافظه

۱- First Fit (اولین مناسب) : در این روش سیستم عامل با دریافت یک تقاضاس حافظه هنگـام بـاز کـردن یک برنامه، در لیست فضاهای خالی حافظه اولین بخش آزاد را که به اندازه نیاز اعلام شده، گنجایش داشـته باشد تخصیص می دهد. شروع جستجو برای یافتن فضای آزاد مناسب همواره از ابتدای لیسـت صـورت مـی گیرد. بدین ترتیب تراکم فضای اشغال شده در اول حافضه بیشتر خواهد بود.

۲- Next Fit (مناسب بعدی): این روش مانند First Fit است با این تفاوت که جستجو از محلی در لیسـت آغاز می شود که آخرین بار تخصیص از آن محل صورت گرفته است. بدین ترتیب یکنواختی توزیع برنامه ها در سطح حافظه نسبت به روش قبلی بیشتر خواهد شد.

۳- Best Fit (بهترین مناسب): در این روش کل لیست فضاهای آزاد جستجو شده و کوچکترین حفره که به اندازه کافی بزرگ است به پردازش تخصیص داده می شود. این روش باعث می شود که کوچکترین حفره بر اثر تخصیص باقی بماند. با این روش فضاهای بزرگتر برای تقاضای بیشتر حفظ می شوند. از آنجـا کـه تمـام لیست بلاکهای آزاد باید بررسی شود، این تکنین قدری زمانبر است.

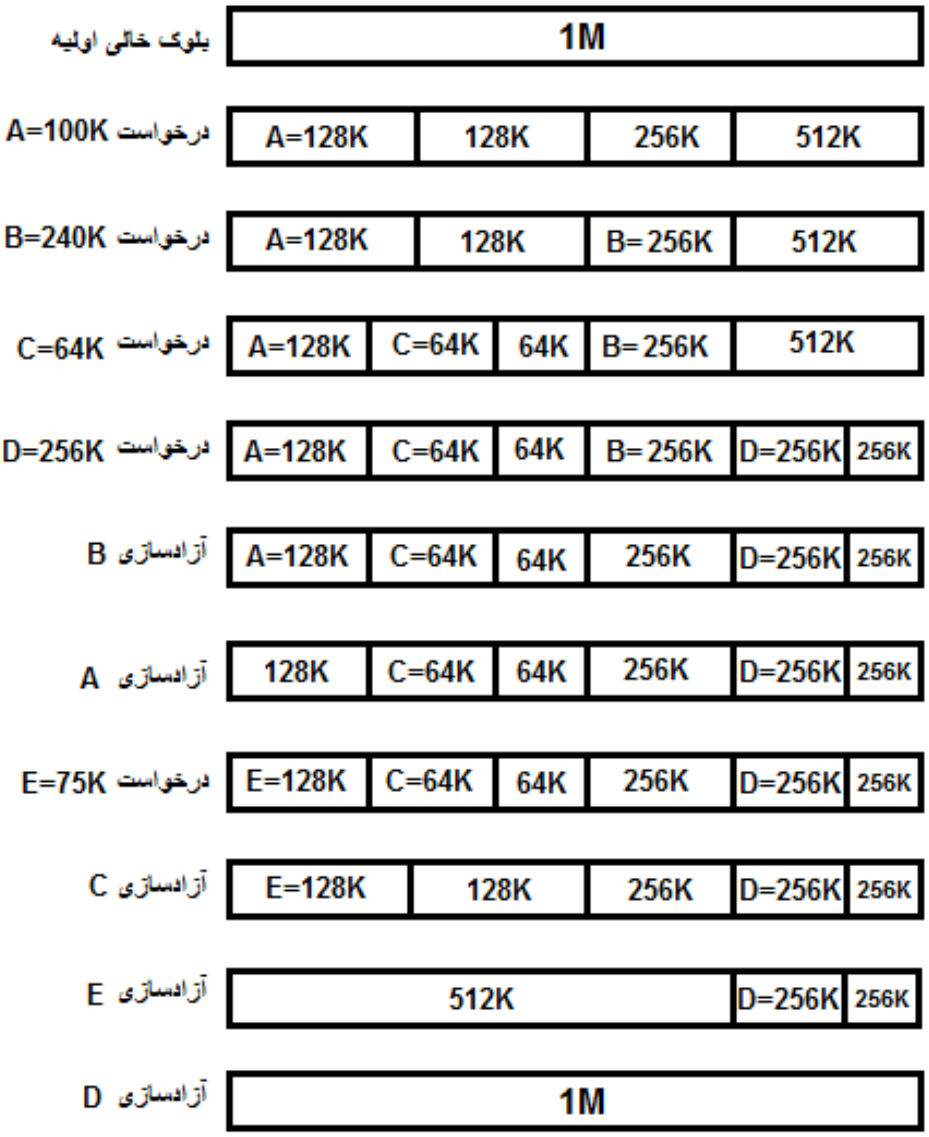
۴- Worst – Fit (بدترین مناسب): در این روش کل لیست فضاهای آزاد جستجو شده و بزرگترین حفره موجود به پردازش تخصیص داده می شود کنطق این روش آن است که حفره‌ی باقی مانده بتوان برای پردازشهای دیگر استفاده کرد ولی در روش Best Fit معمولاً حفره های باقی مانده آنقدر کوچک است که دیگر قابل استفاده برای پردازشهای دیگر نمی باشد. ایراد این تکنیک آن است که امکان دارد، تقاضاهایی که ناحیه بزرگی می خواهند، دیگر نتوانند برآورده شوند چرا که بلاکهای بزرگ که زودتر تخصیص یافته و کوچک می شوند.

شبیه سازیها نشان داده اند که First Fit و Best Fit از نظر سرعت و بهره وری حافظه بهتر از بقیه روشها عمل می کند. البته First Fit سریعتر از Best Fit است. شبیه سازی های انجام شده تویط بیز (Beyz) در سال ۱۹۷۷ نشان می دهد که کارایی Next Fit کمی کمتر از First Fit است.

۵- Quick Fit: در این روش برای هر دسته از پروسس هایی با اندازه های متداول یک لیست جداگانه تهیه می شود. مثلاً این الگوریتم دارای جدولی با n خانه لست که اول اشاره گری به ابتدای لیستی شامل حفره های چهار کیلوبایتی است. خانه دوم به لیست حفره های ۸ کیلوبایتی و خانه سوم به لیست حفره هی ۱۶ کیلوبایتی اشاره می کند و الی آخر. با این الگوریتم پیدا کردن حفره ای با اندازه مناسب بسیار سریع است ولی عیب آن این است که اگر پروسسی خاتمه یابد فضای آزاد شده آن به لیست مناسب اضافه شود که این کار زمان‌بر می باشد.

۶- الگوریتم رفاقتی یا Buddy: در این روش همه بخشهای آزاد حافظه به قطعاتی که همگی توان ۲ هستند، تقسیم می شوند (مثل بلکوهای آزاد 8K, 4K, 2K, 1K و ...) و برای هر گروه یک لیست جداگانه در نظر گرفته می‌شود. بدین ترتیب جهت تخصیص یک بلاک تنها باید بلاک موردنظر را از لیست مناسب خارج کرد. پس از تخصیص اگر فضای باقی مانده آن بلاک، توانی از ۲ باشد در لیست مربوطه‌اش قرار می گیرد و در غیر اینصورت به چندین بخش که اندازه هر کدام توانی از ۲ می باشد تقسیم می شود. از طرف دیگر در این روش بلوکهای کتار هم می توانند با هم ترکیب شده و بخش بزگرتری را پدید بیاورند.

مثال زیر مراحل این الگوریتم را نشان می دهد:


بلوک خالی اولیه
1M
درخواست A=100K
A=128K
128K
256K
512K
درخواست B=240K
A=128K
128K
B=256K
512K
درخواست C=64K
A=128K
C=64K
64K
B=256K
512K
درخواست D=256K
A=128K
C=64K
64K
B=256K
D=256K
256K
آزادسازی B
A=128K
C=64K
64K
256K
D=256K
256K
آزادسازی A
128K
C=64K
64K
256K
D=256K
256K
درخواست E=75K
E=128K
C=64K
64K
256K
D=256K
256K
آزادسازی C
E=128K
128K
256K
D=256K
256K
آزادسازی E
512K
D=256K
256K
آزادسازی D
1M