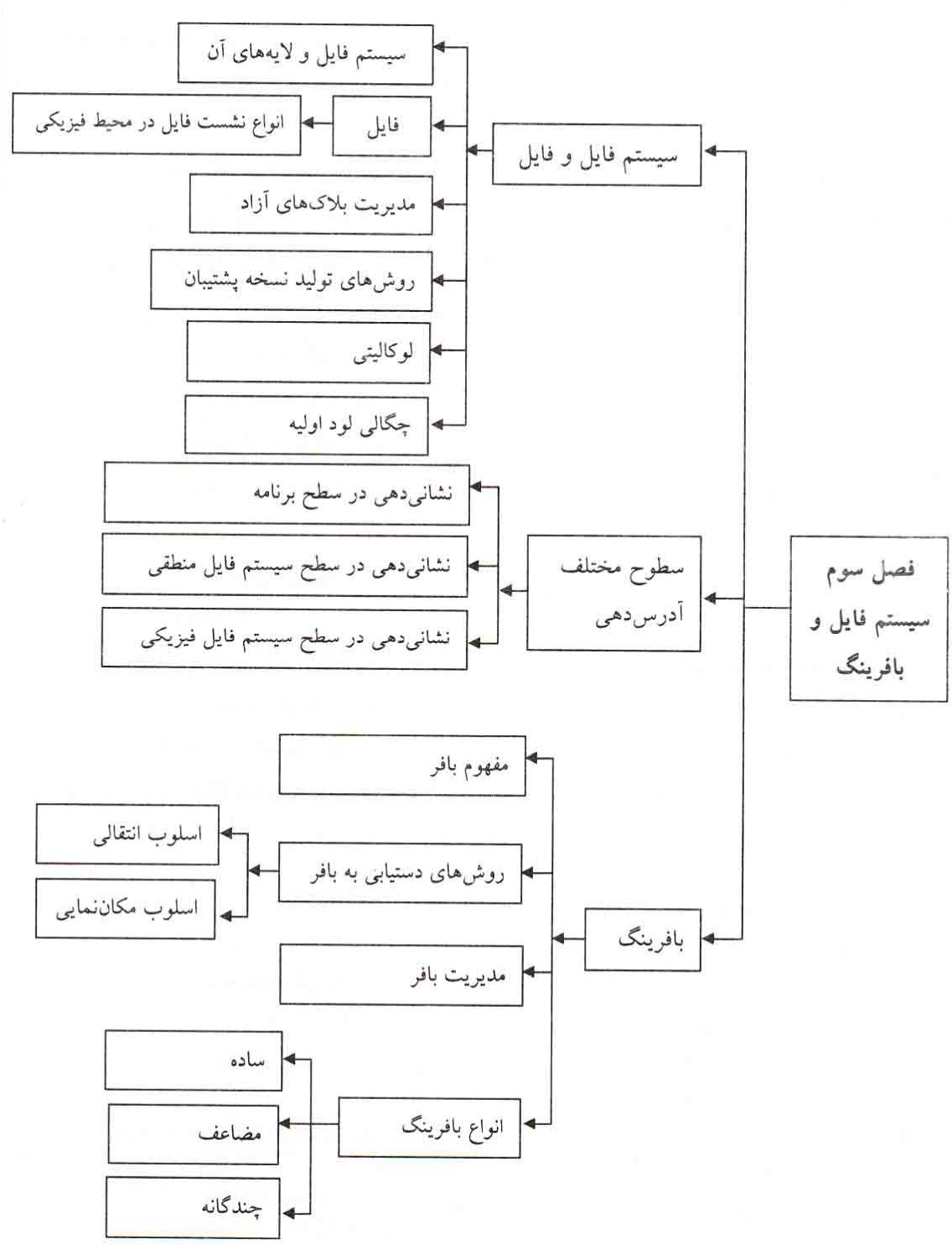


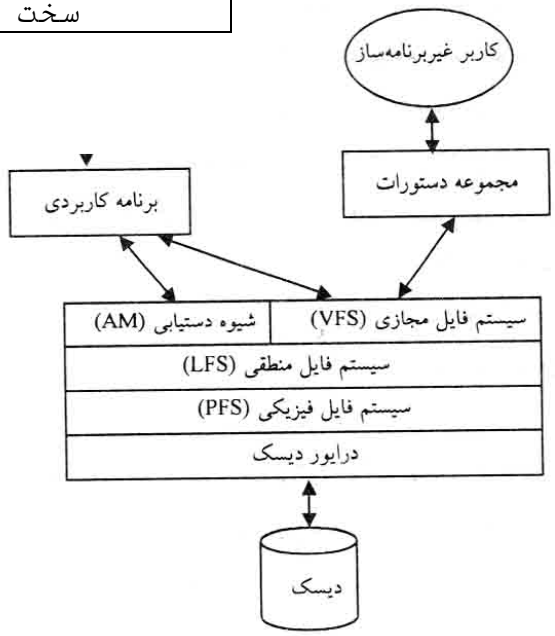
سيستم فايل

بخشي از سيستم عامل كه با فايل ها سر و كار دارد، مباحث مربوط به سيستم فايل به طور كامل در درس مفاهيم سيستم عامل مطرح مي گردد و ما در اينجا بخشي از آن مطالب را كه در درس ذخيره و بازيابي نياز داريم، يادآوري مي كنيم. در يك ديد كلي لايه هاي يك سيستم را مي توان مشابه شكل زير در نظر گرفت:

دستورات كاربر نهايي
برنامه هاي كاربردي
رويه هاي كتابخانه اي
سيستم فايل
سيستم عامل
مجموعه دستورات ماشين
سخت افزار

سيستم فايل درخواست هاي لايه بالاتر را تبديل به صدا زدن توابعي در سيستم عامل مي كند تا آن توابع عمليات فيزيكي I/O را انجام دهند. سيستم عامل نيز به نوبه خود يك برنامه كانال (Channel Program) را اجرا مي كند تا عمليات I/O فيزيكي در ديسك انجام پذيرد.

معماري كل سيستم فايل را مي توان مشابه شكل كلي ز



درايور: در قسمت پائين اين معماري قرار داشته و به طور مستقيم با كانال و كنترلگر ديسك در ارتباط است.

سيستم فايل فيزيكي يا PFS (Physical File System): اين لايه اغلب با محتويات بلاك ها و ساختار فايل ها سر وكار دارند، بلكه مسئول ذخيره كردن بلاك ها روي ديسك و منتقل ساختن آنها از ديسك به بافر و برعكس است. ممكن است اين لايه بخشي از خود سيستم عامل باشد.

سيستم فايل منطقي يا LFS (Logical File System): اين لايه اين امكان را مي دهد كه كاربران به ركوردها دستيابي داشته باشند. برنامه كاربر از طريق يك شيوه دستيابي يا AM (Access Method) اين لايه در ارتباط بوده و اين لايه درخواست هاي كاربر را انجام مي دهد.

شيوه دستيابي (AM): اين قسمت بالاترين لايه سيستم فايل بوده و واسط روابط بين برنامه كاربر و سيستم فايل منطقي است. اين لايه يك روش معين جهت دستيابي به ركوردها را در اختيار برنامه كاربردي قرار مي دهد.

سيستم فايل مجازي يا VFS (Virtual File System): كاربران غيربرنامه ساز اغلب از طريق اين لايه با سيستم فايل كار مي كنند. سيستم فايل مجازي ساختار خاصي براي فايل قائل نبوده و فايل را به صورت دنباله اي از بايت ها و كاراكترها مي بيند. كاربر غيربرنامه ساز به كمك مجموعه اي از دستورات خاص، نيازهاي خود را انجام مي دهد.

فايل:

همان گونه كه قبلاً گفتيم منظور اصلي ما در اين درس از فايل، مجموعه اي نامدار و اغلب داراي يك ساختار مشخص از نمونه هاي مختلف يك نوع ركورد (فايل تك نوعي) يا چند نوع ركورد (فايل چند نوعي) است. ولي فايل ممكن است تنها دنباله اي بي ساختار از كاراكترها باشد و مجموعه اي از ركوردها نباشد.

فايل داراي دو ساختار منطقي و فيزيكي است. ساختار منطقي، ساختاري است كه بر مبناي آن ركوردهاي منطقي كنار هم قرار مي گيرند. ساختار فيزيكي نحوه ذخيره سازي بلاك هاي فايل را روي رسانه خارجي نشان مي دهد به عبارتي ديگر ساختار فيزيكي نمايانگر ديد برنامه ساز سيستم از فايل است.

از ديد سيستم فايل منطقي، فايل مجموعه از ركوردهاي ذخيره شده است كه اغلب يك ساختار خاص داشته و از طريق يك شيوه دستيابي مورد استفاده قرار مي گيرد. ولي ازديد سيستم فايل فيزيكي، فايل از تعدادي بلاك تشكيل شده و ممكن است مجموعه اي از تقسيمات ديگر مثل باكت باشد كه بر مبناي طرح خاصي روي رسانه خارجي ذخيره شده است.

در اين درس، فايل ۳ ويژگي اصلي دارد: ۱- به اندازه اي بزرگ است كه به طور كامل در حافظه اصلي جا نمي گيرد. ۲- پايايي دارند يعني داده هاي آن از بين نمي روند مگر آنكه كاربر درخواست حذف آنها را بكند. ۳- مي تواند بين چند كاربر مجاز به صورت اشتراكي دستيابي شود.

از ديد كاربر مي توان گفت فايل يك مكانيسم انتزاعي است كه به كاربر اجازه مي دهد داده هاي خود را ذخيره، بازيابي، پردازش و احتمالاً داده هايي جديد را توليد كند. در حالت كلي انسان چهار عمل اصلي «توليد، ذخيره، بازيابي و پردازش» را با اطلاعات انجام مي دهد.

به طور كلي فايل ها از ديد كاربر دو دسته هستند: ۱- فايل عادي كه حاوي اطلاعات كاربر بوده و خود به خود به دو نوع اسكي يا دودويي مي باشد. ۲- فايل راهنما (Directory) يا فايل سيستمي كه حاوي اطلاعات خود سيستم فايل است.

تذكر ۱ : در كتاب آقاي روحاني مباحثي ديگر نظير نامگذاري فايل، صفات خاصه فايل، راهنماي فايل و انواع آن، تكنيك هاي پياده سازي راهنماي فايل، عمليات مربوط به راهنما و عمليات مربوط به فايل ها آمده است. از آنجا كه اين مباحث جزو درس سيستم عامل است، مي توانيد براي مطالعه آنها به فصل آخر كتاب مفاهيم سيستم عامل نوشته اينجانب رجوع كنيد.

تذكر ۲: در سيستم فايل، جدول راهنماي (File Directory) وجود دارد كه در آن اطلاعات مربوط به فايل ها ذخيره مي گردد. به طور كلي اطلاعات نمونه اي زير در اين جدول براي هر فايل ثبت مي گردد: نام فايل، آدرس اولين بلاك آن، طول ركورد (اگر ثابت باشد)، تعداد ركوردها، نام صاحب فايل، طول فايل، تاريخ ايجاد يا اصلاح فايل و غيره.

نشست فايل در محيط فيزيكي

شرح كامل اين موضوع را مي توانيد از فصل آخر كتاب مفاهيم سيستم عامل نوشته اينجانب مطالعه فرمائيد.

به طور فايل به دو صورت مي تواند در ديسك ذخيره شود: ۱- نشست پيوسته ۲- نشست ناپيوسته

نشست پيوسته: در اين روش فايل در بلاك هاي فيزيكي پيوسته و همجوار ذخيره مي شود اين تكنيك دو مزيت دارد. يكي آنكه پياده سازي آن ساده است و تنها با داشتن آدرس اولين بلاك و نيز تعداد بلاك ها مي توان به اطلاعات دسترسي داشت. ديگري آنكه كارايي و سرعت اين سيستم بالا مي باشد، چرا كه كل فايل را در يك عمل واحد مي توان خواند. ولي اين روش دو ايراد هم دارد. يكي آنكه اندازه فايل در زمان ايجاد آن بايد معلوم باشد. دوم آنكه اين روش مشكل بندبند شدگي خارجي را دارد يعني مثلاً ممكن است يك فضاي خالي ۱۰K و يك فضاي خالي ۳۰K داشته باشيم ولي نتوانيم يك فايل ۳۵K را ذخيره كنيم.

نشست ناپيوسته: در اين روش فايل مي تواند به صورت ناپيوسته در ديسك ذخيره شود و در اين حال بلاك ها و فايل براساس ترتيب منطقي آنها به يكديگر زنجير شوند. در اين روش با داشتن آدرس اولين بلاك، بقيه بلاك ها با پيمايش اين زنجير دستيابي مي شوند. در اين روش مشكل تكه تكه شدن خارجي وجود ندارد. با اينكه در اين تكنيك پي در پي خواندن فايل كار ساده اي ولي دستيابي تصادفي و مستقيم بسيار كند است.

تذكر ۱: براي اينكه دستيابي تصادفي در نشست ناپيوسته سريع تر گردد مي توان از ليست پيوندي مجهز به جدول راهنما استفاده كرد، مشابه جدول FAT در سيستم عامل DOS كه شرح كامل آن در كتاب مفاهيم سيستم عامل آورده شده است. ولي در اين روش اگر اين جدول راهنما در حافظه اصلي نگهداري شود ممكن است فضاي زيادي از RAM را اشغال كند.

مديريت بلاك هاي آزاد:

دو روش براي مديريت بلاك هاي آزاد وجود دارد: ۱- ليست پيوندي ۲- نگاشت بيتي يا بيت نقش (Bitmap) در روش ليست پيوندي ليستي از چند بلاك ديسك پديد مي آيد كه هر بلاك حاوي شماره بلاك هاي آزاد است. مثلاً

اگر اندازه هر بلاك ۱KB و هر شماره بلاك در ۳۲ بيت (۴ بيت) نمايش داده شود، آنگاه در هر بلاك مي توان شماره ۲۵۵ بلاك آزاد را ذخيره ساخت چرا كه:

$$\text{تعداد شماره هاي ذخيره شده در هر بلاك} = \frac{\text{اندازه هر بلاك}}{\text{اندازه هر شماره بلاك}} = \frac{2^{10}}{2^2} = 2^8 = 256$$

ولي توجه كنيد كه يكي از مدخل هاي برابر صفر بردار بيت متناظر حاوي شماره بلاك هاي آزاد استفاده شده و در نتيجه از ۲۵۶ مدخل، ۲۵۵ عدد آن قابل استفاده است.

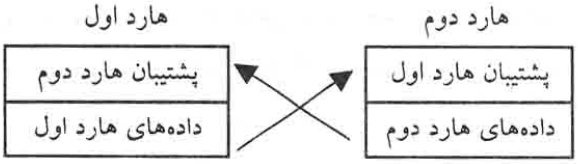
در روش بيت نقش براي ديسكي با n بلاك از n بيت استفاده مي شود. بيت متناظر با بلاك آزاد برابر ۱ و بيت متناظر با بلاك استفاده شده برابر صفر قرار داده مي شود. بنابراين مثلاً در اين روش ديسكي كه ظرفيت ۲۰۰ مگابايت داشته و هر بلاك آن ۱KB است به ۲۰۰ كيلوبيت حافظه احتياج دارد:

$$\text{تعداد بلاك هاي ديسك} = \text{تعداد بيت هاي مورد نياز} = \frac{\text{اندازه هر بلاك}}{\text{اندازه هر شماره بلاك}} = \frac{200 \times 2^{20}}{1 \times 2^{10}} = 200 \times 2^{20} = 200K$$

روش های تولید نسخه پشتیبان

در موارد زيادي براي پشتيبان گيري از اطلاعات ديسك از نوار، ديسكت، CD و غيره استفاده مي شود. چند روش ديگر براي اين كار عبارتند از:

۱- استفاده از نيمه دو ديسك. در اين روش از دو هارددیسك استفاده شده كه فضاي هر كدام به دو قسمت تقسيم مي شود. اغلب در انتهاي هر روز كاري نيمه داده اي هر ديسك روي نيمه پشتيبان ديسك ديگر كپي مي شود:



۲- توليد دامپ هاي تدريجي (Incremental dump). در اين تكنيك علاوه بر ايجاد نسخه پشتبيان به صورت هفتگي يا ماهيانه، دامپ فايل هايي كه از زمان ايجاد آخرين نسخه پشتبيان تغيير كرده اند، به صورت روزانه ايجاد مي شود. به علت مصرف زياد حافظه در اين تكنيك اغلب از نوار استفاده مي گردد.

۳- آينه سازسي (Mirroring): در اين روش از دو يا چند ديسك استفاده مي گرد. مثلاً در حالتي كه دو ديسك داريم عمل نوشتن در دو ديسك صورت گرفته كه به آن آينه سازي مي گوئيم البته عمل نوشتن روي ديسك دوم قدري با تأخير انجام مي گيرد. در اين روش عمل خواندن فقط از روي يك ديسك انجام مي شود و اگر ديسكي خراب شود، داده ها را مي توان از ديسك ديگر خواند.

لوكالیتي (Locality)

ركورد منطقي بعدي ركوردي است كه از ديد برنامه بايد پس از ركورد فعلي مورد پردازش قرار گيرد، به عبارتي ديگر ركورد منطقاً همجوار ركورد فعلي مي باشد. همچواري منطقي ركوردها براساس نظمي است كه برنامه تعريف مي كند (مانند نظم نزولي معدل دانشجويان).

ميزان همسايگي ركورد منطقي بعدي، نسبت به ركورد فعلي روي حافظه جانبي را لوكاليتي مي گويند. همانطور كه قبلاً گفتيم، اگر همجواري منطقي به همان صورت ديسك پياده سازي نشده باشد، در پردازش سريال فايل، هد ديسك به طور مرتب جلو و عقب شده و بدين دليل زمان اين پردازش زياد مي گردد. درجات لوكاليتي از قوي به ضعيف به ترتيب عبارتند از:

۱- ركورد بعدي در همان بلاكي است كه ركورد جاري هست و اين بلاك در بافر قرار دارد. در اين وضعيت عمل I/O نداريم.

۲- ركورد بعدي در بلاك بلافاصله بعدي ركورد جاري از همان استوانه است. دراين وضعيت عمل I/O داريم ولي s=0 و r=0 مي باشد. از حالت ۲ به بعد همواره عمليات I/O را داريم.

۳- ركورد بعدي در همان استوانه است كه ركورد فعلي قرار دارد. در اينحالت s=0 و r>0 مي باشد.

۴- ركورد بعدي روي استوانه اي با همان شماره ولي در ديسكي ديگر است يعني فايل روي چند ديسك توزيع شده است. در اين حالت نيز I/O>0 ، s=0 و r>0 است.

۵- ركورد بعدي در استوانه همجوار مي باشد. در اين حالت $s=s_{min}>0$ و r>0 مي باشد.

۶- ركورد بعدي روي استوانه ديگري است كه آدرس آن از پردازش ركورد جاري به دست مي آيد.

۷- ركورد بعدي روي استوانه اي ناشناخته است كه آدرس آن با انجام محاسباتي خاص به دست مي آيد.

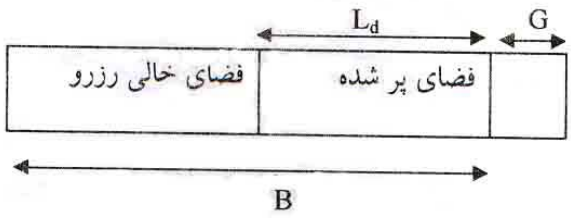
۸- ركورد بعدي روي استوانه اي ناشناخته است كه آدرس آن با رجوع به فايلي ديگر به دست مي آيد.

۹- ركورد بعدي روي رسانه است كه در زمان حاضر روي درايور وجود ندارد.

تذكر: ممكن است لوكاليتي ركوردهاي فايل به علت عمليات درج و حذف و بهنگام سازي به تدريج از قوي به ضعيف ميل كند.

چگالي لود اوليه

اگر بتوانيم ميزان رشد فايل را در آينده به درستي پيش بيني كنيم، مي توان بخشي از فضاي بلاك را در لود اوليه (هنگامي كه براي اولين بار فايل پديد مي آيد) به صورت رزرو شده خالي گذاشت. در اين حالت نسبت فضاي استفاده شده به فضاي كل بلوك را «چگالي لود اوليه» مي گويند و اين نسبت كمتر از يك خواهد بود.



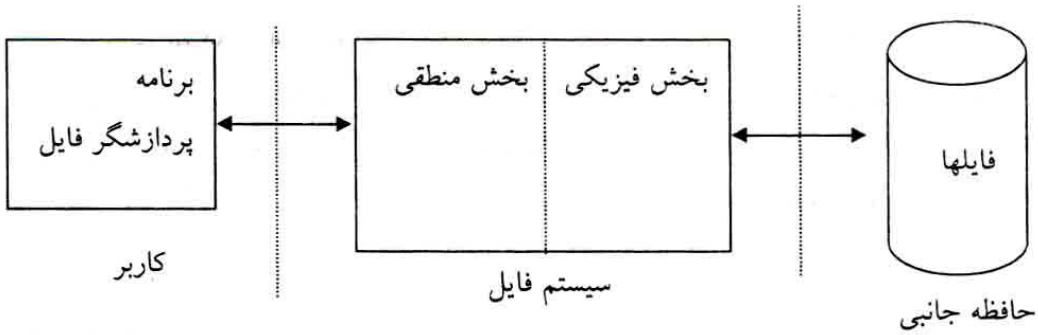
$$\text{چگالی لود اولیه} = \frac{L_d}{B} < 1$$

مزاياي اين تكنيك عبارتند از: ۱- وجود ناحيه رزرو باعث مي گردد لوكاليتي ركوردهاي فايل بهتر حفظ گردد. ۲- اين تكنيك برخي عمليات روي فايل را ساده تر و سريع تر مي سازد، مثلاً براي درج يك ركورد لازم نيست كل فايل را به سمت پائين شيفت دهيم واين كار با شيفت درون بلاكي امكان پذير مي گردد.

معايب اين تكنيك عبارتند از: ۱- ناحيه رزرو شده در واقع نوعي حافظه هرز است كه طول خطي فايل را افزايش داده و در نتيجه خواندن كل فايل زمان بر مي شود. ۲- اگر توزيع درج ركوردها يكنواخت نباشد، آنگاه بعضي از بلوك ها پرشده در حالي كه همچنان بعضي از بلوك ها سبكبار هستند و در انتهاي آنها حافظه هرز باقي مي ماند.

سطوح مختلف آدرس دهي

در يك ديد كلي مي توان سيستم فايل را مشابه شكل زير به دو بخش فيزيكي و منطقي تقسيم بندي كرد:



بخش منطقي سيستم فايل دستورات كاربر نظير باز و بسته كردن و خواندن و نوشتن فايلها را انجام مي دهد. قسمت فيزيكي سيستم فايل به طور مستقيم به فايلهاي موجود بر روي حافظه جانبي دستيابي دارد. اين قسمت فيزيكي دستورات وارد شده از قسمت منطقي سيستم فايل را به دستوراتي جهت صدور به رسانه ذخيره سازي تبديل مي كند. سه عمل اصلي اي بخش عبارتند از: مكان يابي، خواندن و نوشتن بر روي حافظه فيزيكي.

با توجه به شكل و توضيحات فوق سطوح مختلف آدرس دهي عبارتند از:

۱- آدرس دهي در سطح برنامه

در اين سطح آدرس دهي مي تواند به صورتهاي زير باشد:

الف) محتوايي يا مقداري: كاربر مقدار يك يا چند صفت خاصه را جهت جستجو مي دهد. اين صفت خاصه مي تواند كليد باشد يا نباشد.

ب) آدرس دهي نسبي: در اينحالت كاربر فايل را به صورت يك ساختار خطي مي بيندكه هر ركورد يك شماره يكتا دارد و اغلب اين شماره با يك عدد يك آغاز مي گردد. در اين شيوه كاربر از آدرس نسبي ركورد استفاده مي كند.

ج) آدرس دهي نمادين (Symbolic): در اينحالت ركورد دلخواه توسط يك نام، مشخص و آدرس دهي (نشان دهي) مي شود.

۲- آدرس دهي در سطح منطقي سيستم فايل

بخش منطقي سيستم فايل كل فضاي رسانه و فايلها را به صورت يك آرايه از بلاكها مي بيند. هر بلاك يك شماره يكتا دارد كه به آن آدرس نسبي بلاك يا RBA (Relative Block Address) گفته مي شود. شماره RBA از صفر شروع مي شود. قسمت منطقي سيستم فايل آدرس توليد شده در سطح برنامه را دريافت كرده و آن را به آدرس RBA تبديل مي كند و سپس اين RBA را به سمت بخش فيزيكي مي فرستد.

۳- آدرس دهي در سطح فيزيكي سيستم فايل

بخش فيزيكي سيستم فايل آدرس RBA را از بخش منطقي دريافت كرده و آن را به آدرس دهي خاص رسانه ذخيره سازي تبديل مي كند. مثلاً در مورد ديسك، آدرس RBA را به شماره سيلندر، شماره شيار در سيلندر و شماره بلوك در شيار، تبديل مي كند.

نشان دهي در سطح سيستم فايل منطقي

با مثال زير نحوه تبديل آدرس ها را در سيستم فايل به صورت نمونه اي نشان مي دهيم.

مثال ۱: كاربري در سطح برنامه خود دستور فرضي زير را وارد مي كند:

Read from Record=6

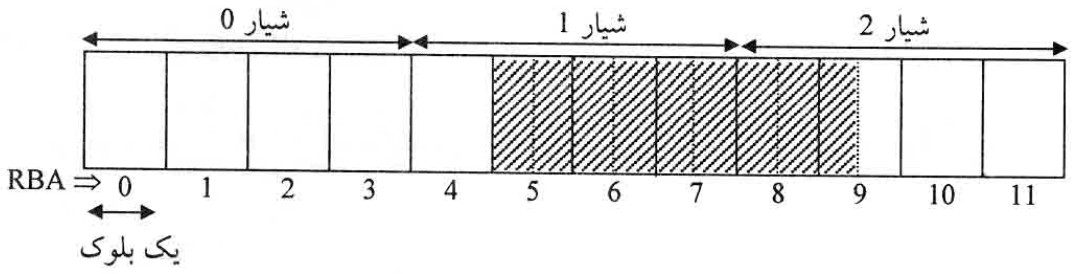
يعني ركورد ششم از فايل student خوانده شود. تعداد ركوردهاي منطقي فايل ۹ عدد بوده و طول هر ركورد منطقي ۱۰۰ بايت و طول هر بلاك ۲۰۰ بايت است. فايل به صورت پيوسته بر روي ديسك ذخيره شده و هر شيار حاوي ۴ بلاك است. آدرس RBA شروع فايل بر روي ديسك برابر ۵ مي باشد. مطلوب است: (شماره RBA از صفر آغاز مي شود).

الف) ركورد ششم فايل از كدام بايت فايل آغاز مي شود؟

ب) RBA حاوي ركورد ششم نسبت به ابتداي فايل چه مي شود؟

ج) RBA حاوي ركورد ششم نسبت به اولين بلاك ديسك چه مي شود؟

حل: شكل زير شيارهاي ديسك و نحوه ذخيره شدن فايل بر روي ديسك را نشان مي دهد:



هر بلوك حاوي دو ركورد بوده و قسمت هاشور خورده فايل ذخيره شده مي باشد.

الف) ركورد ششم فايل از بايت شماره ۵۰۰ فايل آغاز مي شود: $(6\_1)\times 100 = 5\times 100 = 500$

ب) با توجه به شكل مشخص است كه RBA ركورد ششم نسبت به ابتداي فايل برابر ۲ است. به عبارتي ديگر ركورد ششم در سومين بلاك فايل قرار دارد.

ج) با توجه به شكل ركورد ششم فايل در RBA= ۷ نسبت به اول ديسك قرار دارد. از طريق محاسبه داريم:

RBA (نسبت به اول ديسك) =RBA (اول فايل) + RBA (ركورد مورد نظر در فايل)

=۵+۲=۷

ركورد ششم، آدرس دهي در سطح برنامه است. بخش منطقي سيستم فايل اين آدرس را گرفته و به RBA=۷ تبديل مي كند. بخش فيزيكي سيستم فايل اين عدد ۷ را گرفته و در مي يابد كه كدام قسمت فيزيكي ديسك را بايد بخواند (بلوك چهارم از شيار شماره ۱). سيستم فايل بلوك مورد نظر را خوانده و در بافر قرار مي دهد. در آخر سيستم فايل داده ها را از بلاك بندي درآورده و ركورد دوم اين بلاك را به ناحيه كاري كاربر مي فرستد.

افست ركورد iام نسبت به اول فايل برحسب بايت $= (i-1)\times R$

R طول هر ركورد برحسب بايت مي باشد.

افست بلاك حاوي ركورد iام نسبت به اول فايل $=rba_{REC}=\left[\frac{(i-1)\times R}{B}\right]$

افست بلوك حاوي ركورد نسبت به اول فايل + شماره بلوك آغاز فايل = شماره بلوك حاوي ركورد نسبت به اول ديسك

$$RBA_{REC}=RBA_{BOF}+rba_{REC}$$

نشان دهي در سطح سيستم فايل فيزيكي

همانطور كه قبلاً گفتيم عموماً بلوكها به صورت سيلندر بر روي ديسك ذخيره مي شوند. اگر تعداد رويه هاي ديسك يا به عبارتي ديگر تعداد شيارهاي موجود در هر سيلندر را با t و تعداد بلاكهاي موجود در هر شيار را به b نمايش دهيم و آدرس فيزيكي يك بلوك به صورت ۳ عدد (cyl#, trk#, blk#) باشد آنگاه شماره سيلندر (cyl#)، شماره شيار نسبت به اول آن سيلندر (trk#) و شماره بلاك نسبت به اول شيار (blk#) از فرمولهاي زير بدست مي آيند. توجه كنيد شماره ها همواره از صفر شروع مي شوند:

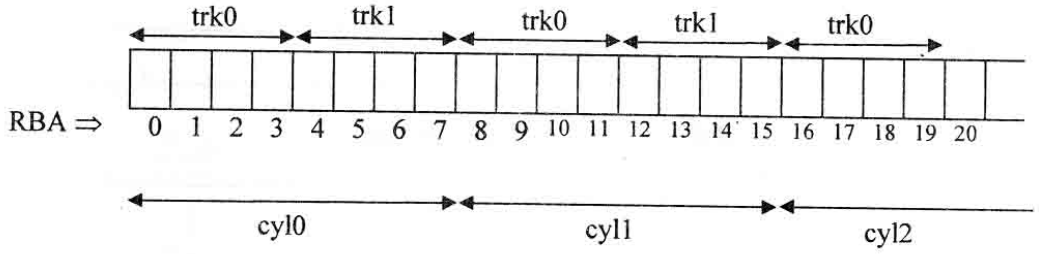
$$cyl\# = [\frac{RBA}{(t \times b)}]$$

$t \times b$ تعداد بلاك در هر استوانه و RBA شماره مطلق بلوك مورد نظر نسب به ابتداي ديسك است، يعني همان RBA_{REC} كه از قسمت سيستم فايل منطقي به دست مي آيد.

$$trk\# = [\frac{RBA \bmod (t \times b)}{b}]$$

Blk#=RBA MOD b

مثال ۲: ديسكي داريم كه هر شيار آن شامل ۴ بلاك است (b=4) و هر سيلندر آن ۲ شيار دارد يا به عبارتي ديگر اين ديسك ۲ عدد هد يا رويه دارد (t=2). آدرس فيزيكي بلوكي با RBA=17 به فرم (cyl#, trk#, blk#) چه مي شود؟



از روي شكل مشخص است كه: cyl#=2, trk#=0, blk#=1

$$cyl\# = [\frac{RBA}{t \times b}] = [\frac{17}{2 \times 4}] = 2$$

blk#=RBA MOD b=۱۷MOD۴=۱

$$trk\# = [\frac{RBA \bmod (t \times b)}{b}] = [\frac{17 \bmod (2 \times 4)}{4}] = 0$$

تذكر ۱: نماد t كه تعداد شيار در سيلندر است در واقع معادل تعداد هدها يا تعداد رويه هاي ديسك نيز مي باشد. همچنين تعداد بلوك در هر شيار كه در اين بخش نماد b نشان داده شده است در اغلب كتاب ها با نماد T_f بيان شده است. از آنجا كه در عموم كتابها نماد t براي نرخ انتقال و نماد b براي عداد بلاك ها به كار رفته است، بهتر بود كه از نماد H (به معناي تعداد هد) به جاي t و از نماد T_f به جاي b استفاده مي شد ولي ما عين نمادكتاب آقاي روحاني را آورديم. البته خود آقاي روحاني فقط دراين قسمت از اين نمادها استفاده كرده و در بقيه كتاب خود از همان نماد T_f براي تعداد بلاك در شيار استفاده كرده است.

تذكر ۲: در فرمول هاي فوق فرض كرده ايم كه فقط يك هارديسك داريم. اگر چندين هارديسك داشته باشيم آنگاه در فرمول هاي فوق به جاي RBA بايد مقدار زير را قرار دهيم:

$RBA=RBA_{REC}-RBA_{BOD}$

منظور از RBA_{BOD} ، شماره RBA اول آن هارديسك (Begin Of Device) نسبت به كل درايو است. توجه كنيد در حالتي كه مثلاً دو هارديسك داريم، سيستم فايل منطقي فقط يك درايو مي بيند، مثلاً در حالت:

شماره ديسك	تعداد استوانه	تعداد شيار در استوانه	تعداد بلاك در شيار
$D_۱$	$C_۱$	$t_۱$	$b_۱$
$D_۲$	$C_۲$	$t_۲$	$b_۲$

آنگاه: $0 \le RBA_{D_1} \le S_1 \_ 1$, ظرفيت ديسك $D_۱ = C_1 \times t_1 \times b_1$

$S_1 \le RBA_{D2} \le S_1 + S_2 \_ 1$, ظرفيت ديسك $D_۲ = C_2 \times t_2 \times b_2$

و محدوده مقادير RBA درکل فضاي ذخيره سازي برابر خواهد بود با $0 \le RBA \le S_1 + S_2 \_ 1$

بافر

بافر ناحيه اي در حافظه اصلي است كه جهت ايجاد هماهنگي بين سرعت وسيله I/O و پردازنده براي بالا بردن سرعت كلي سيستم استفاده مي شود. اگر فايل بلاك بندي نشده باشد در هر بار عمليات I/O يك ركورد و در حالت بلاك بندي حداقل يك بلاك در بافر قرار داده (يا خوانده) مي شود. در اينجا فرض مي كنيم هر باكت يك بلوك باشد. بافر اغلب از منطقه اي از حافظه اصلي به برنامه داده مي شود كه به آن منطقه بافرها يا Buffers Pool گويند. كلمة Pool به معناي حوض، ذخيره مشترك و يك كاسه كردن است. منطقه بافرها در حافظه ناحيه اي پيوسته را تشكيل مي دهند.

سيستم هاي I/O همواره حداقل دو بافر دارند (يكي جهت ورودي و ديگري براي خروجي)، در غير اينصورت كارايي سيستم I/O كاهش مي يابد. براي درك علت اين موضوع عمليات زير را در نظر بگيريد:

برنامه اي براي ورود و خروجي كاراكترها فقط يك بافر I/O دارد. هنگاميكه برنامه اولين كاراكتر را درخواست مي كند، سكتورحاوي آن كاراكتر در بافر قرار داده مي شود وكاراكتر به برنامه منتقل مي گردد. بعد از آنبرنامه نياز به فرستادن كاراكتري به خروجي دارد. در اينحال بافر با سكتور حاوي كاراكتر خروجي، پرشده و محتويات قبلي بافر از بين مي رود. سپس براي خواندن كاراكتر بعدي دوباره محتويات بافر موجود با سكتور جديد جايگزين مي گردد.

برنامه كاربر به دو روش مي تواند به محتويات بافر دستيابي داشته باشد:

۱- **روش انتقالي يا حالت حركت (move mode)** در اينحالت ناحيه بافر جداي از حافظه مربوط به برنامه (ناحيه كاري) بوده و برنامه مستقيماً به بافر سيستم دسترسي ندارد و بافر ويژه خود را خواهد داشت. در اين روش بلاك از بافر ورودي به ناحيه كاري برنامه و يا از ناحيه كاري به بافر خروجي فرستاده مي شود. عمل بلاك بندي ون بلاك گشايي توسط سيستم انجام مي پذيرد. پس در اين شيوه داده ها قبل از دستيابي از محلي در حافظه به محل ديگر انتقال مي يابند و زمان لازم جهت اين كپي كردن مي تواند نسبتاً زياد باشد.

۲- **روش مكان نمايي يا تعيين محل (Locate mode)** به دو روش مي توان از حركت موجود در تكنيك انتقالي اجتناب كرد. يكي آنكه داده هاي I/O مستقيماً بين وسيله I/O و ناحيه كاري برنامه رد و بدل گردد. راه ديگر آنكه مديريت فايل از بافرهاي سيستم براي همه I/Oها استفاده كند ولي به كمك اشاره گرها محل بافرهاي سيستم را در اختيار برنامه قرار دهد. هر دو تكنيك نمونه هايي از روش بافردهي Locate mode مي باشند. بدين ترتيب مستقيماً بر روي داده هاي بافر I/O عمليات لازم را انجام داده و ديگر نيازي به انتقال داده ها بين بافر I/O و بافر برنامه (ناحيه كاري برنامه) وجود ندارد. در اين روش عمل بلاك بندي و بلاك گشايي را خود برنامه انجام مي دهد.

از آنجا كه دو روش فوق (move mode و Locate mode) در هر دو عمل ورودي و يا خروجي قابل استفاده اند، چهار شيوه دستيابي به بافر وجود دارد:

۱ـ روش انتقالي در ورودي و خروجي
۲ـ روش مكان نمايي در ورودي و خروجي
۳ـ روش انتقالي در ورودي و مكان نمايي در خروجي
۴ـ روش مكان نمايي در ورودي و انتقالي در خروجي

تذكر ۱: از يك نظر مي توان بافرها را دو دسته كلي سخت افزاري و نرم افزاري تقسيم كرد. بافر سخت افزاري در كنترلگر دستگاههاي جانبي مثل ديسك و پرينتر قرار دارد و بافر نرم افزاري ناحيه اي در حافظه اصلي است كه توسط سيستم عامل در اختيار برنامه ها قرار داده مي شود.

تذكر ۲: بافرها را با سه روش مي توان ساخت: ۱ـ با ايجاد ناحيه از حافظه در برنامه يعني برنامه ساز خودش بافر را ايجاد كند. ۲ـ با اجراي يك ماكرو از سيستم عامل درخوائست شود كه بافر را ايجاد كند. ۳ـ خود سيستم عامل به صورت خودكار هنگامي كه فايلي را باز مي كند، بافرها را پديد آورده و بعد از بستن فايل، بافرها را بازپس بگيرد.

مديريت بافر

آرايش واقعي بافرهاي سيستم عامل انجام مي پذيرد و به ندرت توسط برنامه نويسان برنامه هاي سطح بالا كنترل مي شود. ولي برنامه نويسان مي توانند تعداد بافرهايي كه از سيستم به برنامه اي نسبت داده مي شود را كنترل كنند.

مديريت بافر بايد به گونه اي باشد كه كارايي استفاده از حافظه هايي جانبي زياد شود. از طرف ديگر افزايش اندازه بافرها باعث مصرف حافظه اصلي مي گردد. تعيين نقطه اي كه در آن، افزايش بافرهاي اضافي، كمكي به

افزايش كارايي نمي كند ساده نيست. با آنكه با كاهش قيمت حافظه اصلي، هزينه استفاده از بافرهاي بزرگتر كاهش يافته ولي هر چه تعداد بافرها بيشتر باشد، سيستم عامل زمان بيشتري را مي بايست براي مديريت آنها صرف كند.

تذكر ۱: در سيستم هايي كه از حافظه صفحه بندي شده (paging) استفاده مي كنند، بخشي از سيستم عامل كه صفحات را مديريت مي كند، مي تواند مديريت بافر را نيز انجام دهد.

تذكر ۲: اگر فقط يك ركورد از بلاك خوانده شده را نياز داشته باشيم، مديريت بافر مي تواند بافر را بعد از دادن ركورد به برنامه پاك كند. اما اغلب حفظ محتويات آن بلاك در بافر كارايي را افزايش مي دهد. چرا كه احتمالاً برنامه به ركوردهاي ديگر آن بلوك در آينده نزديك نياز خواهد داشت، به ويژه در پردازش فايلها به صورت ترتيبي كه در آنها ركورد منقطي بعدي، همان ركورد فيزيكي بعدي است.

انواع بافردهي (بافرينگ)

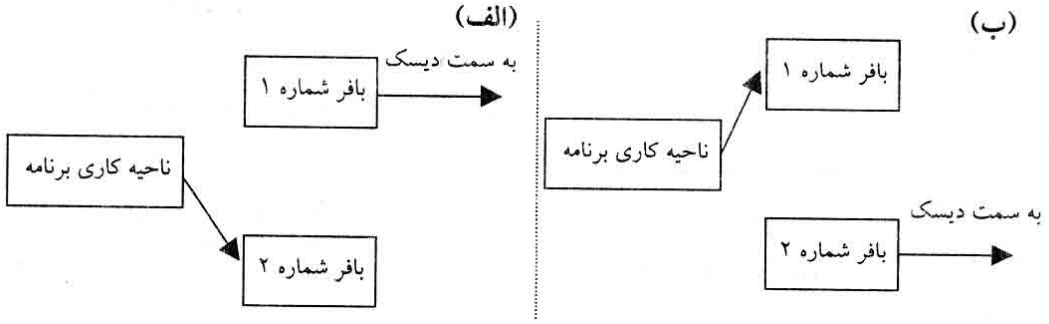
از نظر تعداد بافرهايي كه به برنامه اختصاص داده مي شود انواع بافردهي (buffering) عبارتند از:

۱ـ بافردهي ساده يا يگانه (single buffering)

در اين روش فقط از يك بافر I/O استفاده شده و بدين دليل كارايي برنامه كاهش مي يابد. در هنگامي كه بافر در حال پرشدن است CPU نمي تواند از محتويات بافر استفاده كند و بايد منتظر و بيكار باقي بماند. البته در سيستم هاي چندبرنامه اي از اين زمانهاي انتظار (Idle) مي توان براي اجراي برنامه هاي ديگر استفاده كرد.

۲ـ بافردهي دوگانه يا مضاعف (Double buffering)

اگر از دو بافر استفاده شود و همپوشاني I/O و CPU مجاز باشد، پردازنده مي تواند در هنگام ارسال يك بافر به ديسك بافر ديگري را پركند. به عبارت ديگر با دوبافر مي توان در حين انتقال يك بلاك به بافر، بافر ديگري را كه پر است پردازش كرد، مطابق شكل زير:



در شكل (الف) در حين انتقال محتويات بافر ۱ به ديسك، بافر شماره ۲ پر مي شود. در شكل ب در حين انتقال بافر شماره ۲ به ديسك بافر شماره ۱ پر مي گردد.

در شكل فوق زماني را كه CPU جهت پردازش يك بافر صرف مي كند بايد كمتر از زمان انتقال بلاك به بافر ديگر باشد. به عبارتي ديگر براي اينكه كارايي بافردهي دوگانه خوب باشد بايستي:

$$C_B \le b_{tt} \quad \text{يا} \quad C_B \le \frac{B+G}{t} \quad \text{يا} \quad C_R \le \frac{R+W_R}{t}$$

در اين فرمولها: B طول بلاك برحسب بايت، G طول گپ بر حسب بايت، t نرخ انتقال برحسب بايت در ثانيه، b_{tt} زمان انتقال يك بلاك، C_B زمان پردازش يك بلاك، C_R زمان پردازش يك ركورد و W_R حافظه هرز به ازاي هر ركورد مي باشد.

تذكر ۱: توجه كنيد حرف t در اين درس به معناي زمان (مشابه آنچه كه در فيزيك خوانده ايد) نمي باشد، بلكه جهت نرخ انتقال (كه شبيه سرعت در فيزيك است) استفاده مي شود.

تذكر ۲: با توجه به تعريف b_{tt} داريم: $b_{tt} = \frac{B}{t}$

تذكر ۳: اگر $C_B > b_{tt}$ باشد آنگاه بافردهي دوگانه كارايي مناسبي نداشته ونرخ انتقال كم مي شود.

تذكر ۴: در پردازش فايلها به صورت انبوه و پي در پي مي بايست حتماً دو بافر داشته باشيم، در غير اينصورت كارايي برنامه كاهش مي يابد.

۳ـ بافردهي چند گانه (Multiple buffering)

با استفاده از بافردهي چندگانه، هنگام پردازش پي در پي و انبوه فايل، مي توان بلوك هاي فايل را جلوتر (از پيش) خواند و در بافر ذخيره كرد. بدين ترتيب همواره ركورد بعدي در بافر موجود خواهد بود. به اين روش صف بندي هم مي گويند چرا كه سيستم با پيشدستي در عمليات I/O بافرها را از قبل براي CPU آماده مي سازد. براي كار از چندين بافر استفاده مي شود.

اين بافر دهي اغلب با ساختار صف چرخشي (كه در درس ساختمان داده ها مطرح مي شود) پياده سازي مي گردد و از اين نظر به آن بافر چرخشي هم مي گويند.

تذكر ۱: اگر سرعت پردازش خيلي بيشتر از سرعت عمليات بافرينگ باشد، حتي با بافردهي چندگانه نيز ممكن است وضعيتي پيش بيايد كه پردازنده مجبور به انتظار كشيدن باشد. البته در سيستم هاي چند برنامه اي در اين حالت CPU به فرايندهاي ديگر سوئيچ مي كند. بنابراين در كل وجود بافرينگ كارايي سيستم را افزايش مي دهد.

تذكر ۲: نحوه خواندن يا نوشتن بلوكها به شيوه بافردهي نيز بستگي دارد. مثلاً اگر سيستم از بافر استفاده نكند نمي توان عمليات بلاك بندي را انجام داد و در اين حالت ركورد به ركورد بايد داده ها به ناحيه كاري برنامه منتقل شوند (روش مبنايي يا Basic). اگر برنامه ناحيه كاري نداشته باشد يعني از روش مكان نمايي (Locate mode) استفاده شود، ديگر عمليات I/O پيشرس امكان پذير نخواهد بود.

در روش انتقالي (move mode)، سيستم بايد بلوكها را به ناحيه كاري برنامه انتقال دهد كه بدين ترتيب جهت اينكار زماني به هدر مي رود. جهت جلوگيري از اين اتلاف زماني مي توان از روش مكان نمايي (Locate mode) استفاده كرد. روش مكان نمايي به همراه بافردهي دوگانه در حاليكه $C_B \leq b_{tt}$ باشد بسيار كارآمد خواهد بود.

كارايي سيستم فايل

كارايي سيستم فايل به عوامل متعددي بستگي دارد از جمله:

- اندازه بلاك بندي و نحوه بلاك بندي
- لوكاليتي ركوردها
- پارامترهاي رسانه (كه در فصل بعدي شرح مي دهيم)
- نحوه آرايش سلسله مراتبي حافظه
- نحوه بافردهي و مديريت آن
- مديريت فايلها در سيستم عامل
- روشهاي كاهش زمان پيگرد و درنگ دوراني (كه در فصل بعدي شرح مي دهيم)
- استفاده از ساختار فايلها با كارآمدي بيشتر (كه از فصل ۵ به بعد شرح مي دهيم)