جزوه درس

ذخیره و بازیابی اطلاعات

مدرس :

حمید عرب‌نژاد

مهر ۸۷

www.nayabprojects.com

فصل اول :

وسایل ذخیره و بازیابی اطلاعات

www.nayabprojects.com

تعریف حافظه :

هر دستگاهی که قادر به نگهداری اطلاعات باشد به نحوی که استفاده از آن بتواند به اطلاعات مورد نیازش در هر لحظه که لازم باشد دستیابی داشته باشد . حافظه ها به دو دسته تقسیم می گردند .

۱) حافظه های درون ماشینی که پردازنده مستقیماً با آن در ارتباط است
۲) حافظه های برون ماشینی

خصوصیات و ویژگیهای انواع حافظه :

۱) **نوشتن و خواندن :** هر حافظه باید این قابلیت را داشته باشد که در آن بتواند اطلاعات را نوشت (درج کرد) و یا بتوان از آن اطلاعات را خواند (بازیابی کرد) مانند : RAM ها و هارد دیسک .
۲) **قابلیت دستیابی پذیری :** هر حافظه را می توان با استفاده از مکانیزم آدرس دهی مورد دستیابی قرار داد . دستیابی ممکن است به منظور خواندن یا نوشتن در حافظه صورت گیرد. مدت زمانی را بین لحظه ی دستور خواندن یا نوشتن در حافظه و زمانی که داده دستیابی می شود را زمان دستیابی یا (Access time) گفته می شود .
۳) **آدرس پذیر بودن :** هر حافظه مجهز به یک مکانیزم آدرس است به تعبیر دیگری می توان به کمک این قابلیت به مکانی از حافظه دسترسی پیدا کرد .
۴) **نرخ انتقال یا سرعت انتقال :** به میزان اطلاعاتی که در واحد زمان در حافظه قابل انتقال است نرخ انتقال گفته می شود و واحد آن بایت بر ثانیه می باشد .
۵) **ظرفیت :** هر حافظه دارای ظرفیتی است که با بیت یا بایت یا با آدرس پارامترهای وابسته بیان می شود .
۶) **مانا یا نامانا بودن اطلاعات :** برخی از حافظه ها اطلاعات را به صورت دائمی نگهداری می کنند (مانا) و برخی حافظه ها اطلاعات را به صورت موقت در خود نگهداری می کنند (نامانا)

دلایل به کارگیری انواع مختلف رسانه های ذخیره سازی :

۱) حافظه های درون ماشینی دارای حافظه های محدود هستند .
۲) لزومی ندارد که همه ی اطلاعات مورد نیاز برای یک محیط عملیاتی ، همیشه در حافظه ی درون ماشینی باشد بلکه باید آن دسته از اطلاعات که مورد نیاز برای اجرای برنامه می باشد ، در حافظه ی درون ماشینی وجود داشته با شد .
۳) رسانه های ذخیره سازی سریع (حافظه های درون ماشینی) غالباً گران هستند .
۴) حجم اطلاعات ذخیره سازی روز به روز به صورت تصاعدی در حال افزایش است و ما نمی توانیم این حجم عظیم اطلاعات را در حافظه نگهداری کنیم .
۵) حافظه های درون ماشینی معمولاً نامانا هستند و اطلاعات را نمی توانند به صورت دائم در خود نگه دارند .

مقایسه ی حافظه های درون ماشینی با حافظه های برون ماشینی :

۱) ظرفیت حافظه های برون ماشینی نسبت به درون ماشینی بسیار بالاتر می باشد .
۲) هزینه ی یک بایت اطلاعات در حافظه های درون ماشینی بسیار بیشتر از حافظه های برون ماشینی می باشد .
۳) سرعت دستیابی به اطلاعات در حافظه های درون ماشینی بسیار بیشتر از حافظه های برون ماشینی می باشد .

رده بندی حافظه ها (سلسله مراتب حافظه ها) بر اساس افزایش ظرفیت و کاهش سرعت و هزینه :

۱) ثبات
۲) حافظه نهان (cash)
۳) RAM
۴) دیسک سخت
۵) دیسک نوری
۶) دیسک مغناطیسی

انواع حافظه های برون ماشینی :

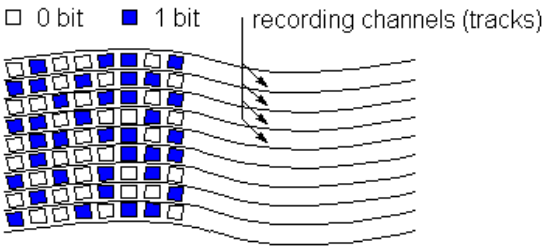
۱) کارت و نوار منگنه شدنی
۲) نوار مغناطیسی
۳) دیسک مغناطیسی
۴) طبله یا درام (Drum)
۵) دیسک نوری
۶) دیسک نوری – مغناطیسی

که در ادامه به بررسی برخی از این موارد خواهیم پرداخت .

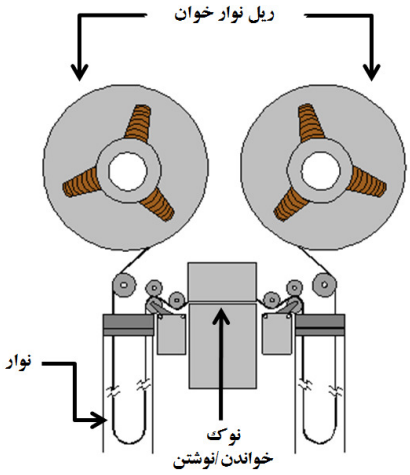
نوار مغناطیسی :

رسانایی است از جنس پلاستیک با غشائ مغناطیسی اطلاعات بر روی شیارهایی که در سطح نوار قرار گرفته اند ذخیره می شود . نوار مغناطیسی توسط دستگاه نوار خوان به حرکت در می آید . هر دستگاه نوار خوان دارای یک نوک خواندن و نوشتن می باشد . نوار مغناطیسی اصولاً برای پردازش ترتیبی یا پی در پی اطلاعات استفاده می گردد .

نکته : در نوار برای هر کاراکتر یک بیت parity وجود دارد . که به آن بیت parity عرضی یا کاراکتری می گویند و برای هـر بـلاک نیز یک بیت parity وجود دارد که به آن بیت parity طولی می گویند .



دستگاه نوارخوان مجهز به نوک خواندن/نوشتن است که می‌تواند اطلاعات را از/برروی نوار بخواند یا بنویسد.

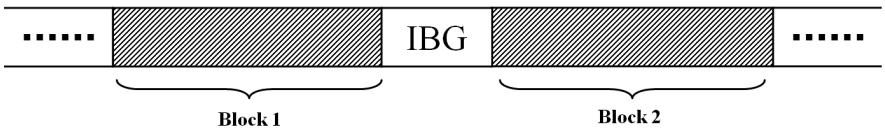


چگالی نوار (Density) :

تعداد بیت قابل ضبط در هر اینچ را چگالی نوار می گویند . چگالی را با واحد بیت در اینچ (bpi) تعریف می کنند .

Gap :

فضایی است بلا استفاده (هرز - waste) بین دو رکورد یا دو بلاک که برای تنظیم سرعت حرکت هد خواندن و نوشتن استفاده می شود . به فضای هرز بین دو رکورد IRG (Inter Record Gap) . به فضای هرز بین دو بلاک IBG (Inter Block Gap) گفته می شود .



نکته : در فضای Gap هیچ اطلاعاتی خوانده یا نوشته نمی شود .

پارامترهای اساسی نوار :

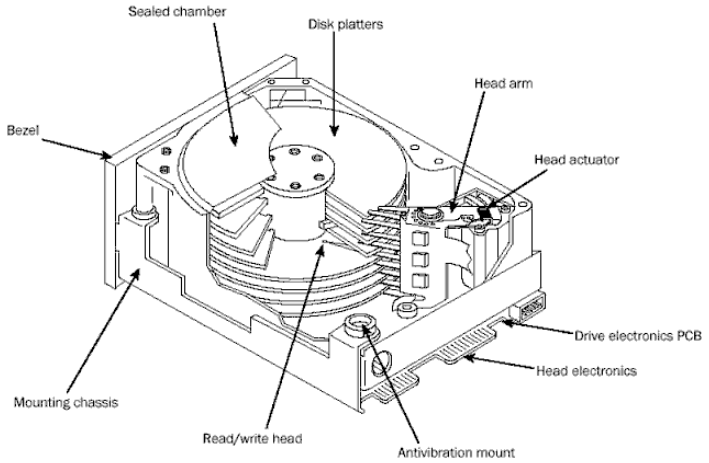
- ✓ سرعت
- ✓ چگالی
- ✓ نرخ انتقال

که در واقع می توان دسته بندی دیگری نیز بیان کرد .

- ✓ پارامترهای ظرفیتی (چگالی ، طول نوار)
- ✓ پارامترهای زمانی (سرعت ، نرخ انتقال ، زمان حرکت ، توقف)

دیسک مغناطیسی :

نوار مغناطیسی دارای محدودیت هایی است . یکی از آنها نوع قرار گرفتن داده ها بر روی نوار به صورت ترتیبی می باشد که امکان دسترسی مستقیم به اطلاعات را مقدور نمی سازد . دیسک مغناطیسی رسانه ای است با امکان دستیابی مستقیم به داده های ذخیره شده . دیسک مغناطیسی شامل صفحه ای است دوار حول یک محور عمودی بر روی این صفحه دوایر با شیارهای متحدالمرکزی وجود دارد که داده ها به صورت سریالی در این شیارها قرار می گیرد . اطلاعات بر روی این شیارها توسط بازوی خواندن و نوشتن نوشته یا از آن خوانده می شود .



دسته بندی دیسک ها :

دیسک ها را از نظر نقطه نظرهای مختلفی رده بندی می کنیم .

۱) **از نظر جا به جا شدن :** دیسک های ثابت و دیسک های جا به جا شدنی

۲) **از نظر ثابت بودن یا متحرک بودن :** هد خواندن و نوشتن که به دو دسته ی دیسک ها با نوک ثابت و دیسک ها با نوک متحرک تقسیم می شوند .

در دیسک ها با نوک ثابت ، بازویی که نوک خواندن و نوشتن به آن متصل است در آن حرکت نمی کند و در دیسک هایی با نوک متحرک این بازو بر روی رویه ی دیسک در مسیر شعاع دیسک از شیاری به شیاری دیگر حرکت می نماید

.در دیسک ها با نوک ثابت هر شیار هد خواندن و نوشتن مخصوص خود را دارد و دیگر نیازی به حرکت بازو از شیار به شیار دیگر نیست . اینگونه دیسک ها سریعتر و البته گران ترند .

۳) **از نظر تعداد صفحاتی که روی محور عمود قرار می گیرند به دو دسته تقسیم می شوند :** دیسک های تک صفحه ای دیسک های چند صفحه ای به دیسک های چند صفحه ای اصطلاحاً PACK می گویند .

یک PACK با n صفحه دارای 2n رویه است . که 2n-2 رویه برای ذخیره سازی اطلاعات و دو رویه ی بالایی و پایینی برای حفاظت و کنترل بیت ها استفاده می شود .

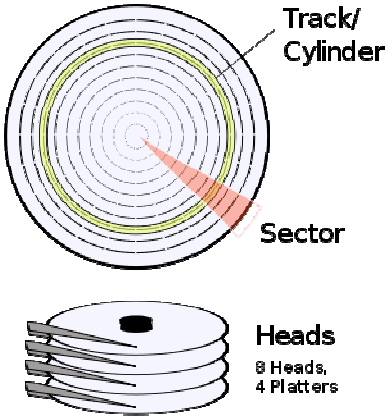
۴) **از نظر جنس صفحه** به دو دسته ی دیسک سخت (Hard disk) و دیسک انعطاف پذیر یا نرم (Floppy disk) تقسیم می شوند .

تقسیمات دیسک :

۱) **استوانه یا سیلندر :** شیارهای هم شعاع بر روی صفحه های هم مرکز بر روی رویه های مختلف سیلندر می گویند .

۲) **شیار یا Track :** به دوائر متحدالمرکز که اطلاعات بر روی آنها در قالب صفر و یک ذخیره می شود شیار گفته می شود .

۳) **سکتور یا قطاع :** تقسیمات از شیار به اندازه های مساوی را سکتور می گویند .



دو نوع سکتور وجود دارد :

۱) سکتور سخت افزاری که توسط سازنده ی دیسک ایجاد می شود .

۲) سکتور نرم افزاری که توسط سیستم فایل یا سیستم عامل ایجاد می گردد و ضریبی از سکتور سخت افزاری است .

همانگونه که از تعریف بالا بر می آید با توجه به دایرهای بودن شیارها ممکن است این مطلب استنباط گردد که سکتورهای خارجی به دلیل طول بیشتر میزان اطلاعات ذخیره شده (چگالی اطلاعات) بیشتری نسبت به شیارهای داخلی دارد . در حالی که این مطلب صحت نداشته و در تمام شیارها میزان اطلاعات ذخیره شده یکسان می باشد و فقط در شیارهای بیرونی ، فضای هرز بیشتری خواهیم داشت . (البته در دیسک های امروزی شیارهای بیرونی به تعداد سکتورهای بیشتری نسبت به شیارهای درونی ، تقسیم می شوند که این امر موجب به هدر رفتن کمتر اطلاعات می گردد .)

پارامترهای دیسک را به دو دسته کلی زیر تقسیم بندی می کنیم :

۱) **پارامترهای ظرفیتی** : شامل سکتور در شیار ، تعداد شیارهای هر رویه یا صفحه ، تعداد صفحات دیسک .

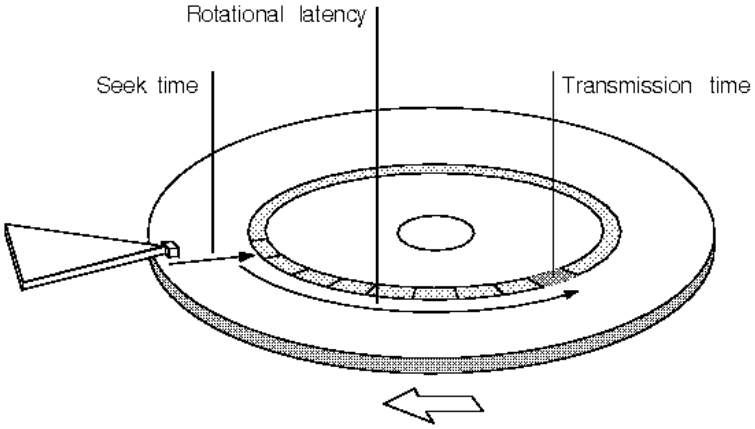
۲) **پارامترهای زمانی** : که عبارتند از زمان استوانه جوئی ، زمان انتظار دورانی ،سرعت گردش دیسک و نرخ انتقال .

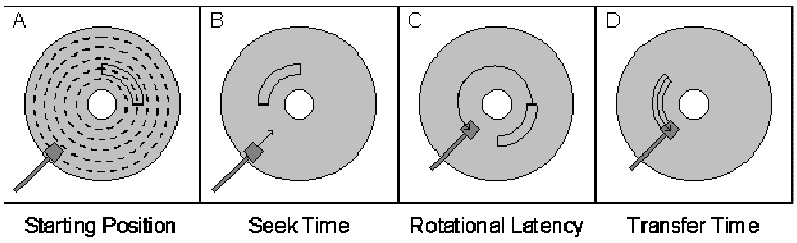
زمان استوانه جوئی (Seek Time) : زمانی است که جهت رسیدن نوک خواندن و نوشتن و قرار گرفتن بر روی استوانه ای که اطلاعات مورد نظر ما بر روی آن قرار دارد صرف می شود و آنرا با s نمایش می دهند و واحد ان میلی ثانیه است .

زمان انتظار دورانی : زمانی است که پس از رسیدن نوک خواندن و نوشتن بر روی استوانه ی مورد نظرباید سپری شود تا داده ی مورد نظر به زیر نوک خواندن و نوشتن برسد و واحد ان میلی ثانیه است و متوسط آن را r با نمایش می دهیم که نصف زمان لازم برای یک دور دیسک (2r) می باشد. $(0 \le r < 2r)$

سرعت گردش دیسک : مقدار دورهای دیسک در دقیقه می باشد و با rpm (Rotation per Minute) نمایش داده می شود و واحد ان دور به دقیقه است .

نرخ انتقال : تعداد بایتی است که در یک ثانیه به رسانه منتقل و یا از رسانه به سیستم منتقل می شود .





فرمت بندی شیارها در دیسک :

این فرمت بندی در سیستم های مختلف ، متفاوت است . به طور کلی دو دسته فرمت بندی داریم :

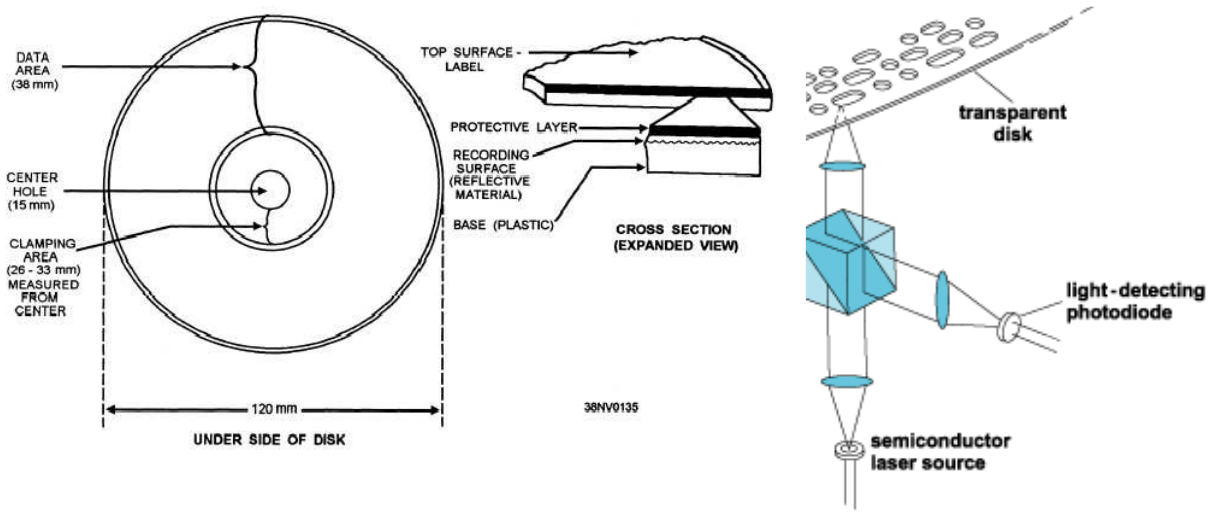
۱) سخت افزاری (Hard Sector)

۲) نرم افزاری (Soft Sector)

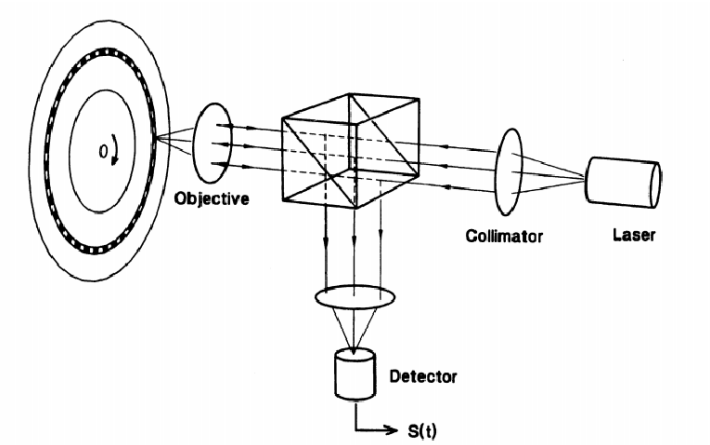
در فرمت بندی سخت افزاری تقسیم بندی های دیسک از قبیل شیار و سکتور از قبل مشخص بوده و توسط شرکت سازنده تعیین می گردد و قابل تغییر نمی باشد در حالی که در فرمت بندی نرم افزاری این تنظیمات را از طریق نرم افزار انجام می دهیم که اطلاعات موجود در مورد هر داده می تواند شامل اطلاعات حفاظتی ، طول داده ، بیت های parity و ... می باشد .

دیسک نوری :

در این دیسک ها برای ذخیره سازی اطلاعات ، از نور لیزر به جای ، مغناطیسی کردن استفاده می گردد که استفاده از این روش این مزیت را دارد که فضای کمتری برای ذخیره سازی یک بیت مصرف می شود.

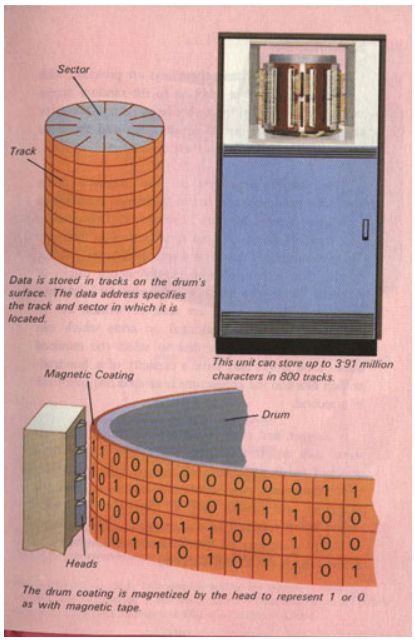


برای خواندن اطلاعات ذخیره شده بازوی اپتیکی تغییرات بازتاب را به سیگنال الکتریکی تبدیل می‌کند. یک عدسی در داخل بازو پرتو کم توان لیزر را به لکه کوچک نوری بر روی مسیر متمرکز می‌کند و همچنین نور بازتاب شده از دیسک را مجددا به آشکار ساز نوری هدایت می‌کند. خروجی آشکار ساز نوری بر اساس توزیع گودالهای طول مسیر تغییر می‌کند و سیگنال الکتریکی بدست می‌دهد که می‌توان سیگنال صدا ، تصویر و یا داده‌ها را دوباره بدست آورد.



طبله یا Drum :

رسانه ای است معادل با دیسک ، البته با نوک ثابت . در واقع یک استوانه است که بر روی سطح آن دارای شیارهایی است و اطلاعات بر روی این شیارها ذخیره می گردد . برای خواندن اطلاعات در این دستگاه برای هر شیار یک نوک خواندن و نوشتن وجود دارد بنابراین در این رسانه زمان استوانه جوئی برابر صفر می باشد .



فصل دوم :

رکورد بلاک و روشهای بلاک بندی

رکورد :

مجموعه اطلاعاتی است که در مورد یک نوع موجودیت در یک محیط عملیاتی نگهداری می کنیم .

محیط عملیاتی :

منظور محیطی است که می خواهیم سیستم ذخیره و بازیابی را برای آن ایجاد کنیم . به عنوان نمونه مـی تـوانیم حسـابداری یـک سازمان یا فعالیتهای ثبت نام یک دانشگاه را به عنوان یک محیط عملیاتی در نظر بگیریم .

موجودیت :

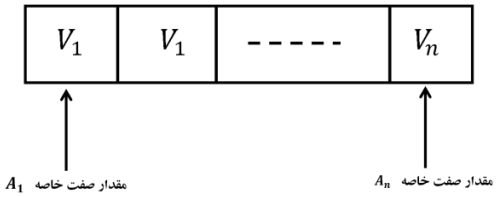
مصداق کلی پدیده ، فرد ، شئ یا مفهومی که می خواهیم در مورد آن اطلاعات داشته باشیم . بدیهی است که ما برای هـر محـیط عملیاتی نیاز به موجودیت هایی برای آن محیط خواهیم داشت .

وجه تمایز بین موجودیت ها توسط صفات خاصه (Attribute) مشخص می گردد . به عنوان نمونه موجودیـت دانشـجو دارای صـفات خاصه شماره دانشجوئی ، نام و نام خانوادگی ، رشته تحصیل و ... می باشد .

مدلهای مختلف رکورد در پیاده سازی :

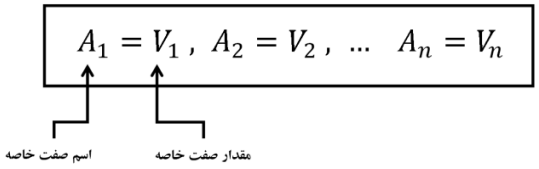
برای پیاده سازی رکورد ، دو طرح کلی وجود دارد که عبارتند از :

۱- **ذخیره سازی رکوردها با قالب ثابت و مکانی :** در این طرح ، رکورد مجموعه است از فیلـدها و در هـر فیلـد یـک مقـدار صفت خاصه ذخیره می گردد . در این روش تعداد فیلدها و طول هر فیلد در تمام نمونه های رکورد یکسان می باشد .



این روش دارای معایبی می باشد . از جمله اینکه اگر مقداری که می خواهید ذخیره کنید از مقدار تعیین شده برای فیلد کوتاهتر باشد ، بقیه فضای در نظر گرفته شده بلا استفاده می ماند که این فضای هرز و بلا استفاده باعـث بزرگتـر شـدن فایل و به هدر رفتن حافظه ی دیسک می گردد . همچنین این روش برای رکوردهایی که ممکن است طول آنهـا متغیـر باشد ، چندان مناسب نیست ، ولی مزیت مهم این روش دسترسی آسان و ساده در هنگام خواندن و نوشتن می باشد .

۲- **ذخیره سازی رکوردها با قالب ثابت و غیر مکانی :** تعداد فیلدها ، طول آنها و همچنین محل قرار گیری آنها در رکورد ، در نمونه های مختلف رکورد متفاوت می باشد .



دلایل متغیر شدن طول رکورد :

۱) ممکن است که طول برخی از فیلدها متغیر باشد . مثلاً ، فیلد مربوط به نام و نام خانوادگی یا آدرس که می تواند طول متغیر داشته باشد .

۲) ممکن است که برخی از فیلدها دارای چندین صفت خاصه باشند که در نمونه های مختلف آن تعداد مقادیر متفاوتی داشته باشد به عنوان نمونه دروس انتخاب شده در یک ترم در رکورد اطلاعاتی دانشجو،که تعداد دروس برای فیلد ثبت نام متفاوت می باشد .

۳) ممکن است برای رکورد ، حالت مختلفی از فیلدها (تعداد فیلدها متغیر است) را دارا باشیم . به عنوان نمونه در رکورد اطلاعاتی مربوط به کارمندان یک سازمان ، فیلدهای کارمندان رسمی و کارمندان قراردادی با یکدیگر متفاوت می باشد .

رکورد را از سه دیدگاه بررسی می کنیم :

۱) **رکورد در سطح مفهومی یا انتزاعی** : که این دید مربوط به طراح سیستم ذخیره و بازیابی است .

۲) **رکورد در سطح منطقی** : در واقع همان رکوردی است که توسط برنامه نویس تعریف می گردد و شامل مجموعه ای از فیلدهای داده ای می باشد .

۳) **رکورد در سطح فیزیکی** : که مربوط به نحوه قرار گیری داده ها بر روی محیط عملیاتی می باشد .

رکورد در محیط ذخیره سازی (رکورد فیزیکی) :

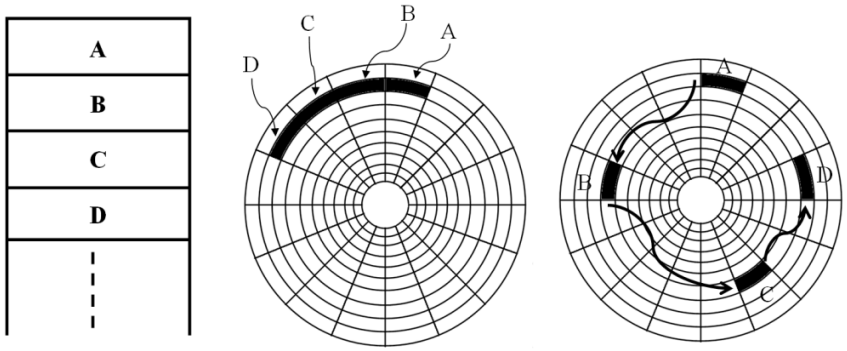
در محیط ذخیره سازی ، رکورد از دو قسمت ، داده ای و غیر داده ای تشکیل شده است .

بخش داده‌ای	بخش غیرداده‌ای

در بخش داده ای اطلاعات مورد نظر ما ذخیره می گردد و در بخش غیر داده ای حاوی اطلاعاتی است که سیستم فایل برای ثبت رکورد در محیط فیزیکی به آنها نیاز دارد که بسته به نوع سیستم های فایلینگ متفاوت است . به عنوان نمونه می توان گفت غیر داده ای شامل اطلاعاتی نظیر طول رکورد ، نوع رکورد ، اشاره گرها و Flagهای عملیاتی می باشند که در زیر به شرح آنها خواهیم پرداخت .

- **طول رکورد** : در هنگامی که رکوردها دارای طول متغیرها هستند از این فیلد برای نگهداری و مشخص کردن محدوده یا طو بخش داده ای (اطلاعات مورد نظر ما جهت ذخیره سازی) می باشد.ولی در رکوردها یی با طول ثابت به این فیلد نیازی نمی باشد.
- **نوع رکورد** : برای فایل هایی که دارای چندین نوع رکورد می باشند از این فیلد جهت مشخص کردن نوع رکورد استفاده می شود.
- **Flagهای عملیاتی (پرچم)** : یک نشانگر برای فیلدهاست، عملیاتی که قرار است بررسی آن انجام گیرد یا انجام شده است را نگهداری می کند.مثلاً در یک فایل، عمل حذف به صورت منطقی انجام گیرد. فلگ مربوطه را علامت گذاری می کنیم تا بعد در هنگام یکپارچه سازی رکوردهای حذف شده منطقی را به صورت فیزیکی حذف می کنیم.

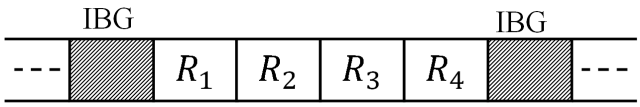
- **اشاره گرها** : از این فیلد جهت ارتباط منطقی بین رکوردهای فایل استفاده می شود.در واقع از این فیلد در پیاده سازی ساختار منطقی فایل در محیط فیزیکی استفاده می شود . ساختار منطقی فایل، همان دیدی است که کاربر نسبت به فایل دارد . یعنی یک مجموعه ای از رکوردها که با نظم خاصی و بر اساس صفت خاصی به طور مرتب، پشت سر هم ذخیره شده اند . ولی در هنگام ذخیره سازی رکوردها، لزوماً دارای همجواری فیزیکی نیستند . لذا ما از این فیلد، جهت برقراری ارتباط منطقی بین رکوردهای ی که از نظر فیزیکی همجوار نیستند، استفاده می کنیم .



فایل منطقی و همجواری رکوردها　　همجواری فیزیکی رکوردها　　ناهمجواری فیزیکی رکوردها

بلاک بندی (Blocking) :

قراردادن چندین رکورد را در یک قالب بزرگتر جهت عملیات خواندن و نوشتن از رسانه ذخیره سازی بلاک بندی می گویند.در واقع بلاک مجموعه ای است از تعدادی رکوردها طول ثابت .بلاک کمترین داده ای است که در یک عملیات خواندن و نوشتن به یا از دیسک منتقل وی شود.بین هردو بلاک فضای هرزی وجود دارد که به آن(Inter Block Gap) IBG گفته می شود.ظرفیت بلاک را با B نمایش می دهیم.



ضریب بلاک بندی (blocking factor) :

به تعداد رکوردهای موجود در بلاک بندی گفته می شود و آن را با B_f نمایش می دهیم.

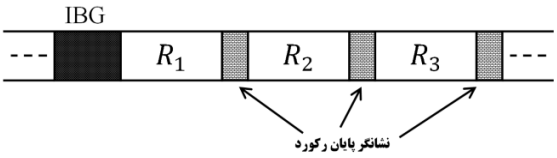
حالات مختلف ذخیره سازی یک بلاک بر روی دیسک :

۱) شیار
۲) یک سکتور سخت افزاری
۳) ترکیبی از چند سکتور سخت افزاری
۴) بخشی از شیار مشخص شده توسط نرم افزار (سکتورنرم افزاری)

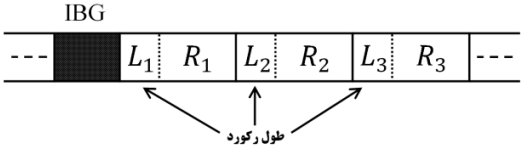
تکنیک های تعیین محدوده ی رکورد با طول متغیر در بلاک :

اگر طول رکورد ثابت باشد ، سیستم فایل یکبار طول رکورد را در سیستم ذخیره کرده و از آن برای تمامی نمونه های آن رکورد در بلاک استفاده می کند.ولی اگر طول رکورد متغیر باشد، از سه تکنیک زیر برای مشخص نمودن محدوده رکـورد در بـلاک اسـتفاده می کنیم.

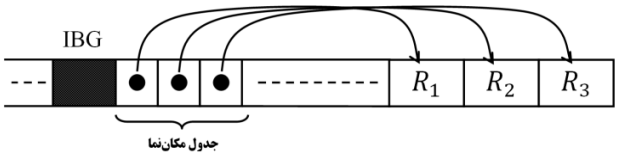
۱) درج نشانگر در پایان رکورد



۲) درج طول در بخش پیوندی



۳) ایجاد جدول مکان نما : در این روش ، آدرس هر رکورد نسبت به ابتدای بلاک در جدولی ذخیره می‌شود.

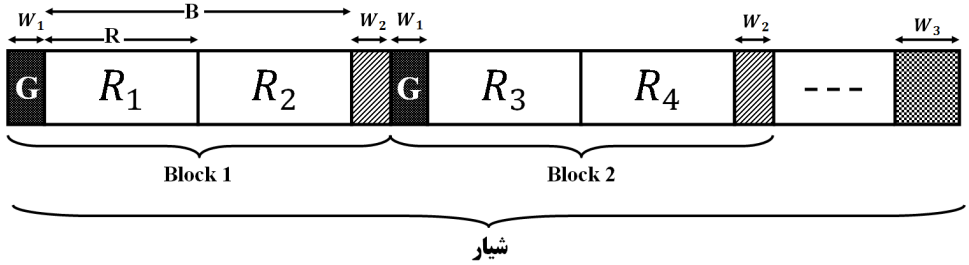


تکنیک های بلاک بندی :

۱) بلاک بندی رکوردها با طول ثابت و یکپاره
۲) بلاک بندی رکوردها با طول متغیر و یکپاره
۳) بلاک بندی رکوردها با طول متغیر و دوپاره

در هریک از این روش ها یک فضای هرز Waste در شیار ایجاد می شود و همچنین تعداد رکوردهایی که در یک بلاک جای مـی-گیرند متفاوت است.دقت داشته باشید در مسائل زیر فضای هرز را با w نمایش می دهیم.

روش اول : تکنیک بلاک‌بندی رکوردهای با طول ثابت و یکپاره



در این تکنیک سه نوع فضای هرز بوجود خواهد آمد:

۱- W_1 : فضای هرز مربوط بهGAP که با G نمایش می‌دهیم.

۲- W_2 : فضای هرز ناشی از نگنجیدن آخرین رکورد در بلاک. (به علت ثابت بودن طول رکورد)

۳- W_3 : فضای هرز ناشی از نگنجیدن آخرین بلاک در شیار

B : سایز بلاک **R** : سایز رکورد

ضریب بلاک‌بندی $B_f = \left\lfloor \frac{B}{R} \right\rfloor$

حافظه هرز به‌ازی هر بلاک برابر است با : $W_B = W_1 + W_2$

و در صورتیکه که W_3 نیز داشته باشیم آنگاه : $W_B = W_1 + W_2 + \frac{W_3}{T_f}$

که در آن T_f فاکتور (ضریب) شیاربندی می‌گوییم و برابر تعداد بلاکهای موجود در شیار می‌باشد.

در صورتیکه برای W_2 مقدار متوسط آن یعنی $W_2 = \frac{R}{2}$ درنظر بگیریم خواهیم داشت:

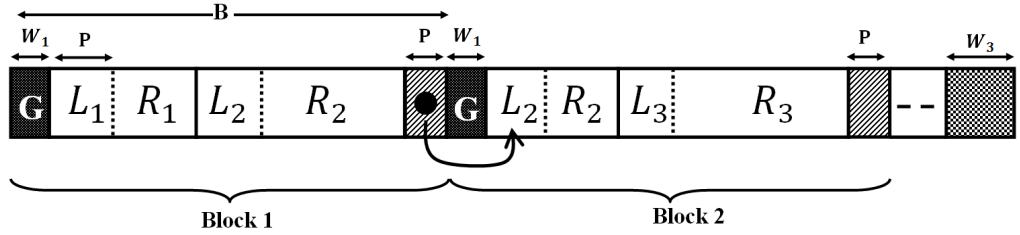
$$W_B = G + \frac{R}{2} + \frac{W_3}{T_f}$$

و در نتیجه با توجه به اینکه $W_R = \frac{W_B}{B_f}$ خواهیم داشت :

$$W_R = \frac{1}{B_f}\left(G + \frac{R}{2} + \frac{W_3}{T_f}\right)$$

روش دوم : تکنیک بلاک‌بندی رکوردهای با طول متغیر و دوپاره

همانند بلاک‌بندی رکوردها با طول ثابت است با این تفاوت که جدول طول‌ها باید موجود باشد (بعلت متغیر بودن سایز رکوردها) و دیگر تفاوت آن را روش فبل در این است که درصورتیکه که یک رکورد در بلاک بصورت یکپاره نتواند ذخیره شود، می‌توان آنرا بـه دو قسمت کرد.



همانگونه که در شکل فوق مشاهده می‌کنید، R2 تماماً در بلاک اول جا نمی‌گیرد، بنابراین به‌صورت دوپاره در بلاک اول و دوم ذخیره می‌شود.

فاکتور بلاک‌بندی در این روش برابر است با : $B_f = \left\lfloor \frac{B-P}{R+P} \right\rfloor$

که در فرمول فوق P سایز فیلد مربوط به طول رکوردها می‌باشد که از نظر سایز با سایز فیلد اشاره‌گر در انتهای بلاک یکی درنظر گرفته می‌شود.

و میزان حافظه هرز به‌ازی هر بلاک برابر است با :

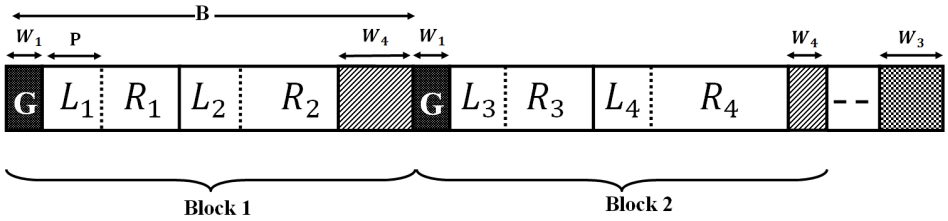
$$W_B = G + P + B_f \cdot P + \frac{W_3}{T_f}$$

و در نتیجه با توجه به اینکه $W_R = \frac{W_B}{B_f}$ خواهیم داشت :

$$W_R = \frac{1}{B_f}\left(G + P + B_f \cdot P + \frac{W_3}{T_f}\right) = P + \frac{G+P}{B_f} + \frac{W_3}{B_f \cdot T_f}$$

روش سوم : تکنیک بلاک‌بندی رکوردهای با طول متغیر و یکپاره

در این تکنیک حافظه هرز جدیدی بنام W_4 بعلت اینکه R2 را دوپاره نکرده‌ایم بوجود آمده است.



متوسط اندازه W_4 را برابر $\frac{R}{2}$ درنظر می‌گیریم.

فاکتور بلاک‌بندی در این روش برابر است با : $B_f = \left\lfloor \frac{B-W_4}{R+P} \right\rfloor = \left\lfloor \frac{B-\frac{R}{2}}{R+P} \right\rfloor$

و میزان حافظه هرز به‌ازی هر بلاک برابر است با :

$$W_B = G + B_f \cdot P + W_4 + \frac{W_3}{T_f} = G + B_f \cdot P + \frac{R}{2} + \frac{W_3}{T_f}$$

و در نتیجه با توجه به اینکه $W_R = \frac{W_B}{B_f}$ خواهیم داشت :

$$W_R = \frac{1}{B_f}\left(G + B_f \cdot P + W_4 + \frac{W_3}{T_f}\right) = P + \frac{G + \frac{R}{2}}{B_f} + \frac{W_3}{B_f \cdot T_f}$$

مزایا و معایب این سه روش :

در روش اول پیاده سازی و مدیریت راحت تر است ولی در صورتی که طول رکوردها عوض نشود بـه همـین دلیـل روش اول فاقـد انعطاف پذیری می باشد ولی در روش دوم و سوم نسبت به روش اول انعطاف پذیری بیشتری بیشتری وجـود دارد ولـی نـرم افـزار پیچیده تری نیاز می باشد.

در مقایسه دو روش دوم و سوم، روش سوم از نظر حافظه مقرون به صرفه تر می باشد.ولی به نرم افزاار پیچیـده تـری نیـاز دارد.در تکنیک های اول و دوم مشکلی که مطرح است این است که حد اکثر طول رکورد به طول بلاک محدود می شود یعنـی در مرحلـه ایجاد فایل باید حداکثر طول رکورد را برابر طول بلاک درنظر بگیریم در حالیکه تکنیک دوم این محدودیت را ندارد و از ایـن نظـر انعطاف‌پذیرتر است.

از نظر زمانی به علت اینکه روش سوم حافظه ی هرز بیشتری دارد طول فایل بیشتر از روش دوم می باشد.برای خواندن تمام فایـل و یا پردازش آن زمان بیشتری صرف می کنیم.

مزایای بلاک بندی :

۱- کاهش دفعات خروجی/ورودی و در نتیجه کاهش زمان برنامه

۲- صرفه جوئی در مصرف حافظه ی دیسک به علت کم شدن میزان Gap ها . چرا که در صورت اسـتفاده نکـردن از بـلاک بنـدی، رکوردها دارای Gap بوده که سبب افزایش طول فایل و در نتیجه افزایش زمان پردازش خواهد شد .

معایب بلاک بندی :

۱- مصرف حافظه ی اصلی به علت نیاز به Buffering

۲-کار نرم افزاری بیش تر برای بلاک بندی و بلاک گشائی

مسائل:

۱) در یک فایل طول بلاک **1700** بایت می‌باشد. درصورتیکه طول هر رکورد را ثابت و برابر **300** درنظر بگیریم، ضریب بلاک-بندی در این فایل را محاسبه کنید.

$$B_f = \left\lfloor \frac{B}{R} \right\rfloor = \left\lfloor \frac{1700}{300} \right\rfloor = 5$$

---

۲) برای بلاک‌بندی رکوردهای یک فایل از روش بلاک‌بندی رکوردها با طول متغیر و دوپاره استفاده مـی‌شـود. درصـورتیکه طول هر بلاک در این سیستم برابر **1140** بایت و طول متوسط رکورد برابر **180** بایت درنظر بگیریم، ضریب بلاک‌بندی در این روش چقدر است؟ (با فرض اینکه سایز فیلد نگهداری طول رکورد و اشاره‌گر انتهای بلاک برابر **20** بایت می‌باشد.

$$B_f = \frac{B-P}{R+P} = \frac{1140-20}{180-20} = \frac{1120}{160} = 7$$

---

۳) درسیستم‌فایلی از روش بلاک‌بندی رکوردها با طول ثابت و یکپاره استفاده می‌شود. اگر فضای هرز بین بلاکـی **19** بایـت، متوسط فضای هرز ناشی از نگنجیدن رکورد در بلاک **86** بایت و فضای هرز ناشی از نگنجیدن بلاک در شـیار **360** بایـت درنظربگیریم، با فرض $T_f = 9$ ، مطلوبست یافتن فضای هرز به‌ازی هر رکورد.

$$W_B = G + \frac{R}{2} + \frac{W_3}{T_f} \quad \Rightarrow \quad W_B = 19 + 86 + \frac{360}{9} = 18 + 86 + 40 = 145\ Byte$$

---

۴) در یک رسانه ذخیره‌سازی، اگر ضریب بلوک‌بندی برابر **20** و میزان فضای هرز هر بلاک برابر **320** بایت باشد، فضای هـرز هر رکورد در این رسانه را محاسبه کنید.

$$W_R = \frac{1}{B_f} \times W_B = \frac{1}{20} \times 320 = 16\ Byte$$

---

۵) در روش بلاک‌بندی رکوردها با طول متغیر و یکپاره، اگر $2W_4 = 2P = G$ و فضای هرز انتهای هر بلاک **18** بایت و فضای هرز انتهای شیار صفر و ضریب بلاک‌بندی **5** فرض می‌شود، مطلوبست یافتن میزان فضای هرز به ازای هر رکورد.

$W_3 = 0$ و $B_f = 5$ و $W_4 = 18$

$$W_B = G + P \times B_f + W_4 + \frac{W_3}{T_f} \quad \Rightarrow \quad W_B = 2W_4 + \frac{2}{3}W_4 \times 5 + W_4 + 0 \quad \Rightarrow \quad W_B = 2 \times 18 + \frac{10}{3} \times 18 + 18$$

$$\Rightarrow \quad W_B = 114\ Byte$$

---

۶) اگر طول بلاک داده به‌همراه بخش کنترلی آن **57** بایت باشد و $W_2 = 2W_1$ و $W_3 = 3W_1$ و طول شکاف بـین بلاکـی **6** بایت فرض شود، رابطه بین حافظه هرز بلاکی با ظرفیت اسمی شیار C_N را بدست آورید.

$$W_B = W_1 + W_2 + \frac{W_3}{T_f} \quad \text{و} \quad T_f = \frac{C_N}{B+C} \quad \Rightarrow \quad W_B = W_1 + 2W_1 + \frac{3W_1}{\frac{C_N}{B+C}}$$

$$\Rightarrow \quad W_B = 3W_1 + \frac{3W_1 \times (B+C)}{C_N}$$

$$\Rightarrow \quad W_B = 3 \times 6 + \frac{3 \times 6 \times (57)}{C_N} = 18\left(1 + \frac{57}{C_N}\right)$$

---

۷) در سیستم‌فایلی، بلاک‌ها بگونه‌ای انتخاب شده‌اند که تمامی فضای شیار را پر می‌نمایند. اگر طول هر رکورد و بلاک بترتیب برابر **90** و **425** بایت درنظر گرفته شود. با فرض آنکه طول هر شکاف بین بلاکی **19** بایت باشد، فضای هرز هر رکورد را بدست آورید.

$$W_1 = G = 19\ (B) \quad \text{و} \quad W_2 = \frac{R}{2} = \frac{90}{2} = 45\ (B) \quad \text{و} \quad W_3 = 0$$

$$W_R = \frac{1}{B_f}\left(W_1 + W_2 + \frac{W_3}{T_f}\right) \quad \text{و} \quad B_f = \left\lfloor \frac{B}{R} \right\rfloor = \left\lfloor \frac{425}{90} \right\rfloor = 4$$

$$W_R = \frac{1}{4}(19 + 45 + 0) = \frac{64}{4} = 16\ (B)$$

---

۸) در یک فایل که با استفاده از رکوردهای با طول ثابت، بلوک‌بندی شده است، اطلاعات زیر موجود است:

$W_R = 15\ (B)$ $\quad R = 42\ (B)$ $\quad B_f = 4$

$W_1 = 12\ (B)$ $\quad C_N = 350\ (B)$ $\quad W_3 = 8W_1$

مطلوبست یافتن طول بخش کنترلی هر بلاک (با فرض اینکه هر بلاک بطور کامل از رکوردها پرشده است)

$$W_R = \frac{1}{B_f} \times W_B \quad \rightarrow \quad 15 = \frac{1}{4} \times W_B \quad \Rightarrow \quad W_B = 60\ (B)$$

$$W_B = G + W_2 + \frac{W_3}{T_f} \quad \text{و} \quad G = W_1 = 12 \quad \text{و} \quad W_3 = 8W_1 = 98$$

$$\Rightarrow \quad 60 = 12 + 0 + \frac{96}{T_f} \quad \Rightarrow \quad 48 = \frac{96}{T_f} \quad \Rightarrow \quad T_f = 2 \quad \text{و} \quad B = R \times B_f = 42 \times 4 = 168$$

$$T_f = \frac{C_N}{B+C} \quad \Rightarrow \quad 2 = \frac{350}{168 + C} \quad \Rightarrow \quad C = 7\ (byte)$$

---

۹) درصورتیکه از روش رکوردها با طول متغیر و یکپاره در بلاک‌بندی استفاده کنیم، اگر طول هر رکورد **10P** که **P** سایز قسمت اندازه طول رکورد است و طول بلاک را **190** بایت درنظر بگیریم، مطلوبست یافتن ضریب بلاک‌بندی در این سیستم فایل.

$$B_f = \frac{B - R/2}{R + P} = \frac{B - 5P}{11P} = \frac{190 - 25}{55} = \frac{165}{55} = 3$$

---

۱۰) اگر در روش بلاک‌بندی رکوردها با طول متغیر و یکپاره، طول بلاک داده برابر **76** بایت و فضای هرز بلاکی برابر **20** بایت باشد و **R=6P** به‌ازای **P=2** بایت برقرار باشد، فضای هرز یک رکورد را بدست آورید.

$$B_f = \frac{B - R/2}{R+P} = \frac{76-3P}{7P} = \frac{76-3\times 2}{7\times 2} = \frac{70}{14} = 5$$

$$W_R = \frac{1}{B_f}\times W_B = \frac{1}{5}\times 20 = 4\ (Byte)$$

---

۱۱) درمسئله قبل اگر طول رکورد سه‌برابر شود، فضای هرز هر رکورد را بدست آورید و با حالت قبل مقایسه کنید.

$$B_f = \frac{B - R/2}{R+P} = \frac{76-9P}{19P} = \frac{76-9\times 2}{7\times 2} = \frac{58}{38} \cong 2$$

$$W_R = \frac{1}{B_f}\times W_B = \frac{1}{2}\times 20 = 10\ (Byte)$$

باتوجه به آنکه فضای هرز انتهای بلاک برابر **R/2** می‌باشد پس با افزایش میزان سایز رکورد فضای هرز انتهای بلاک افزایش یافته و در نتیجه بر روی فضای هرز به ازای هر بلاک و رکورد موثر خواهد بود

---

۱۲) دریک فایل با رکوردهایی با طول متغیر و دوپاره، اطلاعات زیر موجود می‌باشد:

$B = 120\ (B)$ $\quad R = 25\ (B)$ $\quad G = 15\ (B)$

$P = 4\ (B)$ $\quad W_3 = 2G$ $\quad T_f = 3$

مطلوبست یافتن میزان فضای هرز هر بلاک و هر رکورد این فایل.

$$W_B = G + P\times B_f + P + \frac{W_3}{T_f} \quad \text{و} \quad B_f = \frac{B-P}{R+P} \Rightarrow B_f = \frac{120-4}{25+4} = 4$$

$$W_B = 15 + 4\times 4 + 4 + \frac{2\times 15}{3} = 15 + 16 + 4 + 10 = 45$$

$$W_R = \frac{1}{B_f}\times W_B = \frac{1}{4}\times 45 \cong 11\ (Byte)$$

---

۱۳) با مفروضات زیر، درصورتیکه از بلاک‌بندی رکوردها با طول ثابت و یکپاره استفاده کنیم،

$B = 68\ (B)$ $\quad C = 18\ (B)$ $\quad C_N = 430\ (B)$

$R = 16\ (B)$ $\quad W_1 = 3\ (B)$ $\quad W_3 = 85(B)$

مطلوبست محاسبه میزان حافظه هرز به‌ازای هر رکورد

$$T_f = \frac{C_N}{B+C} = \frac{430}{68+18} = \frac{430}{86} = 5 \quad \text{و} \quad W_B = W_1 + W_2 + \frac{W_3}{T_f} = G + \frac{R}{2} + \frac{W_3}{T_f} = 3 + \frac{16}{2} + \frac{85}{5} = 3 + 8 + 17 = 28\ (B)$$

$$B_f = \left\lfloor \frac{B}{R} \right\rfloor = 4 \quad \Rightarrow \quad W_R = \frac{1}{B_f}\times W_B = \frac{1}{4}\times 28 = 7\ (B)$$

فصل سوم :

سیستم‌فایل

فایل :

فایل مجموعه‌ای از اطلاعات مرتبط به هم است که بر روی حافظه جانبی ذخیره می‌شود، از دید کاربر، فایل کوچکترین واحد تخصیص یافته در حافظه ثانویه است. یعنی داده‌ها نمی‌توانند در حافظه جانبی نوشته شوند مگر اینکه در فایلی قرار گیرند. معمولاً فایل حاوی برنامه و داده‌ها است. فایلهای ممکن است عددی، الفبایی یا دودویی باشند. فایل ممکن است فرمت آزاد داشته باشد مانند فایلهای متنی یا دقیقاً فرمت‌بندی شده باشد مثل فایلهایی که شامل رکوردها هستند.

فایل در دسیک به دو صورت ذخیره می شود .

۱) **پیوسته :** فایل در بلاک های فیزیکی همجوار بر روی دیسک ذخیره می شود.

۲) **ناپیوسته :** برای ذخیره سازی فایل تعدادی بلاک نا همجوار به آن تخصیص داده می شود.

در مورد مزایای روش اول می‌توان به این نکات اشاره کرد که اولا این طرح پیاده‌سازی ساده است چرا که با داشتن آدرس بلاک اول فایل بر روی دیسک می توان به کلیه‌ی بلاک‌های دیگر دسترسی پیدا کرد و ثانیا این نوع ذخیره‌سازی از راندمان عملیاتی بالایی در خواندن فایل برخوردار است ولی روش اول دارای معایبی نیز می باشد. از جمله اینکه برای مشخص کردن تعداد بلاک‌های فیزیکی هم جوار برای تخصیص به فایل حد اکثر اندازه‌ی فایل باید مشخص شود و هم چنین فایل امکان رشد و افزایش حجم نخواهد داشت.

ساختار فایل خود به دوصورت فیزیکی و منطقی تقسیم‌بندی می‌شود. ساختار منطقی فایل نشان دهنده طرحی است که بر اساس آن رکوردهای منطقی پشت‌سرهم قرار دارند. به بیان دیگر، از دید کاربر، ساختار منطقی فایل چگونگی ارتباطات و پیوندهای بین رکوردهای منطقی را نشان می‌دهد. ساختار فیزیکی فایل نشان دهنده چگونگی ذخیره‌سازی بلاک‌ها در رسانه (مثلاً) دیسک است.

صفات فایل :

- **نام :** برای فایل نامی انتخاب می‌شود تا کاربران بتوانند به آن مراجعه کنند. نام فایل رشته‌ای از کاراکترها است. مثلاً TEST.txt نام فایل از دو بخش تشکیل شده است که با نقطه از هم جدا شده‌اند.
- **شناسه :** نامی است که توسط سیستم‌فایل قابل تشخیص است که گاه می‌تواند عددی باشد.
- **مکان فایل :** به دستگاه یا محلی اشاره می‌کند که فایل در آنجا ذخیره شده است.
- **اندازه فایل :** اندازه فعلی فایل بر حسب بایت، کلمه یا بلوک است.
- **حفاظت :** اطلاعاتی که دستیابی به فایل را کنترل می‌کند. مانند اینکه چه کسانی می‌توانند فایل را بخوانند و یا چه کسانی می‌توانند در فایل تغییراتی را اعمال کنند.
- **زمان، تاریخ :** این اطلاعات برای کنترل بیشتر بر روی فایل بکاربرده می‌شود. مثلاً فایل در چه تاریخی ایجاد شده است، آخرین دسترسی به فایل کی بوده است و یا آخرین تغییرات بر روی فایل در چه زمانی انجام شده است و موارد مشابه.

اطلاعات مربوط به فایلها در ساختار دایرکتوری نگهداری می‌شوند که این ساختار نیز در حافظه جانبی قرار دارد. ورودی دایرکتوریها شامل نام فایل و شناسه یکتای آن است. شناسه فایل، مکان صفات دیگر فایل را مشخص می‌کند. برای هر فایل ممکن است که اندازه این دایرکتوری بسته به نوع سیستم فایل از ۱ کیلوبایت تا ۱ مگابایت متغییر باشد.

عملیات فایل :

فایل یک نوع داده انتزاعی (Abstract Data Type) است. هر سیستم فایل نه تنها باید شامل ابزاری برای ذخیره‌سازی داده‌ها به صورت فایل باشد بلکه باید مجموعه‌ای از عملیات که می‌تواند بر روی فایل انجام پذیرد را نیز شامل شود.

عملیات متداول بر روی سیستم‌فایل عبارتند از:

- **ایجاد فایل** : فایل جدیدی تعریف می‌شود و درساختار دایرکتوری قرار می‌گیرد. ایجاد فایل در دو مرحله صورت می‌گیرد : در مرحله اول در سیستم فایل فضایی برای فایل پیدا می‌شود و در مرحله دوم، یک ورودی باید برای فایل در ساختار دایرکتوری ایجاد شود.
- **حذف فایل** : برای حذف فایل در ساختار دایرکتوری جستجو می‌شود، پس از پیدا شدن، فضای آن آزاد شده و از ساختار دایرکتوری حذف می‌گردد.
- **بازکردن فایل** : فایل وقتی باز است که فرآیندی بتواند کاری روی آن انجام دهد. برای خواندن از فایل، سیستم یک اشاره‌گر خواندن را نگهداری می‌کند تا محل بعدی خواندن در فایل مشخص باشد. در بعضی از سیستم‌ها، علاوه بر اشاره-گر خواندن، اشاره‌گر نوشتن نیز وجود دارد که محل بعدی نوشتن در فایل را مشخص می‌کند.
- **بستن فایل** : فایل وقتی بسته است که فرآیند نتواند کاری روی آن انجام دهد.
- **نوشتن در فایل** : فرآیند می‌تواند فایل را به‌روز رسانی (به‌هنگام‌سازی) کند. برای این کار داده جدیدی به فایل اضافه می-کند تا اندازه فایل افزایش یابد یا داده‌های موجود در فایل را تغییر می‌دهد.

سیستم‌های مدیریت فایل

سیستم مدیریت فایل، نرم‌افزاری است که خدماتی را برای کاربران و برنامه‌های کاربردی در استفاده از فایل فراهم می‌کند. به این ترتیب، کاربر یا برنامه‌نویس مجبور نیست برای هر برنامه کاربردی نرم‌افزار ویژه‌ای ایجاد کند.

اهداف سیستم‌های مدیریت فایل عبارتند از:

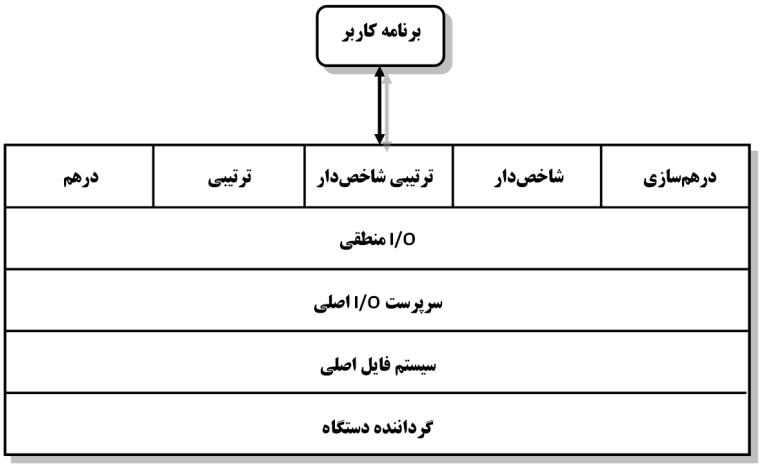
- برآورده کردن نیازهای مدیریت داده‌ها و خواسته‌های کاربر شامل ذخیره‌سازی داده‌ها و توانایی انجام عملیاتی است که گفته شد.
- تضمین معتبر بودن داده‌ها
- بهینه‌سازی کارآیی، هم از نظر سیستم براساس توان عملیاتی و هم از نظر کاربر براساس زمان پاسخ
- پشتیبانی I/O برای انواع مختلفی از دستگاه‌های ذخیره‌سازی
- به‌حداقل رساندن یا از بین بردن احتمال حذف یا تخریب داده‌ها
- تهیه مجموعه از روال‌های واسط I/O استاندارد
- پشتیبانی I/O برای چندین کاربرد در سیستم‌های چند کاربره

در مورد نکته اول (برآورده کردن خواسته‌های کاربر)، گسترش این خواسته به تنوع برنامه‌های کاربردی و محیطی بستگی دارد که کامپیوتر در آن استفاده می‌شود. در سیستم‌های محاوره‌ای و همه‌منظوره، حداقل نیازها عبارتند از:

۱- هرکاربر باید قادر به ایجاد، حذف، خواندن و یا تغییر فایل‌ها باشد
۲- هر کاربر باید بتواند دسترسی کنترل‌شده‌ای برای فایلهای خود برای جلوگیری از دسترسی سایر کاربران داشته باشد.
۳- هر کاربر باید قادر به انتقال داده‌ها بین فایل‌ها باشد.
۴- هر کاربر باید در هنگام خرابی، قادر به تهیه پشتیبان و ترمیم داده‌ها باشد.

معماری سیستم فایل :

یک روش درک حوزه مدیریت فایل، نگاهی به سازمان یک نرم‌افزار سیستم‌فایل است که در شکل زیر مشاهده می‌نمایید. البته در سیستم‌های مختلف این سازمان به اشکال مختلف پیاده‌سازی می‌شود، اما این سازمان به عنوان نماینده منطقی آنها محسوب می‌شود.



درپایین‌ترین سطح **گرداننده‌های دستگاه** مستقیماً با دستگاه‌های جانبی یا کنترل کننده‌های آنها یا کانالها ارتباط برقرار می‌کنند. گرداننده دستگاه مسئول شروع I/O بر روی یک دستگاه و پردازش کامل درخواست I/O است. برای عملیات فایل، معمولاً گرداننده‌های دیسک و نوار کنترل می‌شوند.

سطح بعدی **سیستم فایل اصلی** یا **I/O فیزیکی** نام دارد. این سطح واسط اصلی با دنیای خارج سیستم کامپیوتری است. با بلاک‌های داده‌ایی سروکار دارد که با سیستم دیسک یا نوار مبادله می‌شوند. محتویات داده‌ها یا ساختار فایل را درک نمی‌کند. سیستم فایل اصلی معمولاً بعنوان بخشی از سیستم عامل درنظر گرفته می‌شود.

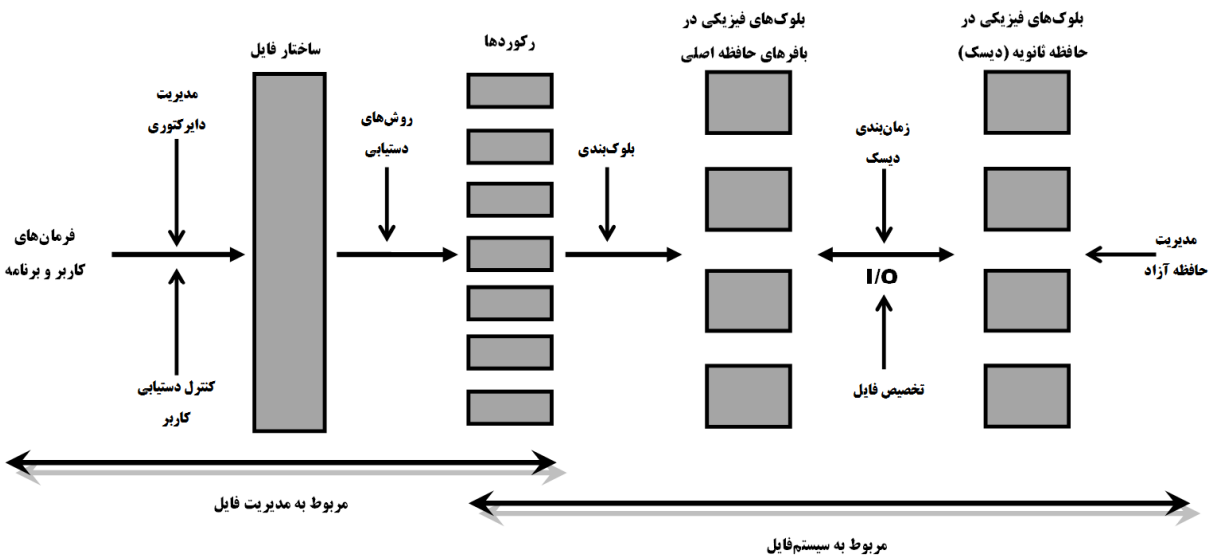
سرپرست I/O اصلی، مسئول تمام عملیات آغازین و پایانی I/O فایل است. دراین سطح، ساختارهای کنترلی نگهداری می‌شوند که با دستگاه I/O، زمان‌بندی و وضعیت فایل سروکار دارند. سرپرست I/O با انتخاب دستگاهی سروکار دارد که عمل I/O باید بر روی آن اجرا شود. این انتخاب، به فایلی که کاربر انتخاب کرده است، بستگی دارد. علاوه بر این به زمانبندی دیسک و دستیابی به نوار برای بهینه‌سازی کارآیی سروکار دارد.

I/O منطقی ، کاربران و برنامه‌های کاربردی را قادر به دستیابی به رکوردها می‌کند. لذا برعکس سیستم فایل اصلی که با بلاک‌هایی از داده سروکار دارد، I/O منطقی با رکوردهای فایل سروکار دارد.

نزدیکترین سطح سیستم فایل به کاربر، **روش دستیابی** نام دارد. واسطهای استانداری را مابین برنامه‌های کاربردی، سیستم فایل و دستگاههای حاوی داده‌ها فراهم می‌کند. روش‌های دستیابی مختلف، ساختارهای فایل گوناگون (ترتیبی، شاخص‌دار و ...) و روش‌های مختلفی را برای دستیابی و پردازش اطلاعات ارائه می‌کند.

وظایف مدیریت فایل

شکل زیر نگاهی مختصر به اعمال سیستم‌فایل دارد. همانگونه که در زیر مشاهده می‌کنید کاربران و برنامه‌های کاربردی از طریق فرمان‌هایی برای ایجاد و حذف فایل‌ها و انجام عملیات بر روی فایل‌ها، با سیستم فایل در ارتباط هستند. قبل از اجرای هر عملیات، سیستم فایل باید فایل موردنظر را شناسایی و پیدا کند. این کار مستلزم استفاده از دایرکتوری است که محل فایل‌ها و صفات آنها را توصیف می‌کند. علاوه بر این، اغلب سیستم‌های اشتراکی، کنترل دستیابی کاربر را اعمال می‌کنند به‌طوری که فقط کاربران مجاز می‌توانند به روش خاصی به فایل ویژه‌ای دستیابی داشته باشند. عملیات اصلی که کاربر یا برنامه کاربردی می‌تواند در فایل انجام دهد، در سطح رکورد انجام می‌شوند. ازنظر کاربر یا برنامه کاربردی، فایل ساختاری دارد که رکوردها را سازمان‌دهی می‌کند، مثل ساختار ترتیبی که در آن رکوردهای پرسنلی به ترتیب حروف الفبای نام ذخیره می‌شوند. لذا، برای ترجمه فرمان‌های کاربر به فرمان‌های ویژه دستکاری فایل، روش دستیابی متناسب با ساختار فایل باید انتخاب شود.



برعکس کاربران و برنامه‌های کاربردی که با رکوردها سروکار دارند، I/O براساس بلوک انجام می‌گیرد. لذا، رکوردهای فایل برای خروجی باید به صورت بلوک درآیند و برای ورودی باید از حالت بلوک خارج شوند. برای پشتیبانی I/O بلوکی فایل، چند عمل باید انجام گیرد. حافظه ثانویه باید مدیریت شود. این کار شامل تخصیص فایل‌ها به بلوک‌های آزاد روی حافظه ثانویه و مدیریت بر حافظه آزاد است،به‌طوری که مشخص باشد چه بلوک‌هایی برای فایل‌های جدید وجود دارند و رشد فایل موجود چگونه است. علاوه بر این، هر درخواست I/O بلوک، باید زمانبندی شود. زمانبندی دیسک و تخصیص فایل با بهینه‌سازی کارآیی سروکار دارد. همانطور که انتظار می‌رود، این اعمال باید باهم در نظر گرفته شوند.

دایرکتوری‌های فایل

به همراه هر سیستم مدیریت فایل و مجموعه‌ای از فایل‌ها، یک دایرکتوری فایل وجود دارد. دایرکتوری فایل حاوی اطلاعاتی راجع به فایل‌ها از جمله صفات، محل و مالکیت است. اغلب این اطلاعات بخصوص اطلاعات مربوط به حافظه، توسط سیستم عامل مدیریت می‌شود. دایرکتوری خودش یک فایل است که در مالکیت سیستم عامل قرار دارد و روال‌های مختلف مدیریت فایل می‌توانند به آن دستیابی داشته باشند. اگرچه بعضی از اطلاعات موجود در دایرکتوری در اختیار کاربران و برنامه‌های کاربردی است،

ولی این اطلاعات بطور غیرمستقیم توسط روال‌های سیستم فراهم می‌شوند. لذا، کاربران حتی در حالت فقط خواندنی نیز نمی‌توانند مستقیماً به دایرکتوری دستیابی داشته باشند.

جدول زیر اطلاعاتی را که معمولاً برای هر فایل در دایرکتوری ذخیره می‌شوند، نشان می‌دهد. ازدیدگاه کاربر، دایرکتوری، نگاشتی را بین اسامی فایل که کاربران و برنامه‌های کاربردی آن‌ها را می‌شناسند و خود فایل ایجاد می‌کند. لذا، هر وارده فایل شامل نام فایل است. تقریباً تمام سیستم‌ها با انواع مختلفی از فایل‌ها و سازمان‌های فایل سروکار دارند و این اطلاعات نیز فراهم می‌شوند. یک بخش مهم از اطلاعات مربوط به فایل‌ها، در مورد حافظه آنها از جمله محل و اندازه است. در سیستم‌های اشتراکی، تهیه اطلاعاتی برای کنترل دستیابی به فایل اهمیت دارد. سرانجام، اطلاعات استفاده از فایل، برای مدیریت استفاده فعلی ئ ثبت سابقه استفاده از آن مفید است.

عناصر اطلاعاتی در دایرکتوری فایل	
اطلاعات اصلی	
نام فایل	نامی که توسط ایجاد کننده (یا برنامه کاربر) انتخاب می‌شود باید در دایرکتوری، منحصربفرد باشد.
نوع فایل	مانند فایل متنی، دودویی و غیره
سازمان فایل	برای سیستم‌هایی که سازمان‌های مختلفی را پشتیبانی می‌کنند.
اطلاعات آدرس	
جلد (Volume)	دستگاهی را مشخص می‌کند که فایل بر روی آن ذخیره شده است.
آدرس شروع	شروع آدرس فیزیکی در حافظه جانبی (سیلندر، شیار، شماره بلوک روی دیسک)
اندازه فعلی	اندازه فعلی فایل برحسب بایت، کلمه یا بلاک
اندازه تخصیص یافته	حداکثر اندازه فایل
اطلاعات کنترل دستیابی	
مالک	کاربری که کنترل فایل را برعهده دارد. مالک ممکن است مجوزها را به کاربران دیگر بدهد یا لغو کند و این امتیازات را تغییر دهد.
اطلاعات دستیابی	نسخه ساده‌ای از این عنصر شامل نام کاربر و کلمه رمز هرکاربر مجاز است.
فعالیت‌های مجاز	خواندن، نوشتن، اجرا و انتقال در شبکه را کنترل می‌کند.
اطلاعات استفاده از فایل	
تاریخ ایجاد	فایل در چه زمانی در دایرکتوری قرار داده شده.
هویت ایجادکننده	معمولاً ولی نه الزاماً مالک فعلی است.
تاریخ آخرین دستیابی خواندن	تاریخ آخرین زمانی که رکوردی خوانده شده
هویت آخرین خواننده	کاربری که فایل را خوانده است
تاریخ آخرین تغییر	تاریخ آخرین به‌هنگام‌سازی شامل درج، حذف یا به‌روز رسانی رکوردها
هویت آخرین تغییر دهنده	کاربری که فایل را تغییر داده است
تاریخ آخرین پشتیبان	تاریخ آخرین زمانی است که از فایل بر روی رسانه دیگر پشتیبان تهیه شده است.
استفاده فعلی	اطلاعات مربوط به فعالیت فعلی در فایل، مثل فرآیندهایی که توسط فایل باز شدند، اینکه توسط فرآیند قفل شده است و اینکه فایل در حافظه اصلی تغییر یافته ولی هنوز نوشته نشده است.

ساختار دایرکتوری

روش ذخیره اطلاعات جدول مبحث قبلی در سیستم‌های مختلف فرق می‌کند. بعضی از اطلاعات ممکن است در رکورد سرآیند (Header record) فایل ذخیره شوند. بدین ترتیب، میزان حافظه موردنیاز برای دایرکتوری کاهش می‌یابد و برای بهبود سرعت می‌توان تمام دایرکتوری‌ها را در حافظه اصلی نگهداری کرد. بعضی از عناصر کلیدی نظیر نام فایل، آدرس فایل و اندازه فایل در حافظه اصلی نگهداری می‌شود.

این ساختار ساده‌ی دایرکتوری، لیستی از وارده‌ها است که هر وارده مربوط به یک فایل است. این ساختار را می‌توان با فایل ترتیبی ساده نمایش داد به‌طوری که نام فایل به عنوان کلید عمل می‌کنند. در بعضی از سیستم‌های تک کاربره اولیه، از این تکنیک استفاده شده است. اما، وقتی چندین کاربر از سیستمی بطور اشتراکی استفاده می‌کنند، یا هریک از کاربران فایل‌های زیادی داشته باشند، مفید نیست.

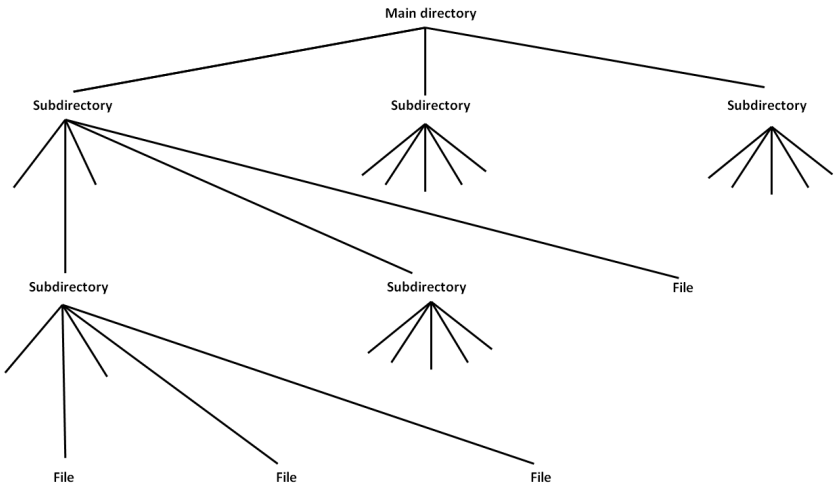
برای درک خواسته‌های ساختار فایل، بررسی انواع عملیاتی که ممکن است در دایرکتوری انجام شود مفید است

- **جست‌وجو** : وقتی کاربر یا برنامه کاربردی به فایل مراجعه می‌کند، دایرکتوری باید برای یافتن وارده متناظر با آن، فایل جستجو شود.
- **ایجاد فایل** : وقتی فایل جدیدی ایجاد می‌شود، وارده‌ای باید در دایرکتوری قرار گیرد.
- **حذف فایل** : وقتی فایلی حذف می‌شود، وارده‌ای باید از دایرکتوری حذف گردد.
- **لیست دایرکتوری** : تمام یا بخش از دایرکتوری ممکن است درخواست شود. معمولاً این درخواست توسط کاربر انجام می‌گیرد و لیستی از فایل‌های تحت مالکیت آن کاربر به همراه بعضی صفات (مانند نوع، اطلاعات کنترل دستیابی، اطلاعات استفاده از فایل) نمایش دشده می‌شوند.
- **به‌هنگام‌سازی دایرکتوری** : چون بعضی از صفات فایل در دایرکتوری ذخیره می‌شود، تغییر در یکی از این صفات، نیاز به تغییر در وارده متناظر در دایرکتوری دارد.

لیست ساده برای پشتیبانی از این عملیات مناسب نیست. نیازهای یک کاربر را درنظر بگیرید. کاربر ممکن است فایل‌هایی از انواع مختلف داشته باشد، مثل فایلهای متنی واژه‌پرداز، فایلهای گرافیکی، صفحه گسترده‌ها و غیره. کاربر ممکن است بخواهد آنها را بر حسب پروژه، نوع یا راه‌های دیگری سازمان‌دهی کند. اگر دایرکتوری یک لیست ترتیبی ساده باشد، به سازماندهی فایلها کمکی نمی‌کند و کاربر ملزم به عدم استفاده از فایل‌های همنام از یک نوع است. مسئله در سیستم‌های اشتراکی بدتر است و نام‌گذاری منحصربفرد جدی‌تر می‌شود. علاوه بر این، وقتی ساختار ارثی در دایرکتوری وجود نداشته باشد، پنهان کردن بخشی از دایرکتوری از کاربران دشوارتر خواهد بود.

شروع حل این مسئله را با طرح دوسطحی آغاز می‌کنیم. در این حالت یک دایرکتوری اصلی و برای هر کاربر نیز یک دایرکتوری وجود دارد. برای هر دایرکتوری کاربر، وارده‌ای در دایرکتوری اصلی وجود داد که آدرس و اطلاعات کنترل دستیابی را ارائه می‌کند. هر دایرکتوری کاربر، لیست ساده‌ای از فایل‌های آن کاربر است. معنای این ساختار این است که اسامی فقط باید در داخل مجموعه‌ای از فایل‌های یک کاربر، محنصربفرد باشد و سیستم فایل به آسانی می‌تواند محدودیتهای دستیابی را به دایرکتوری‌ها اعمال کند. اما در سازماندهی مجموعه‌ای از فایل‌ها به کاربر کمکی نمی‌کند.

یک روش قدرتمندتر و باقابلیت انعطاف بیشتر که تقریباً توسط همه مورد قبول است، روش سلسه مراتبی یا ساختار درختی است (نمونه‌ای از آن را در شکل زیر مشاهده می‌کنید.)



همانند قبل، یک دایرکتوری اصلی وجود دارد که چند دایرکتوری کاربر در زیر آن واقع هستند. هریک از دایرکتوری‌های کاربر نیز ممکن است حاوی دایرکتوری فرعی (Sub Directory) باشد که فایل‌هایی در داخل هریک از آنها موجود باشد. این موضوع درهر سطحی صادق است. یعنی در هر سطح دایرکتوری ممکن است حاوی وارده‌هایی برای فایل یا دایرکتوری دیگری باشد.

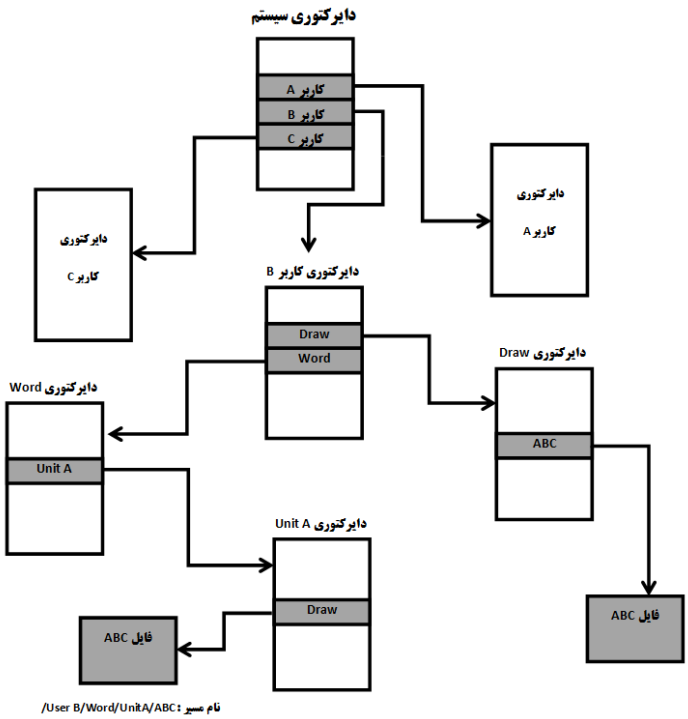
اکنون به چگونگی سازماندهی دایرکتوری‌ها می‌پردازیم البته ساده‌ترین روش این است که هر دایرکتوری بصورت یک فایل ترتیبی ذخیره شود. اگر تعداد وارده‌های دایرکتوری خیلی زیاد باشد زمان جستجو زیاد می‌شود. در این صورت باید از روش درهم‌سازی استفاده شود.

نام‌گذاری

کاربران باید بتوانند از طریق نام نمادین به فایل مراجعه کنند. بدیهی است که هر فایل موجود در سیستم باید نام منحصربفردی داشته باشد تا در مراجعه به آن ابهامی وجود نداشته باشد. از طرف دیگر نمی‌توان کاربران را وادار به انتخاب نام منحصربفرد برای فایل، بخصوص در سیستم اشتراکی کرد.

استفاده از دایرکتوری با ساختار درختی؛ نام‌گذاری منحصربفرد فایل‌ها را ساده می‌کند. هر فایل در سیستم می‌تواند با دنبال کردن مسیری از دایرکتوری اصلی(ریشه) به شاخه‌های مختلف مشخص شود. مجموعه‌ای از نام دایرکتوری‌ها، از جمله نام آن فایل، **نام مسیر** را برای آن فایل تشکیل می‌دهد.

به عنوان مثال، مسیر مربوط به قایل موجود در گوشه چپ در شکل زیر برابر User B/Word/UnitA/ABC/ است. از نماد / برای جدا کردن نامها استفاده می‌شود. نام دایرکتوری اصلی به‌طور ضمنی اعلان می‌شود، زیر تمام مسیرها از این دایرکتوری شروع می‌شوند. توجه کنید که چند فایل همنام می‌توانند در مسیرهای مختلفی وجود داشته باشند. لذا فایل دیگری به‌نام ABC در سیستم و در مسیر User B/Draw/ABC/ نیز وجود دارد.



گرچه نام مسیر، انتخاب اسامی فایل‌ها را تسهیل می‌کند، ولی عیبش این است که کاربر در هر با مراجعه به فایل باید نام مسیر را مشخص کند. معمولاً هرکاربر یا فرآیند دارای دایرکتوری فعلی است که **دایرکتوری کاری** نامیده می‌شود. در نتیجه فایلها نسبت به دایرکتوری کاری مراجعه می‌شوند. به عنوان مثال اگر Word دایرکتوری کاری مربوط به کاربر B باشد، آنگاه نام مسیر Unit A/ABC برای مشخص کردن فایل موجود در گوشه سمت چپ شکل بالا کافی است. وقتی کابر محاوره‌ای وارد سیستم می‌شود یا فرآیندی ایجاد می‌گردد، بطور پیش فرض دایرکتوری کاری، دایرکتوری کاربر است. در حین کار کاربر می‌تواند در درخت بالا یا پایین برود و دایرکتوری کاری جدیدی را ایجاد کند.

اشتراک فایل

در سیستم چندکاربره، تقریباً همیشه لازم است فایل‌ها بین تعدادی از کاربران مشترک باشند، در این صورت دو موضوع حقوق دستیابی و مدیریت دستیابی همزمان مطرح می‌شود.

حقوق دستیابی

سیستم فایل باید ابزار انعطاف‌پذیری برای اشتراک فایل‌ها بین کاربران فراهم کند. سیستم فایل باید گزینه‌هایی را فراهم کند تا روش دستیابی به فایل تحت کنترل باشد. معمولاً به کاربران یا گروهی از کاربران، حقوق دستیابی خاصی به فایل اعطا می‌شود. این حقوق می‌تواند سلسله مراتبی را ایجاد کند، بطوری که هر حقی شامل حقوق قبل از آن نیز هست. لذا اگر حق به هنگام‌سازی فایلی به کاربر داده شود، آنگاه حقوق آگاهی، اجرا، خواندن و افزودن نیز به کاربر داده خواهد شد.

معمولاً کسی که فایل را ایجاد کرده است مالک فایل است. مالک تمام حقوق دستیابی را دارا است و می‌تواند به دیگران نیز واگذار کند.

دستیابی همزمان

وقتی حق دستیابی به‌هنگام‌سازی و افزودن به فایل، به بیش ازیک کاربر داده شد، سیستم عامل یا سیستم مدیریت فایل باید نظمی را اعمال کند. در یک روش عادی، کاربر مجاز است کل فایل را در زمان به‌هنگام‌سازی، قفل کند. کنترل عالی‌تر این است که رکوردی که به‌هنگام‌سازی می‌شود قفل گردد. اساس این مسئله، مسئله خوانندگان و نویسندگان است. موضوعات انحصار متقابل و بن‌بست باید در طراحی قابلیت دستیابی اشتراکی درنظر گرفته شوند.

مدیریت حافظه ثانویه

فایل در حافظه ثانویه از چندین بلوک تشکیل شده است، سیستم عامل یا مدیریت فایل مسئول تخصیص بلوکها به فایل است. دو موضوع مدیریتی مطرح می‌شود. اولاً فضای حافظه ثانویه باید به فایل‌ها تخصیص یابد و ثانیاً ردیابی فضای موجود برای تخصیص ضروری است. خواهیم دید که این دو وظیفه به‌هم مرتبط هستند یعنی روش تخصیص فایل ممکن است روش مدیریت فضای آزاد را تحت تأثیر قرار دهد. علاوه براین خواهیم دید که تعاملی بین ساختار فایل و سیاست تخصیص وجود دارد.

تخصیص فایل

موضوعات زیادی در مورد تخصیص فایل وجود دارد:

۱. وقتی فایل جدیدی ایجاد می‌شود، آیا حداکثر فضای موردنیاز فایل تخصیص می‌یابد؟
۲. فضا یک یا چند واحد پیوسته به فایل تخصیص می‌یابد که به آنها بخش گفته می‌شود. اندازه هر بخش می‌تواند از یک بلوک تا کل فایل باشد. برای تخصیص فایل، اندازه بخش چقدر باید باشد؟
۳. برای نگهداری بخش‌های تخصیص یافته به فایل از چه ساختمان داده یا جدولی استفاده می‌شود؟

چنین جدولی را **جدول تخصیص فایل (FAT)** می‌نامند.

پیش تخصیص در مقابل تخصیص پویا

در سیاست پیش تخصیص، لازم است حداکثر اندازه فایل در زمان ایجاد فایل اعلان شود. در مواردی مثل ترجمه برنامه، تهیه خلاصه‌ای از فایل داده، یا انتقال فایل از سیستم دیگر از طریق شبکه، مقدار حافظه مورد نیاز را بخوبی می‌توان برآورد کرد. اما برای بسیاری از برنامه‌های کاربردی برآورد دقیق اندازه فایل اگر غیرممکن نباشد، دشوار است. در این مورد، کاربران و برنامه‌نویسان کاربردی سعی می‌کنند فضای بیشتری را برآورد کنند تا دچار کمبود حافظه نشوند. این موضوع از دیدگاه تخصیص حافظه ثانویه، هدر دادن حافظه است. لذا بهتر است از تخصیص پویا استفاده شود که درصورت نیاز بخش‌هایی را به فایل تخصیص می‌دهد.

اندازه بخش

مسئله دوم در تخصیص فایل، اندازه بخش تخصیص یافته به فایل است. یک حالت این است که بخش بتواند کل فایل را دربرگیرد و حالت دیگر این است که هربار یک بلوک از فضای دیسک تخصیص یابد، در انتخاب اندازه بخش، بین کارآیی از دیدگاه یک فایل و کارآیی کل سیستم، توازنی برقرار است. در این موازنه باید چهار عنصر زیر درنظر گفته شود:

۱. پیوستگی فضا مخصوصاً برای عملیات بازیابی بعدی و انجام تراکنش در سیستم عامل تراکنشی موجب افزایش کارآیی می‌شود.
۲. تعداد زیادی از بخشهای کوچک، اندازه جدول‌های موردنیاز برای مدیریت اطلاعات تخصیص را دشوار می‌کند.

۳. بخش‌هایی با اندازه ثابت (مثل بلوک‌ها) تخصیص مجدد فضا را تسهیل می‌کند.
۴. بخش‌هایی بااندازه متغیر یا کوچک و با اندازه ثابت، اتلاف حافظه در اثر تخصیص بیش از حد را به حداقل می‌رساند

البته این موارد باهم تعامل دارند و باید باهم درنظر گرفته شوند. نتیجه‌اش دو گزینه عمده زیر است:

- **بخش‌های پیوسته بزرگ و متغیر** : این گزینه کارآیی بهتری فراهم می‌کند. اندازه متغیر از اتلاف حافظه‌ها جلوگیری می‌کند و جدول‌های تخصیص فایل کوچکتر هستند. اما استفاده مجدد از فضا دشوار است.
- **بلوک‌ها** : بخش‌های کوچه با اندازه ثابت، قابلیت انعطاف زیادی دارند. ممکن است برای تخصیص آنها نیازی به جدول‌های بزرگ یا ساختارهای پیچیده‌ای باشد. پیوستگی مطرح نیست و بلوک‌ها در صورت نیاز تخصیص می‌یابند.

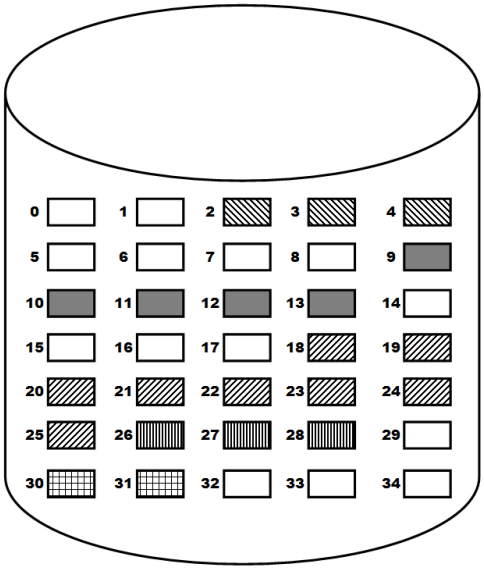
هر دو گزینه با تخصیص پویا و پیش تخصیص سازگارند. در مورد بخش‌های پیوسته بزرگ و متغیر، گروه پیوسته‌ای از بلوک‌ها به فایل تخصیص می‌یابد. دراین صورت نیازی به جدول تخصیص فایل نیست بلکه فقط به اشاره‌گر اولین بلوک و تعداد بلوک‌های تخصیص یافته نیاز است. در مورد گزینه دوم (بلوک‌ها)، تمام بخش‌های موردنیاز همزمان تخصیص می‌یابند، بدین معنا که اندازه جدول تخصیص فایل ثابت باقی می‌ماند.
در بخش‌هایی با طول ثابت، با تکه‌تکه‌شدن فضای آزاد مواجه‌ایم. بعضی از راهبردهای ممکن عبارتند از:

- **اولین برازش (اولین جای مناسب)** : اولین گروه از بلوک‌های استفاده نشده و به اندازه کافی را از لیست بلوک‌های آزاد انتخاب می‌کند.
- **بهترین برازش (بهترین جای مناسب)** : کوچک‌ترین گروه از بلوک‌های استفاده نشده و با اندازه کافی را انتخاب می‌کند.
- **نزدیک‌ترین برازش** : گروه استفاده نشده و با اندازه مناسب از بلوک‌هایی را انتخاب می‌کند که به آخرین تخصیص قبلی فایل نزدیک است. به این ترتیب احتمال محلی بودن را افزایش می‌یابد.

روش‌های تخصیص فایل

پس از بررسی تخصیص پویا در مقابل پیش تخصیص و اندازه بخش، می‌توانیم روش‌های تخصیص فایل را بررسی کنیم. سه روش پیوسته، زنجیره‌ای و شاخص‌دار متداول هستند که در زیر به معرفی و شرح آنها خواهیم پرداخت.

در **تخصیص پیوسته**، هنگام ایجاد فایل مجموعه‌ای پیوسته‌ای از بلوک‌های تخصیص می‌یابد. یعنی از راهبرد پیش تخصیص با بخش‌هایی با اندازه متغیر استفاده می‌شود (شکل زیر). جدول تخصیص فایل برای هر فایل یک وارده دارد که شروع بلاک و طول فایل را مشخص می‌کند. از دیدگاه ترتیبی، تخصیص پیوسته بهترین راهبرد است.

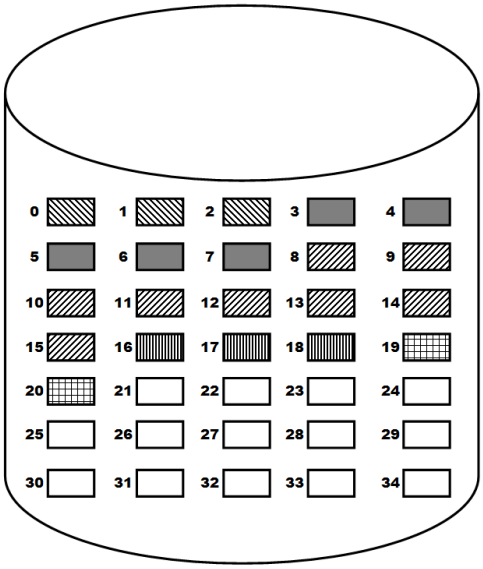


تخصیص فایل به‌صورت پیوسته

جدول تخصیص فایل

نام فایل	بلوک شروع	طول
فایل A	2	3
فایل B	9	5
فایل C	18	8
فایل D	30	2
فایل E	26	3

برای بهبود کارآیی I/O در پردازش ترتیبی، چندین بلوک را می‌توان بصورت همزمان به حافظه آورد. بازیابی یک بلوک نیز آسان است. به عنوان مثال اگر فایلی از بلوک b شروع شودو بلوک iام فایل درخواست گردد، محل آن در حافظه ثانویه برابر b+i-1 است. تخصیص پیوسته مشکلاتی نیز دارد. تکه‌تکه‌شدن خارجی بوجود می‌آید که یافتن بلوک‌های پیوسته‌ای از فضای آزاد و با اندازه کافی را دشوار می‌کند. هر از گاهی نیاز به اجرای الگوریتم فشرده‌سازی است تا فضای بیشتری از دیسک آزاد شود (شکل زیر). حتی در پیش تخصیص نیاز به اعلان اندازه فایل در هنگام ایجاد آن است که مسائل خاص خودش را دارد.



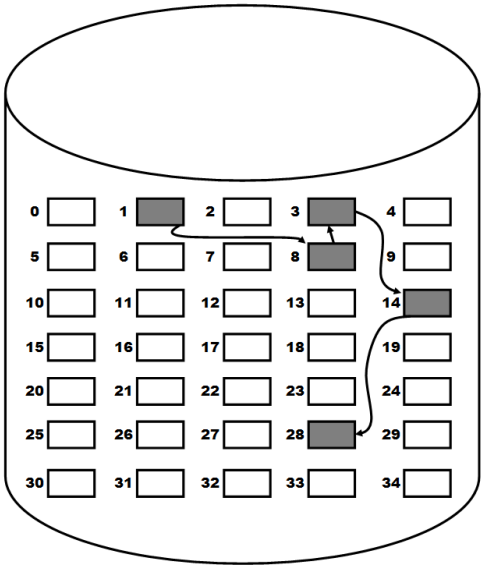
تخصیص فایل به‌صورت پیوسته

(پس از فشرده‌سازی)

جدول تخصیص فایل

نام فایل	بلوک شروع	طول
فایل A	0	3
فایل B	3	5
فایل C	8	8
فایل D	19	2
فایل E	16	3

در مقابل تخصیص پیوسته، **تخصیص زنجیره‌ای** قرار دارد (شکل زیر) معمولاً تخصیص بر اساس بلوک انجام می‌شود. هر بلوک حاوی اشاره‌گری به بلوک بعدی موجود در زنجیره است. در اینجا نیز جدول تخصیص فایل به ازای هر فایل یک وارده دارد که بلوک شروع و طول فایل را نشان می‌دهد. گرچه پیش‌تخصیص امکان‌پذیر است ولی تخصیص بلوک موردنیاز متداول است. انتخاب بلوک‌ها نیز چندان مهم نیست. هر بلوک آزاد را می‌توان به زنجیره اضافه کرد. چون در هر زمان یک بلوک نیاز است در مورد تکه-تکه‌شدن خارجی نگرانی وجود ندارد. این نوع سازمان فیزیکی برای پردازش ترتیبی فایل مناسب‌ترین است. برای انتخاب بلوک خاصی از فایل، ردیابی زنجیره تا بلوک موردنظر ضروری است.

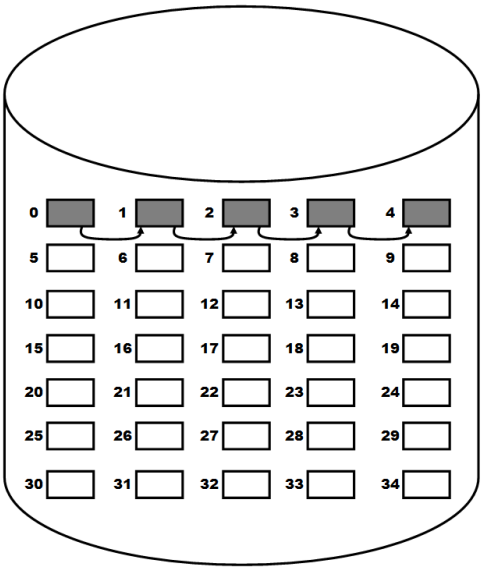


تخصیص فایل به‌صورت زنجیره‌ای

جدول تخصیص فایل

نام فایل	بلوک شروع	طول
...	...	...
فایل B	1	5
...	...	...

یک پیامد زنجیره‌ای که تاکنون توصیف شده است این است که اصل محلی بودن جایی ندارد. لذا اگر همانند پردازش ترتیبی لازم باشد چندین بلوک به حافظه بیایند نیاز به مجموعه‌ای از دستیابی‌ها به نقاط مختلف دیسک است. احتمالاً این مهمترین تأثیر در سیستم تک کاربره است ولی ممکن است در سیستم اشتراکی نیز مورد توجه باشد. برای غلبه بر این مسئله بعضی از سیستم عامل‌ها فایل‌ها را بصورت متناوب یکپارچه می‌کنند (شکل زیر).

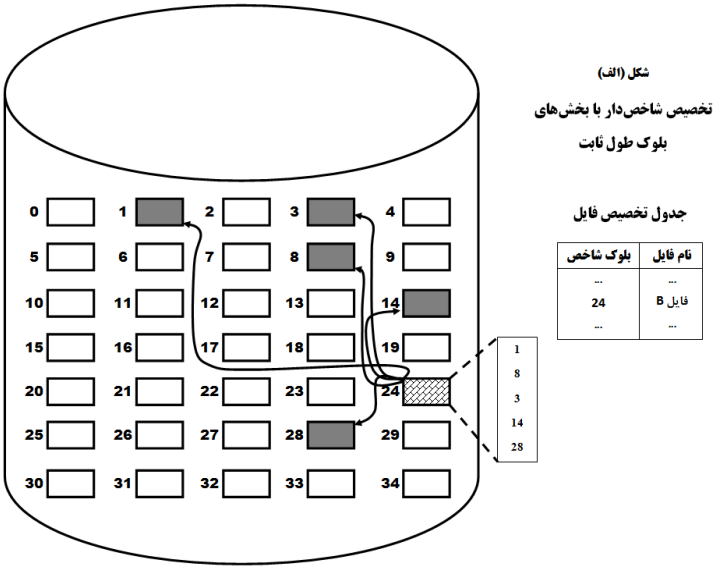


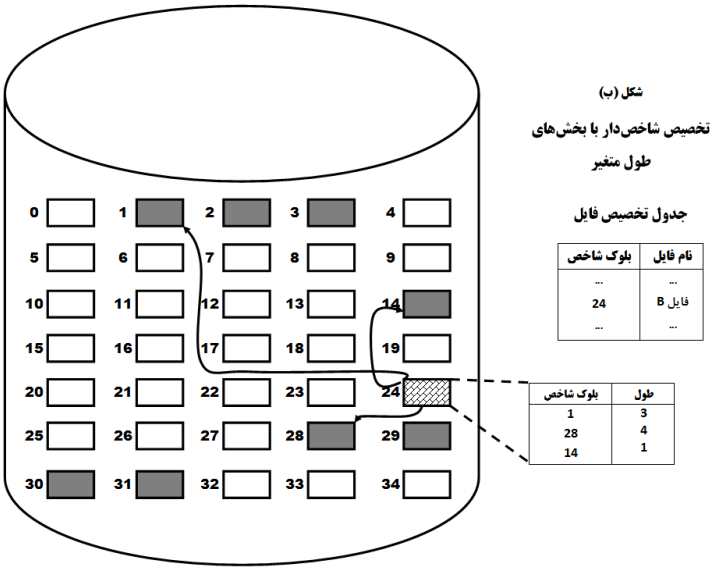
تخصیص فایل به‌صورت زنجیره‌ای

جدول تخصیص فایل

نام فایل	بلوک شروع	طول
...	...	...
فایل B	0	5
...	...	...

تخصیص شاخص‌دار بسیاری از مسائل تخصیص پیوسته و زنجیره‌ای از حل می‌کند. در این روش، جدول ایجاد فایل برای هر فایل، شاخص یک سطحی جداگانه‌ای دارد. این شاخص به‌ازای هر بخش تخصیص یافته به فایل یک وارده دارد. معمولاً شاخص‌های فایل بطور فیزیکی به عنوان بخش از جدول تخصیص فایل ذخیره نمی‌شود. در عوض شاخص فایل در بلوک جداگانه‌ای نگهداری می‌شود و وارده فایل در جدول تخصیص فایل، به آن بلوک اشاره می‌کند. تخصیص ممکن است بر اساس بلوک‌های با طول ثابت (شکل الف زیر) یا بخش‌های طول متغیر (شکل ب زیر) باشد.





تخصیص بلوک‌ها، تکه‌تکه‌شدن خارجی را از بین می‌برد درحالیکه تخصیص بخش‌های با طول متغیر، محلی بودن را بهبود می‌بخشد. در هرحال یکپارچه‌سازی فایل هرازگاهی انجام می‌شود. یکپارچه‌سازی فایل در مورد بخش‌های طول متغیر از اندازه شاخص

می‌کاهد اما در مورد تخصیص بلوک اینطور نیست. تخصیص شاخص‌دار از دستیابی ترتیبی و مستقیم به فایل پشتیبانی می‌کند و درنتیجه معروفترین شکل تخصیص است.

مدیریت فضاهای آزاد دیسک

فضای آزاد نیز همانند فضای تخصیص یافته به فایل باید مدیریت شود. برای اجرای هریک از تکنیک‌های تخصیص فایل (که بطور کامل تشریح شدند) دانستن بلوک‌های موجود روی دیسک ضروری است. لذا علاوه بر جدول تخصیص فایل به **جدول تخصیص دیسک** نیز نیاز داریم. بعضی از تکنیک‌های پیاده‌سازی شده را در زیر بررسی می‌کنیم.

۱- **جدول‌های بیتی :** این روش از برداری استفاده می‌کند که به‌ازای هر بلوک یک بیت دارد. مقدار "صفر" نشان دهنده بلوک آزاد و مقدار "یک" نشان دهنده بلوک استفاده شده است.به عنوان مثال برای دیسک شکل مربوط به تخصیص فایل به صورت پیوسته به برداری با طول ۳۵ نیاز است که مقدار آن بردار برابر است با :

00111000011111000011111111111011000

مزیت جدول بیتی آن است که یافتن یک یا گروه پیوسته‌ای از بلوک‌ها آسان است. لذت جدول بیتی برای هر روش تخصیص فایل مناسب است. مزیت دیگرش کوچک بودنش است. میزان حافظه مورد نیاز (برحسب بایت) برای بلوک بیتی به‌صورت زیر محاسبه می‌شود

$$\frac{\text{اندازه دیسک بر حسب بایت}}{\text{اندازه بلوک سیستم فایل} \times ۸}$$

لذا برای یک دیسک ۱۶ گیگابایتی با بلوک‌های ۵۱۲ بیتی جدول تقریبا ۴ مگابایت را اشغال می‌کند.

۲- **بخش‌های آزاد زنجیره‌ای :** بخش‌های آزاد ممکن است با استفاده از اشاره‌گر و مقدار طول در هر بخش، با هم زنجیره شوند. این روش سربار فضای کوچکی دارد زیرا نیاز به جدول تخصیص دیسک نیست و فقط نیاز به اشاره‌گر ابتدای زنجیر و طول اولین بخش است. این روش برای تمام روش‌های تخصیص فایل مناسب است. اگر هربار یک بلوک تخصیص یابد، بلوک آزاد ابتدای زنجیر انتخاب می‌شود و اولین اشاره‌گر با مقدار طول تنظیم می‌گردد. اگر تخصیص با بخش متغیر انجام شود، ممکن است از الگوریتم اولین برازش استفاده شود : هر با سرآیندهای (Header) هر بخش واکشی می‌شوند تا بخش آزاد مناسب بعدی در زنجیر پیدا شود، سپس اشاره‌گر و مقادیر طول تنظیم می‌شوند.

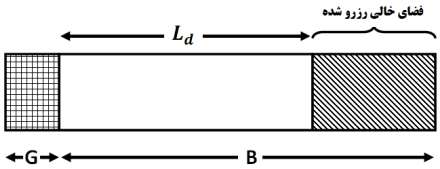
این روش مشکلات خاص خودش را دارد. پس از مدتی استفاده، دیسک کاملاً تکه‌تکه می‌شود و طول بسیاری از بخش‌ها به اندازه یک بلوک می‌شود. همچنین توجه کنید که هر وقت بلوکی تخصیص می‌یابد قبل از نوشتن داده‌ها در آن بلوک، ابتدا باید بلوک خوانده شود تا اشاره‌گر به اولین بلوک آزاد جدید، تعیین شود. اگر قرار باشد برای عملیات فایلی، چند بلوک به‌طور همزمان تخصیص یابند ایجاد فایل کند می‌شود. به‌طور مشابه، حذف کردن فایلی که خیلی تکه‌تکه شده است نیز وقت زیادی می‌گیرد.

۳- **شاخص‌بندی :** روش شاخص‌بندی، با فضای خالی مثل فایل رفتار می‌کند و همانطور که در تخصیص فایل گفته شد، از جدول شاخص استفاده می‌شود. برای کارآیی، شاخص باید بر اساس بخش‌هایی با طول متغیر باشدو نه بلوک‌ها. لذا برای هر بخش آزاد روی دیسک، یک وارده در جدول وجود دارد. این راهبرد از تمام روش‌های تخصیص فایل پشتیبانی می‌کند.

در ادامه بعد از بررسی روش‌های و تکنیک‌های ذخیره‌سازی فایل در دیسک به سایر موارد مربوط به سیستم فایل همانند لوکالیتی، آدرس‌دهی و بافرینگ خواهیم پرداخت.

چگالی لود اولیه (Loading Density) :

درصورتیکه بتوانیم حجم عملیات ذخیره‌سازی بر روی فایل را از قبل تخمین بزنیم، می‌توانیم تمام فضای یک بلاک را در لود اولیه برای ذخیره‌سازی پر نکرد و مقداری از فضا در هر بلاک داشته باشیم تا از این حافظه بعداً در عملیات ذخیره‌سازی استفاده شود.



چگالی لود اولیه توسط درصدی از اندازه بلاک مشخص می‌شود که از رابطه زیر بدست می‌آید

$$\text{چگالی لود اولیه} = \frac{L_d}{B} < ۱$$

مزایای درنظر گرفتن حافظه رزرو در بلاک:

۱- این امر موجب افزایش لوکالیتی فایل می‌شود چراکه از پراکندگی نشست رکوردها روی دیسک تا حدودی جلوگیری می‌کند. این موضوع میزان پراکندگی در زمان دست‌یابی تصادفی و نیز زمان پردازش ترتیبی فایل را بهبود می‌بخشد

۲- موجب تسهیل در انجام برخی از عملیات بر روی فایل خواهد شد. مثلاً در هنگام درج رکوردی که طول آن در اثر عملیات بهنگام‌سازی افزایش یافته دیگر نیاز به حذف منطقی از بلاک فعلی و درج در بلاک دیگری نخواهد بود و می‌توان با یک شیفت درون بلاکی، رکورد تغییریافته را در جای قبلی خود بازنویسی کرد.

معایب درنظر گرفتن حافظه رزرو در بلاک:

۱- این حافظه، در واقع نوعی حافظه هرز محسوب می‌شود که موجب افزایش اندازه و حجم فایل خواهد شد و درنتیجه زمان خواندن کل فایل افزایش می‌یابد.

۲- درصورتیکه توزیع رکوردها یکنواخت نباشد، موجب باقی ماندن حافظه هرز در انتهای بعضی از بلاک‌ها خواهد شد.

لوکالیتی یا میزان همسایگی رکوردها در فایل (Locality) :

رکورد منطقی بعدی، رکوردی است که در هنگام تعریف فایل باید بعد از رکورد فعلی پردازش شود و در واقع رکورد منطقاً همجوار رکورد فعلی است ولی می‌دانیم که در هنگام ذخیره‌سازی این امر اتفاق نمی‌افتد و رکوردهای منطقی پشت‌سرهم ذخیره می‌شوند.

تعریف : میزان همسایگی (نزدیکی) فیزیکی رکورد منطقی بعدی به رکود فعلی را لوکالیتی می‌گویند.

هرچه لوکالیتی رکوردها قوی‌تر باشد زمان پردازش سریال آنها کمتر خواهد بود. چراکه دیگر در هنگام خواندن، زمان کمتری برای انتقال هد دیسک به جلو و عقب برای خواندن رکورد بعدی صرف خواهد شد.

نکته : در هنگام کار سیستم فایل، لوکالیتی کاهش می‌یابد که باید فایل را سازماندهی مجدد گردد.

سطوح آدرس دهی به فایل:

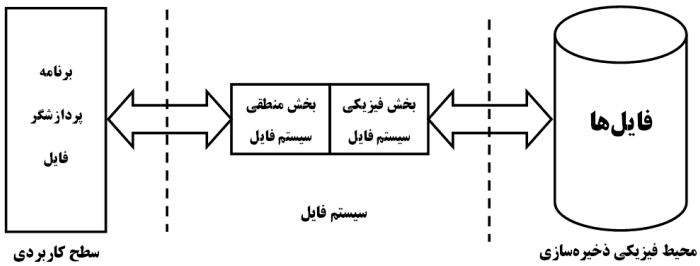
سطوح آدرس‌دهی (Addressing Level) :

فایل را می توان از دو دیدگاه مورد بررسی قرار داد :

۱- **فایل منطقی** : فایلی که نشان دهنده ساختاری که رکوردهای فایل بر اساس آن مورد دستیابی و پردازش قرار می گیرد.

۲- **فایل فیزیکی** : نشان دهنده نحوه ذخیره رکوردها برروی دیسک می باشد.

سیستم فایل : بخشی از سیستم عامل است که با فایل ها سروکار دارد از دو بخش تشکیل شده است.



۱) **بخش منطقی سیستم فایل** که وظیفه انجام در خواست های کاربر نظیر باز کردن فایل و...را دارد.

۲) **بخش فیزیکی سیستم فایل** که وظیفه دارد که درخواست های دریافتی از بخش منطقی را جهت ارسال به دیسک تبدیل به دستورات مناسبی نماید.

فرمان هایی که از طرف بخش فیزیکی سیستم فایل بررسی دیسک ارسال میشود متفاوت از فرمان های بخش منطقی می باشد ولی به طور کلی سه عمل اصلی در محیط فیزیکی انجام می شود که عبارتند از :

۱. مکان یابی(seek)
۲. خواندن از دیسک(read)
۳. نوشتن در دیسک(write)

که تمامی این عملیات باید به صورت فیزیکی انجام شود.

سطوح مختلف آدرس دهی فایل:

الف- در سطح برنامه پردازش گر فایل

ب- در سطح سیستم فایل منطقی

ج- در سطح سیستم فیزیکی

الف- آدرس‌دهی در بخش برنامه پردازشگر فایل:

معمولا در این سطح به یکی از سه روش زیر آدرس دهی انجام می شود.

۱) نسبی
۲) محتوایی
۳) نمادی

در آدرس‌دهی نسبی کاربر فایل، فایل را به صورت یک ساختار خطی مشاهده می کند که درآن هر رکورد دارای شماره دارد، با شروع از صفر برای اولین رکورد فایل. در این روش کاربر آدرس رکورد RRA (Relative Record Address) را به عنوان آرگومان جستو می دهد.

در آدرس‌دهی محتوایی کاربر یک مقدار صفت خاصه را به عنوان پارامتر جستجو می کند .

در آدرس‌دهی نمادی کاربر رکورد مورد نظرش را به کمک یک اسم سمبلیک مشخص می کند.

ب- آدرس‌دهی درسطح منطقی سیستم فایل:

در این سطح از آدرس‌دهی نسبی، استفاده می کنیم ولی این آدرس نسبی با آدرس نسبی از دید کاربر متفاوت است.در واقع این آدرس نسبی در سطح کل فضای ذخیره‌سازی محاسبه می شود. بخش منطقی سیستم فایل کل فضای ذخیره‌سازی را به شکل آرایه‌ای از بلاک‌ها می‌بیند آدرس یا شماره بلاک را با RBA (Relative Block Address) نشان می‌دهند که آدرس نسبی بلاک اول برابر صفر در نظر گرفته می شود.

برای بدست آوردن آدرس نسبی حاوی رکورد موردنظر، ابتدا باید آدرس نسبی بلاکی را که رکورد در آن قرار دارد به‌دست آوریم

$$\text{آدرس نسبی بلاک حاوی رکورد i ام (نسبت به بلاک اول فایل)} = \left[\frac{(\mathbf{i}-\mathbf{1})\times \boldsymbol{R}}{\mathbf{B}}\right]$$

(که این آدرس نسبت به شروع بلاک‌های می‌باشد)

حال برای بدست آوردن آدرس نسبی بلاک رکورد مورد نظر باید این آدرس نسبی را با RBA بلاک اول جمع می کنیم.

$$\boldsymbol{RBA_R} = \boldsymbol{RBA_B} + \left[\frac{(\mathbf{i}-\mathbf{1})\times \boldsymbol{R}}{\mathbf{B}}\right]$$

$\boldsymbol{RBA_R}$: آدرس نسبی حاوی رکورد موردنظر

$\boldsymbol{RBA_B}$: آدرس نسبی بلاک اول فایل

مثال: فرض کنید که می خواهیم رکورد ۲۲ را از یک فایل بخوانیم با فرض اینکه تعداد کل رکوردهای فایل برابر ۵۰ رکورد و هر رکورد ۱۰۰بایت ، فضا را جهت ذخیره سازی لازم داشته باشد . اگر طول هر بلاک ۳۰۰ بایت و آدرس اولین بلاک بر روی دیسک را برابر ۸ در نظر بگیریم ، RBA بلاک حاوی رکورد مورد نظر نسبت به اولین بلاک و نسبت به اول دیسک چقدر می باشد ؟

ج- آدرس دهی در سطح فیزیکی سیستم فایل :

در این سطح باید آدرس فیزیکی یعنی آدرسی که داده ی مورد نظر بر روی دیسک ذخیره شده است مشخص گردد که بخش فیزیکی سیستم فایل جهت این کار ابتدا آدرس را از بخش منطقی دریافت و سپس تبدیل به آدرس فیزیکی برای محیط ذخیره سازی می نماید .

بافر (Buffer) :

ناحیه ای است از حافظه ی اصلی که جهت هماهنگی در عملیات ورودی و خروجی و عملیاتCPU بکار می رود . بافرناحیه ای است که حداقل یک بلاک را بتواند در خود جای دهد که در هر بار عملیات خواندن و نوشتن از دیسک حداقل یک بلاک در بافر گذاشته و یا خوانده می شود . در صورتی که فایل را بلاک بندی کرده باشیم معمولاً برای انجام عملیات ورودی و خروجی بر روی فایل ها از دو بافر ورودی و خروجی استفاده می شود .

بافرها به سه روش ایجاد می شوند :

۱) توسط برنامه نویس با ایجاد ناحیه ای از حافظه در برنامه
۲) توسط سیستم عامل در هنگام باز شدن فایل و یا بستن آن
۳) اجرای یک ماکرو و درخواست ایجاد بافر از طرف سیستم عامل

روشهای دسترسی سیستم فایل به محتوای بافر :

۱- روشهای انتقالی (Move Mode)
۲- روشهای مکان نمائی (Locate Mode)

روش انتقالی : در این روش رکورد از بافر ورودی به ناحیه کاری که همان ناحیه برنامه می‌باشد منتقل شده و برعکس از ناحیه کاری برنامه به بافر منتقل می گردد . در این صورت همانگونه که مشاهده می‌کنید تنها بافری که برنامه به آن دسترسی دارد همان بافر مربوط به خود می‌باشد و ناحیه جداگانه‌ای از حافظه را جهت بافر ورودی/خروجی اختصاص نداده‌ایم.

روش مکان‌نمائی : در این روش برنامه به طور مستقیم عملیات نقل و انتقال داده‌ها را بر روی بافر ورودی/خروجی انجام می‌دهد و در واقع کاربر از همان بافر به عنوان ناحیه کاری استفاده می کند .

انواع بافر از نظر محل ایجاد :

۱) **بافر نرم افزاری :** ناحیه‌ای است از حافظه اصلی که توسط سیستم عامل در اختیار برنامه‌ها قرار می‌گیرد .
۲) **بافر سخت افزاری :** بافر موجود در دستگاههایی مانند کارت خوان، چاپگر و .. استفاده می‌شود.این بافرها معمولاً حجم زیادی از اطلاعات را می‌توانند ذخیره کنند این نوع بافر با سرعت دستگاه ذخیره‌سازی پر شده و پس از آن که بافر تکمیل شد محتوای آن را با سرعت انتقال کانال به کامپیوتر و از آنجا به بافر نرم‌افزاری منتقل می‌شود.

انواع بافرینگ:

از نظر تعداد بافرهایی که به عملیات برنامه سیستم فایل تخصیص داده می شود عبارتند از:

الف) بافرینگ استاندارد (Single buffering)

ب) بافرینگ مضاعف (Double buffering)

ج) بافرینگ چند گانه (Multiple buffering)

الف) بافرینگ استاندارد: در این روش یک بافر در اختیار برنامه پردازش‌گر فایل قرار داده می شود.هنگامی که بافر در حال پر شدن است CPU نمی‌تواند از محتویات بافر استفاده کند پس باید منتظر بماند تا کار فعلی بافر تمام شده که این امر موجب پایین آمدن کارایی و تعلیق برنامه‌ها خواهد شد .در محیط‌های چند برنامگی می‌توان از این زمان انتظار برای ایجاد دیگر برنامه‌ها استفاده کرد.

ب) بافرینگ مضاعف: در هنگامی که یک بلاک خوانده می شود و به یک بافر منتقل می شود می‌توانید به طور همزمان بافری دیگر را پردازش کنیم. در پردازش فایل‌ها به صورت انبوه و پی‌درپی که تمام بلاک‌ها خوانده می‌شوند، لازم است که حتماً دو بافر داشته باشیم، چرا که در غیر این صورت سرعت و کارایی عملیات پایین خواهد آمد.

با توجه به این مطلب که در هنگام پر شدن یک بافر می توانیم بافر دیگر را مورد استفاده قرار دهیم.به این نتیجه می رسیم که باید زمانی که CPU صرف پردازش یک بافر می کند از زمان پر شدن بافر دیگر کمتر باشد.

$$C_B < b_{tt} \qquad C_B \leq \frac{B+G}{t} \qquad C_B \leq \frac{R+W_R}{t}$$

در فرمول های فوق:

C_B : زمان لازم جهت پردازش محتوای بلاک در بافر

b_{tt} (Block Transfer Time) : زمان انتقال یک بلاک به حافظه بر حسب ثانیه برابر است با

t : نرخ انتقال بر حسب بایت بر ثانیه

B : طول بلاک بر حسب بایت

G : طول گپ بر حسب بایت

W_R : حافظه هرز به ازای هر رکورد

R : طول رکورد بر حسب بایت

نکته : درصورتیکه شرط $C_B < b_{tt}$ برقرار نباشد با فرینگ مضاعف کارایی خود را از دست می‌دهد و نرخ انتقال کاهش پیدا می‌کند.

ج) بافرینگ چند گانه : در این روش با تخصیص چند بافر به برنامه پردازش‌گر در هنگام خواندن فایل‌ها این امکان را که رکوردهای فایل از قبل خوانده شده و در بافر قرار گیرند وجود داشته و بنابراین در هر لحظه رکوردهای بعدی در بافر است البته در پردازش فایل‌ها با ساختار مستقیم و تصادفی این روش کارایی چندانی ندارد چرا که از قبل نمی‌دانیم که رکورد قبلی در کجا قرار دارد.

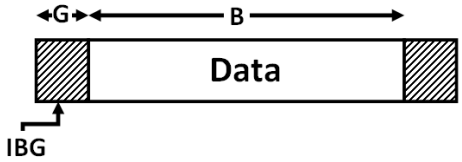
فصل چهارم :

ارزیابی پارامتریک رسانه‌های ذخیره‌سازی (دیسک ، نوار)

در این فصل دو ویژگی مهم وسایل ذخیره‌سازی را ارزیابی می‌کنیم . ظرفیت واقعی ذخیره‌سازی و نرخ یا سرعت واقعی انتقال که این دو پارامتر را در مورد نوار مغناطیسی و دیسک مغناطیسی بررسی خواهیم کرد.

ظرفیت واقعی نوار:

ظرفیت واقعی نوار را با داشتن ظرفیت اسمی آن می‌توان بررسی کرد. عاملی که باعث می‌شود از ظرفیت واقعی نوار به طور کامل استفاده نکنیم، وجود Gap می‌باشد.



باتوجه به شکل فوق به ازای هر B بایت (B+G) بایت از نوار جهت ذخیره‌سازی استفاده می‌کنیم. بنابرین درصد استفاده واقعی از نوار برابر خواهد بود با :

$$\text{درصد استفاده واقعی از نوار} = \frac{B}{B+G} \times 100$$

$$\text{ظرفیت اسمی نوار} = L \times D$$

که در فرمول فوق L طول نوار بر حسب اینچ و D چگلی نوار بر حسب بایت در اینچ می باشد.پس میزان استفاده واقعی از نوار برابر خواهد بود با :

$$\text{درصد استفاده واقعی از نوار} = \frac{B}{B+G} \times L \times D$$

همانطور که می‌بینید طول گپ و طول بلاک در میزان واقعی استفاده از نوار تاثیر گذار هستند.

اندازه بلاک:

اندازه یک بلاک با توجه به نوع دستگاه ذخیره‌سازی و نوع سیستم عامل تعیین می شود.در صورتی که اندازه یک بلاک را کوچک در نظر بگیریم باعث می‌شود میزان بلاک‌های استفاده شده برای ذخیره‌سازی افزایش یافته در نتیجه به موازات آن تعداد گپ‌ها نیز افزایش می‌یابد و فضای هرز بیشتری در سیستم تولید می‌شود.

مثال : فایلی با 10^5 رکورد که هر رکورد 100 بایت می باشد را بر روی نواری به چگالی 1000 بایت در اینچ می خواهیم ذخیره سازی نماییم.در صورتی که طول گپ را 0.04 اینچ فرض کنیم با فرض $B_f=1$ و $B_f=4$ موارد زیر را حساب کنید؟

الف) درصد استفاده واقعی از نوار

ب) طول واقعی بر روی دیسک

نرخ انتقال واقعی نوار :

برای خواندن نوار به طور کلی دو روش وجو دارد.

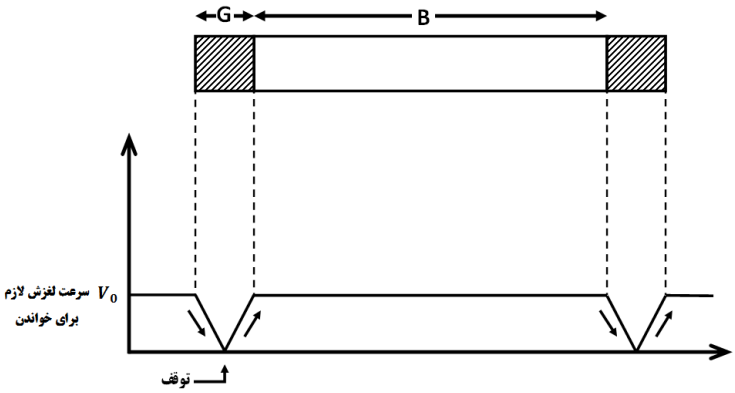
۱) شیوه بلاکی(Block mode) : در این روش در هر بار فقط یک بلاک خوانده یا نوشته می‌شود.

۲) شیوه جریانی (Stream mode): در این روش در هر بار N بلاک خوانده و نوشته می‌شود.

در واقع مشکل تفاوت بین نرخ انتقال واقعی و نرخ انتقال رسمی به دو دلیل عامل کاهنده زیر است.

۱- زمان حرکت توقف ۲- زمان طی کردن گپ

در صورتی که نوار را به صورت بلوکی بخوانیم نوک خواندن و نوشتن فقط یک بلاک را می‌خواند و می‌ایستد و در صورتی که از روش جریانی استفاده کنیم این توقف بعد از خواندن N بلاک انجام می‌گیرد. در روش جریانی گپ‌های بین بلاک با همان سرعتی که بلاکها خوانده می‌شوند طی خواهند شد ولی در روش بلوکی این زمان برابر زمان حرکت توقف می باشد.



زمان خواندن یک بلاک b_{tt} بدون درنظر گرفتن گپ برابر است با $b_{tt} = \frac{B}{t}$ که t نرخ انتقال اسمی بر حسب بایت در ثانیه می‌باشد .

زمان خواندن یک بلاک (بدون درنظر گرفتن عوامل کاهنده) $\frac{B}{t}$ است. اما واقعاً زمان سپری شده برابر است با : $\frac{B}{t} + \frac{G}{t} + \frac{\tau}{1000}$

τ : زمان حرکت-توقف بر حسب میلی ثانیه می‌باشد. در اینصورت تقسیم بر ۱۰۰۰ می‌شود تا برحسب ثانیه بدست آید.

$$\text{نرخ انتقال واقعی در روش بلوکی} = \frac{\frac{B}{t}}{\frac{B}{t}+\frac{G}{t}+\frac{\tau}{1000}} \times t$$

درصورتیکه τ را برحسب ثانیه درنظر بگیریم خواهیم داشت :

ولی در روش جریانی، هر بار N بلاک بدون توقف از نوار خوانده می‌شود .بنابراین خواهیم داشت.

$$\text{نرخ انتقال واقعی در روش جریانی} = \frac{N\times\frac{B}{t}}{N\times\frac{B}{t}+N\times\frac{G}{t}+\frac{\tau}{1000}}\times t$$

درصورتیکه τ را برحسب ثانیه درنظر بگیریم خواهیم داشت :

مثال۱ : در یک نوار با طول بلاک ۵۰۰ بایت و طول گپ ۱۵۰ بایت، زمان لازم برای حرکت توقف ۲۰ میلی ثانیه می‌باشد . اگر نرخ انتقال اسمی۱۸۰۰ بایت بر ثانیه باشد، نرخ انتقال واقعی این رسانه چقدر خواهد بود؟(فرض کنید از شیوه بلاکی برای خواندن بلاک ها استفاده شده است.)

مثال۲ : برای خواندن یک فایل با روش جریانی به صورت ۸ بلاکی با فرض آنکه طول هر بلاک ۳۲۰ بایت و طول هر گپ۸۰ بایت در نظر گرفته شود، با در اختیار داشتن نرخ انتقال اسمی نوار برابر ۹۵۰ بایت بر ثانیه و سرعت حرکت توقف هد برابر ۲۰ میلی ثانیه باشد . نرخ انتقال واقعی چقدر خواهد بود؟

مثال۳ : برای خواندن یک فایل با بلاک‌های به طول ۸۰۰ بایت و طول گپ ۱۵۰ بایت و نرخ انتقال اسمی ۶۰۰۰ بایت بر ثانیه و حرکت/توقف برابر با ۵۳۰ میلی ثانیه ، مطلوبست نرخ انتقال واقعی در هر یک از حالات زیر:
الف- استفاده از شیوه بلاکی
ب-استفاده از شیوه جریانی با N=5

مثال۴ : در یک نوار مغناطیسی با طول بلاک وگپ به ترتیب برابر ۲۵۰ و ۶۰ بایت زمان لازم برای حرکت یا توقف مجدد سوزن ۲۵ میلی ثانیه می باشد .اگر برای انتقال یک بلاک به بافر نیازمند زمانی برابر ۸۰ ثانیه باشیم، زمان واقعی بلاک چقدر خواهد بود؟

مثال۵ : اگر در یک نوار مغناطیسی نرخ انتقال واقعی برابر 0.25 نرخ انتقال اسمی آن باشد و شیوه جریانی برای دست یابی به بلاک های نوار با N=4 استفاده شود، طول هر گپ 25 و طول هر بلاک 150 بایت و ح برابر 50 میلی ثانیه . مطلوبست یافتن نرخ انتقال اسمی در این نوار؟

ظرفیت واقعی نوار یا دیسک:

نحوه فرمت‌بندی شیار در ارزیابی ظرفیت واقعی دیسک تاثیر به سزایی دارد.برای ارزیابی ظرفیت واقعی باید تمام حافظه های هرز (گپ) را با توجه به نحوه فرمت‌بندی شیار و تکنیک‌های بلاک‌بندی در سطح یک شیار از ظرفیت اسمی، کم نمود. اندازه یک بلاک از جمله پارامترهای مهم در میزان استفاده واقعی از دیسک می‌باشد . هر چه طول بلاک کمتر در نظر گرفته شود، تعداد بلاک‌ها افزایش پیدا می‌کند و در نتیجه تعداد گپ‌های ایجاد شده در یک شیار افزایش خواهد یافت و از طرف دیگر اگر طول بلاک را بزرگ در نظر بگیریم باعث مشکل پارگی داخلی و همچنین باعث بروز حافظه هرز ناشی از نگنجیدن آخرین بلاک در شیار می‌شود.

برای محاسبه میزان واقعی استفاده از حافظه دیسک‌ها از رابطه‌های زیر استفاده می کنیم.

$$N = \left\lceil \frac{B}{L_s} \right\rceil \qquad E = \frac{R \times B_f}{L_s \times N} \times 100$$

R : طول رکورد

L_s : طول سکتور

N : تعداد سکتورها در بلوک

E : درصد استفاده واقعی

که در رابطه بالا $R \times B_f$ میزان واقعی ذخیره‌سازی و $L_s \times N$ برابر میزان فضایی است که برای ذخیره‌سازی در نظرگرفته می‌شود.

مثال : فرض کنید در یک سیستم فایل هر رکورد دارای طولی برابر 600 بایت بوده.طول هر سکتور برابر 512 بایت. در صورتی که B_fبرابر 6 در نظر گرفته شود . درصد استفاده واقعی از این نوار را محاسبه نمایید.

نکته : یکی از عوامل موثر در کاهش فضای هرز (Gap) در دیسک‌ها طول بلاک می باشد که بهتر است ضریب صحیحی از اندازه سکتور باشد. چرا که همانطور که می‌دانیم این امکان که فضای هرز در انتهای شیار (به علت نگنجیدن بلاک) بوجود آید، زیاد می-باشد.

تعیین طول بلاک در تعیین طول بلاک باید به محدودیت ها و فاکتورهای زیر توجه داشته باشیم :

۱- نحوه پردازش فایل : اگر فایل به صورت تصادفی پردازش شود، به‌علت اینکه برای پردازش هر رکورد باید کل بلاک آن خوانده شود،در نتیجه طولانی بودن زمان خواندن بلاک، زمان کل پردازش را افزایش خواهد داد.

۲- نوع سیستم عامل : در محیطهایی که سیستم عامل آنها از تکنیک حافظه مجازی استفاده می کنند .اندازه بلاک محدود به میزان حافظه‌ای است که سیستم برای هر بار خواندن تعریف می نماید.(اندازه صفحه)

۳ - در محیط‌هایی که چندین برنامه از یک فایل استفاده می‌نماید اندازه بلاک باید با توجه به اندازه کوچکترین بافر تعیین گردد.

۴ – امکانات بافرینگی (buffering) که سیستم در اختیار ما می گذارد.

نرخ انتقال واقعی در دیسک:

عوامل موثر در نرخ انتقال واقعی عبارتند از:

۱- سرعت انتقال بلاک به حافظه $\boldsymbol{b_{tt}}$

۲- زمان استوانه جویی $\boldsymbol{S}$

۳- زمان درنگ دورانی $\boldsymbol{r}$

برای محاسبه نرخ انتقال واقعی خواهیم داشت :

($S + r + b_{tt}$ برابر زمان خواندن یک بلاک می‌باشد)

$$\text{نرخ انتقال واقعی} = \frac{\boldsymbol{B}}{\boldsymbol{S + r + b_{tt}}}$$

البته دقت داشته باشید که در اینجا فرض شده است که در هر بار نقطه یک بلاک از هر کجای دیسک و به صورت تصادفی پردازش می کنیم.

بنابراین برای خواندن N بلاک باید برای هر بلاک زمان $S + r + b_{tt}$ صرف کرد بنابراین خواهیم داشت.

$$\text{زمان (بلاک‌ها به صورت تصادفی خوانده می شوند.)} = \boldsymbol{N \times (S + r + b_{tt})}$$

ولی چنانچه بخواهیم بلاک ها را به صورت انبوه و پشت سر هم پردازش کنیم .زمان خواندن N بلاک کمتر از حالت قبل خواهد شد.چرا که در این حالت $S + r$ فقط یکبار و آن هم در مر حله مکان یابی آغاز بلاک ها مورد استفاده قرار می گیرد .

$$\text{زمان انتقال N بلاک به صورت انبوه (پی‌درپی)} = \boldsymbol{S + r + N \times b_{tt}}$$

فصل پنجم :

بهبود کارآیی سیستم‌فایل

در بعضی از کاربردها، لازم است که فایل به صورت ترتیبی و یا پی‌درپی خوانده شود. داشتن مدیریت بافرینگ کارا و امکانات تخصیص بافر، نقش مهمی در بهبود کارآیی پردازش تمام فایل، به ویژه در حالت پی‌درپی ایفا می‌کند. علاوه برآن تکنیک‌های متعددی، چه نرم‌افزاری و چه سخت افزاری وجود دارند که منجر به بهبود کارآیی دیسک می‌شوند. این تکنیک‌ها زمان پیگرد (S) و درنگ دورانی (r) را کاهش می‌دهند و یا سرعت پردازش فایل را افزایش می‌دهند.

از بین تکنیک‌های متعددی که وجود دارد موارد زیر را تشریح خواهیم کرد.

- استفاده از بافر برای افزایش سرعت پردازش فایل‌ها. (درفصلهای قبلی بررسی شده است)
- سازماندهی داده‌ها با استفاده از سیلندرها (استوانه‌ها) جهت کاهش زمان پیگرد و درنگ دورانی
- استفاده از الگوریتم‌های مناسب جهت حرکت هِد خواندن/نوشتن برای ترتیب انتخاب درخواستها و کاهش زمان پیگرد
- توزیع داده‌ها بین چند دیسک کوچکتر به جای یک دیسک بزرگ. در این روش با تعداد هِدهای بیشتر برای خواندن، به تعداد بیشتری از بلوک‌ها در واحد زمان دستیابی خواهیم داشت و زمان پیگرد کاهش می‌یابد (استفاده از سیستم RAID)
- استفاده از حافظه نهان دیسک
- استفاده از تکنیک درهم یا تداخل بلاک‌ها (Interleaving) جهت کاهش درنگ دورانی
- تغییر نقطه آغاز شیارها (Track Stagging) جهت کاهش درنگ دورانی

الف) سازماندهی داده‌ها با استفاده از سیلندرها (استوانه‌ها)

چون زمان پیگرد تقریباً نصف میانگین زمان دستیابی به بلوک‌ها را شامل می‌شود، در بسیاری از کاربردها بهتر است داده‌هایی که با احتمال زیاد با هم دستیابی می‌شوند، در یک سیلندر قرار داده شوند. اگر فضای کافی وجود نداشته باشد (حجم بلوک‌ها بیش از ظرفیت یک سیلندر باشد)، می‌توان از چند سیلندر همجوار استفاده کرد.

در واقع اگر تمام بلوک‌های موجود در یک شیار یا یک سیلندر را بطور ترتیبی بخوانیم، می‌توانیم از تمام زمانهای پیگرد و درنگ دورانی (به جز زمان اولین پیگرد برای انتقال به سیلندر و اولین درنگ دورانی جهت انتقال بر روی اولین بلوک در سیلندر) صرف نظر کرد.

این روش در مواردی که بلوک‌های داده روی دیسک به ترتیب خوانده و نوشته می‌شوند به‌طوریکه از قبل قابل پیش‌بینی باشند و فقط یک فرآیند در هر زمان از دیسک استفاده کند، بسیار مفید است. اما در مواردی که چندین فرآیند به‌طور موازی اجرا می‌شوند و از یک دیسک به‌صورت اشتراکی استفاده می‌نمایند، مناسب نیست.

ب) استفاده از الگوریتم‌های مناسب جهت حرکت هِد خواندن/نوشتن

همانگونه که قبل گفته شد و مشاهده کردید؛ می‌توان با تقویت لوکالیتی زمان خواندن فایل را در روش انبوه کاهش داد و با استفاده از بافرینگ مضاعف زمان پردازش فایل‌ها را کم نمود ولی در یک محیط چند برنامه ای (اجرای چندین برنامه به طور همزمان بر روی CPU) سیستم عامل باید به حرکت برروی دیسک چند برنامه پاسخ دهد. بنابراین امکان بی‌نظمی در اثر حرکت زیاد هِد وجود دارد که این حرکت باعث افزایش زمان استوانه‌جویی می شود. بنابراین بهتر است در چنین سیستم‌هایی از الگوریتم‌های مناسب برای پاسخگویی به درخواست‌های برنامه‌ها برای دسترسی به دیسک استفاده گردد.

۱) الگوریتم FCFS یا FIFO (First In First Out – First Come First Out) :

ساده‌ترین شکل زمانبندی دیسک، سرویس به ترتیب ورود (FCFS) یا خروج به ترتیب ورود (FIFO) است که عناصر صف را بترتیب ورود به صف پردازش می‌کند. امتیاز این روش عادلانه بودن آن است زیرا درخواست‌ها به‌ترتیب واردشدن به صف پردازش می‌شوند. در این روش اگر فقط چند فرآیند نیاز به دستیابی داشته باشند و سکتورهای مجاور درخواست شوند، می‌توانیم به کارآیی خوب روش امیدوار باشیم. اما اگر تعداد زیادی از فرآیندها برای دیسک رقابت کنند، این تکنیک از نظر کارآیی چندان خوبی ندارد.

در این روش درخواست‌ها به‌ترتیب ورود اجرا می‌شوند.

مثال : بافرض اینکه هِد خواندن/نوشتن بر روی شیار 80 قرار داشته باشد و درخواست‌های زیر را از چپ به راست برای دسترسی به شیارها داشته باشیم، متوسط زمان حرکت به ازای هر درخواست در صورت استفاده از الگوریتم FCFS بدست آورید.

درخواست‌ها : 30 , 55 , 95 , 70 , 150 , 120 , 180 , 25

۲) الگوریتم SSTF (Shortest Seek Time First) :

منطقی است که درخواست‌های نزدیک به محل فعلی هِد، زودتر اجرا شوند. این ایده اصلی برای الگوریتم سرویس بر اساس کوتاهترین زمان پیگرد (SSTF) است. الگوریتم SSTF درخواستی را انتخاب می‌کند که نسبت به موقعیت فعلی هِد کمترین تفاضل زمان استوانه‌جویی (نزدیک‌ترین شیار) را دارد. چون زمان استوانه‌جویی با تعداد شیارهایی که توسط هِد پیموده می‌شود رابطه مستقیم دارد، بنابراین اگر بتوانیم هرچه مقدار این شیارهای پیموده شده را کمتر کنیم، موجب افزایش کارآیی دیسک خواهد شد. البته انتخاب کمترین تفاضل زمان پیگرد تضمین نمی‌کند که میانگین استوانه‌جویی حداقل باشد، اما نسبت به FCFS کارآیی بهتری دارد. البته اشکالی که در این روش است این می‌باشد اگر بطور مداوم درخواست‌هایی نزدیک محل فعلی هِد (دقت داشته باشید که صف درخواست‌ها پویا است) وارد صف شوند این امکان که درخواست‌های دورتر از محل فعلی دیرتر جواب داده شوند وجود دارد که به این وضعیت گرسنگی می‌گویند.

مثال : بافرض اینکه هِد خواندن/نوشتن بر روی شیار 80 قرار داشته باشد و درخواست‌های زیر را از چپ به راست برای دسترسی به شیارها داشته باشیم، متوسط زمان حرکت به ازای هر درخواست در صورت استفاده از الگوریتم SSTF بدست آورید.

درخواست‌ها : 30 , 55 , 95 , 70 , 150 , 120 , 180 , 25

۳) الگوریتم پیمایش (SCAN) :

دراین روش هد دیسک مرتب به طرف شیارهای ابتدایی و انتهایی در حال حرکت است و در این حرکت درخواست‌هایی که در سر راهش قرار دارند را پاسخ می‌دهد .دقت داشته باشید دراین روش پس از رسیدن به یک سمت دیسک (شیار ابتدایی وشیار انتهایی)جهت حرکت عوض می‌شود. الگوریتم پیمایش را گاهی الگوریتم آسانسور نیز می‌نامند زیرا هِد دیسک مانند آسانسور در یک ساختمان عمل می‌کند.

مثال : بافرض اینکه هِد خواندن/نوشتن بر روی شیار 80 قرار داشته باشد و درخواست‌های زیر را از چپ به راست برای دسترسی به شیارها داشته باشیم، متوسط زمان حرکت به ازای هر درخواست در صورت استفاده از الگوریتم SCAN بدست آورید. (جهت حرکت دیسک به سمت شیارهای بیرونی می‌باشد)

درخواست‌ها : 30 , 55 , 95 , 70 , 150 , 120 , 180 , 25

۴) الگوریتم پیمایش حلقوی (Circular Scan – C-SCAN) :

این الگوریتم شکل دیگری از الگوریتم پیمایش است که زمان انتظار را به‌طورت یکنواخت توزیع می‌کند. هِد دیسک در یک جهت حرکت می‌کند تا به انتهای دیسک برسد و در بین راه به درخواست‌ها پاسخ می‌دهد. وقتی به انتهای دیسک رسید، فوراً به ابتدای دیسک برمی‌گردد و حرکت را دوباره از ابتدا آغاز می‌کند (توجه کنید که در برگشت به هیچ درخواستی پاسخ نمی‌دهد و این زمان برگشت از شیار انتهایی به شیار ابتدایی را صفر درنظر می‌گیریم).

مثال : بافرض اینکه هِد خواندن/نوشتن بر روی شیار 80 قرار داشته باشد و درخواست‌های زیر را از چپ به راست برای دسترسی به شیارها داشته باشیم، متوسط زمان حرکت به ازای هر درخواست در صورت استفاده از الگوریتم C-SCAN بدست آورید.

درخواست‌ها : 30 , 55 , 95 , 70 , 150 , 120 , 180 , 25

۵) الگوریتم پیمایش LOOK :

در الگوریتم پیمایش و پیمایش حلقوی، دیدید که هِد دیسک در کل دیسک حرکت می‌کند. درعمل هیچ الگوریتمی بدین صورت پیاده‌سازی نمی‌شود بلکه، هِد دیسک حداکثر تا آخرین درخواست (موجود فعلی در صف درخواست‌ها) برای شیار در جهت حرکت پیش می‌رود و سپس فوراً برمی‌گردد (بدون آنکه به انتهای دیسک برسد).

۶) الگوریتم LIFO (Last In First Out) :

سیاست پاسخگویی به تازه‌ترین درخواست یا خروج به ترتیب عکس ورود (LIFO) امتیازاتی دارد. در سیستم‌های پردازش تراکنش، تخصیص دستگاه به کار تازه وارد، برای حرکت در فایل ترتیبی منجر به هیچ حرکتی یا حرکت اندک هِد دیسک می‌شود. مزیت این حرکت محلی، موجب بهبود توان عملیاتی و کاهش وطول صف‌ها می‌شود.

انتخاب الگوریتم زمان‌بندی مناسب

با توجه به وجود الگوریتم‌های متعدد برای زمان‌بندی حرکت هِد دیسک، کُدام یک را باید انتخاب کرد؟ SSTF متداول بوده و جذابیت آن طبیعی است. پیمایش و پیمایش حلقوی در سیستم‌هایی مناسب است که بار زیادی بر روی دیسک وجود دارد، زیرا احتمال اینکه منجر به گرسنگی شود، کمتر است. برای هر لیستی از درخواست‌ها، می‌توان ترتیب بهینه‌ای را برای بازیابی تعریف کرد اما محاسبات لازم برای یافتن زمان‌بندی بهینه ممکن است نسبت به استفاده از الگوریتم SSTF یا SCAN مقرون به‌صرفه نباشد.

در هر الگوریتم، کارآیی به تعداد و نوع درخواست‌ها موجود در صف بستگی دارد. به عنوان مثال، فرض کنید همیشه یک درخواست در صف وجود دارد، دراینصورت تمام الگوریتم‌ها مانند هم عمل می‌کنند زیرا یک انتخاب بیشتر ندارند و همه مانند FCFS عمل می‌کنند.

ج) استفاده از سیستم RAID

کوتاه شده عبارت Redundant Array of Inexpensive Disks می‌باشد و کار آن ایجاد یک واحد از مجموع چند هارد دیسک میباشد. یکی از موانع مهم در مقابل افزایش کارائی سیستم‌های رایانه‌ای محدودیت سرعت عملیات ورودی/خروجی یا I/O است .در حالی که سرعت ریز پردازنده‌ها و حجم حافظه با شتاب زیادی رو به افزایش است. در سال ۱۹۹۸ سه محقق از دانشگاه برکلی در مقاله‌ای ایده اولیه RAID را مطرح ساختند.RAID آرایه‌ای از دیسک‌های مستقل است که به طور موازی عملیات I/O را انجام داده و بدین ترتیب باعث بهبود کارائی در ورودی و خروجی سیستم می‌شوند. ایده اصلی RAID ترکیب چندین هارددیسک مستقل، ارزان و با ظرفیت پائین و استفاده از آنها به عنوان یک هارد دیسک بزرگ و سریع است

طرح RAID شامل هفت سطح است که از 0 تا 6 شماره‌گذاری می‌شود. این سطوح دارای معماری (طراحی) مختلفی دارند ولی سه ویژگی مشترک دارند که عبارتند از:

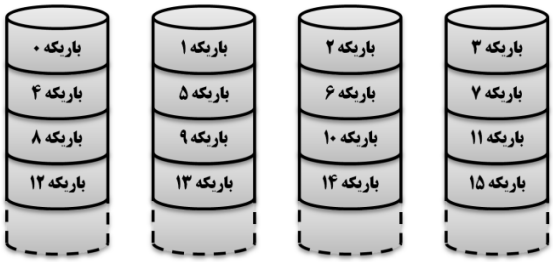
۱. RAID مجموعه‌ای از گرداننده‌های دیسک فیزیکی است. سیستم‌عامل آن را یک درایو منطقی می‌بینند.
۲. داده‌ها بر روی گرداننده‌های فیزیکی هر آرایه توزیع می‌شوند.
۳. ظرفیت افزونگی دیسک، برای ذخیره اطلاعات توازن (Parity) به‌کار می‌رود که قابلیت ترمیم داده‌ها در صورت خرابی دیسک را تضمین می‌کند.

جزئیات ویژگی‌های اول و دوم برای سطوح مختلف RAID فرق می‌کند. سطح صفر RAID ویژگی سوم را پشتیبانی نمی‌کند.

۱- RAID سطح صفر

RAID سطح صفر عنصر واقعی خانواده RAID نیست چراکه فاقد افزونگی برای بهبود کارآیی می‌باشد. برای RAID سطح صفر، داده‌های کاربر و سیستم در تمام دیسک‌های آرایه‌ای توزیع می‌شوند. این کار امتیاز قابل توجهی نسبت به دیسک یکپارچه دارد. اگر دو درخواست I/O مختلف برای دو بلوک محتلف داده‌ای وجود داشته باشد، احتمال اینکه بلوک‌ها درخواستی روی دیسک‌های مختلفی باشند، زیاد است. بنابراین دو درخواست را می‌توان به‌طور موازی صادر کرد تا زمان صف‌بندی I/O کاهش یابد.

اما RAID سطح صفر، همانند سایر سطوح RAID فقط توزیع داده‌ها بر روی آرایه دیسک‌ها نیست. داده‌ها بر روی چند دیسک باریکه‌سازی (Stripped) می‌شوند.

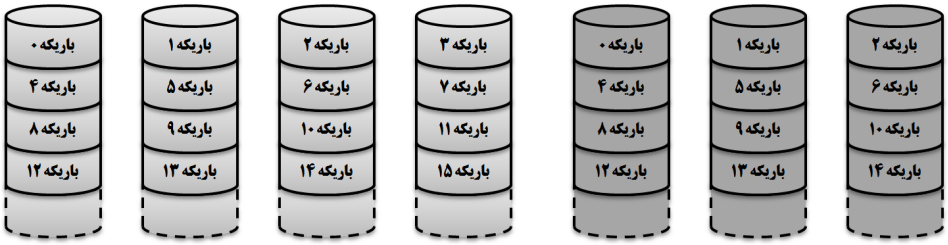


RAID سطح صفر (بدون افزونگی)

تمام داده‌های کاربر و سیستم بر روی دیسک منطقی خیره می‌شوند. دیسک به باریکه‌هایی تقسیم می‌شود. این باریکه‌ها ممکن است بلوک‌های فیزیکی، قطاع‌ها یا واحد دیگری باشند. باریکه‌ها به‌طور گردشی به عناصر متوالی حافظه نگاشت می‌شوند. مجموعه‌ای از باریکه‌ها منطقاً یک باریکه را به هر عنصر آرایه نگاشت می‌کند یک نوار (Strip) نامیده می‌شود. در آرایه n دیسکی، n باریکه منطقی اول (به‌طور فیزیکی) به عنوان اولین نوار هر n دیسک ذخیره می‌شود. n باریکه دوم به عنوان دومین نوار بر روی هر دیسک ذخیره می‌گردد و به همین ترتیب الی آخر.

۲- RAID سطح ۱

تفاوت RAID سطح ۱ با سطوح ۲ تا ۶ این است که افزونگی در آن ایجاد می‌شود. در سطوح ۲ تا ۶ نوعی محاسبات توازن برای معرفی افزونگی به‌کار می‌رود، در حالیکه در RAID سطح ۱ افزونگی از طریق تکرار داده‌ها ایجاد می‌شود. همانگونه که در شکل زیر مشاهده می‌کنید، همانند RAID سطح صفر، از باریکه‌سازی داده‌ها استفاده می‌شود. اما در RAID سطح ۱ هر باریکه منطقی، در دو دیسک جداگانه نوشته می‌شود بطوریکه هر دیسک موجود دارای یک دیسک دیگر و یکسان از نظر داده‌ای می‌باشد. RAID سطح ۱ را آینه‌کردن دیسک‌ها (Disk Mirroring) نیز می‌نامند.



RAID سطح ۱ (آینه بندی)

مزایایی این روش عبارت است از:

- درخواست خواندن را از بین دو دیسکی که حاوی داده درخواستی هستند، آن دیسکی پاسخ می‌دهد که حداقل زمان استوانه‌جویی و درنگ دورانی را برای درخواست موردنظر داشته باشد.
- درخواست نوشتن باید توسط هر دو دیسک پاسخ‌دهی شود ولی اینکار می‌تواند به‌صورت موازی انجام گیرد. لذا کارایی نوشتن توسط آن در این روش کندتر است.(زمان کل نوشتن را آن دیسکی مشخص می‌کند که دارای زمان استوانه‌جویی و درنگ دورانی بیشتری باشد)
- ترمیم خرابی در آن آسان است. وقتی گرداننده‌ای با شکست مواجه می‌شود، داده‌ها می‌توانند از گرداننده دیگر دستیابی شوندد.

عیب RAID سطح ۱ هزینه زیاد آن می‌باشد چراکه به دو برابر فضای دیسک منطقی جهت ذخیره‌سازی داده‌ها نیاز دارد.

۳- RAID سطح ۲

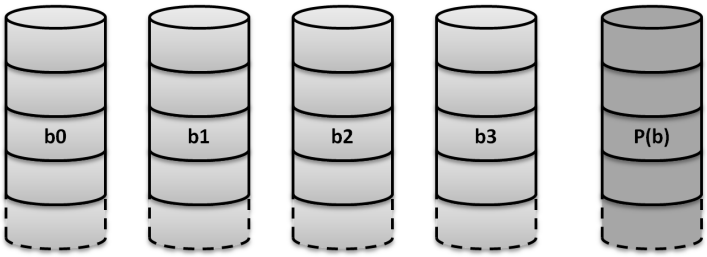
RAID سطح ۲ از تکنیک دستیابی موازی استفاده می‌کند. در این روش به‌جای افزونگی از کُدهای تصحیح خطا که بر روی بیت‌های متناظر در هر دیسک داده محاسبه می‌شود و بیت‌های این کُد در بیت‌های متناظر بر روی دیسک‌های توازن چندگانه ذخیره می‌شود. معمولاً از کُد همینگ استفاده می‌شود که قادر است خطاهای یک بیتی را تصحیح و خطاهای دو بیتی را تشخیص دهد، استفاده می‌شود. اگر چه RAID سطح ۲ نسبت به سطح ۱ به تعداد دیسک‌های کمتری نیاز دارد ولی هنوز هم گران است.(تعداد دیسک‌های افزونگی، متناسب با لگاریتم تعداد دیسک‌های داده‌ای است) در هنگام درخواست عمل خواندن، تمام دیسک‌ها همزمان دستیابی می‌شوند. داده‌ها درخواستی از دیسک‌های داده‌ای و کُدهای کنترل خطا نیز از دیسک‌های افزونگی خوانده می‌شوند واگر خطای یک بیتی وجود داشته باشد، تشخصیص داده می‌شود و تصحیح می‌شود.

RAID سطح ۲ (افزونگی از طریق کد همینگ)

RAID سطح ۲ در محیط‌هایی استفاده می‌شود که تعداد خطاهای دیسک زیاد باشد.

۴- RAID سطح ۳

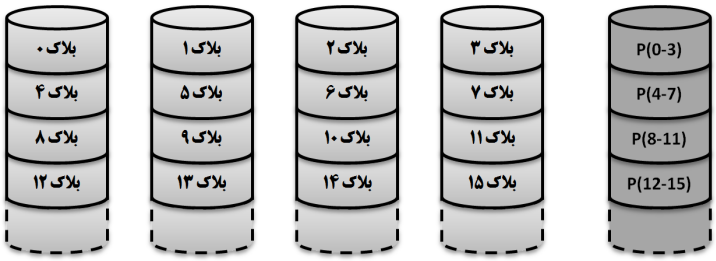
RAID سطح ۳ همانند سطح ۲ سازماندهی می‌شود با این تفاوت که RAID سطح ۳ بدون توجه به تعداد دیسک‌های داده‌ای فقط به یک دیسک اضافه نیاز دارد. RAID سطح ۳ به جای کُد تصحیح خطا از یک بیت توازن برای مجموعه بیت‌های موجود در مکان یکسان برروی دیسک‌های داده‌ای استفاده می‌شود.



RAID سطح ۳ (توازن بیتی)

۵- RAID سطح ۴

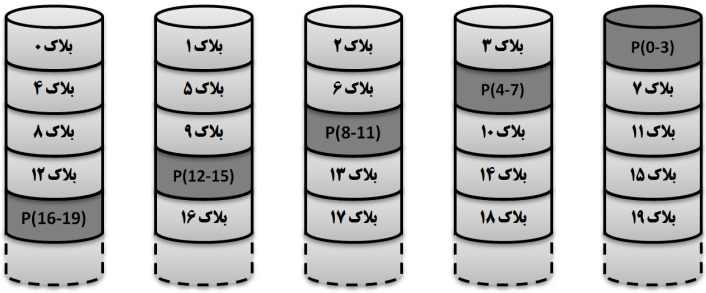
در RAID سطوح ۴ الی ۶ باریکه تقریباً بزرگ هستند. در RAID سطح ۴، یک باریکه توازن برای بیت‌به‌بیت باریکه‌های متناظر بر روی دیسک‌های داده‌ای، محاسبه می‌شود و بیت‌های توازن در باریکه متناظر روی دیسک توازن ذخیره می‌شوند.



RAID سطح ۴ (توازن سطح بلاک)

۶- RAID سطح ۵

RAID سطح ۵ همانند سطح ۴ سازماندهی می‌شود با این تفاوت که در سطح ۵ باریکه‌های توازن بر روی تمام دیسک‌ها توزیع می‌شود. یک روش توزیع طرح نوبت گردشی است که در شکل زیر مشاهده می‌کنید.



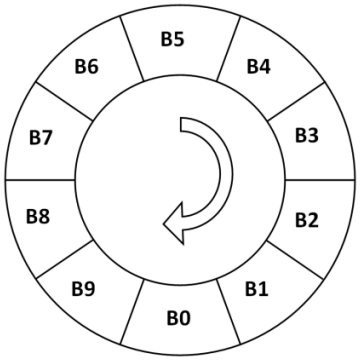
RAID سطح ۵ (توازن توزیعی سطح بلاک)

د) حافظه نهان دیسک

حافظه نهان (Cache memory) به حافظه‌ای گفته می‌شود که کوچک‌تر و سریع‌تر از حافظه اصلی است و بین حافظه اصلی و پردازنده قرار می‌گیرد. این نوع حافظه با استفاده از اصل محلی بودن از میانگین زمان دستیابی به حافظه می‌کاهد. چنین قاعده‌ای را می‌توان برای حافظه دیسک نیز درنظر گرفت. حافظه نهان دیسک حاوی نسخه‌ای از چند قطاع دیسک است. وقتی درخواستی برای داده خاصی صورت می‌گیرد، بررسی می‌شود که آیا آن داده در حافظه نهان دیسک وجود دارد یا خیر، اگر وجود دارد که درخواست از طریق حافظه نهان برآورد می‌شود وگرنه درخواست به دیسک ارجاع داده می‌شود تا داده خوانده شده و به حافظه نهان دیسک منتقل شود. به دلیل محلی بودن ارجاعات وقتی بلوکی از داده‌ها به حافظه نهان واکشی می‌شود، احتمالاً درخواست-های بعدی نیز در همان بلوک داده‌ای منتقل شده می‌باشد.

هـ) تکنیک درهم یا تداخل بلاک‌ها (Interleaving)

فرض کنید تعداد ۱۰ بلاک از B0 تا B9 در شیاری قرار دارند بطوریکه بلاک‌هایی که از نظر منطقی همجوار هم هستند، بطور فیزیکی بر روی یک شیار و بترتیب پشت‌سرهم قرار گرفته‌اند.

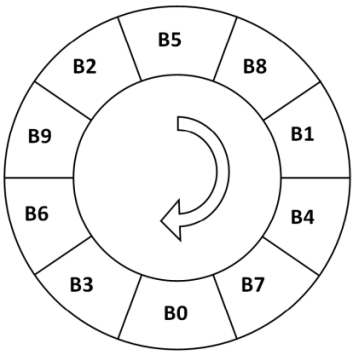


فرض کنید هِد خواندن/نوشتن بر روی بلاک B0 واقع است و فقط یک بافر وجود داردو خواندن هر بلاک 0.5 میلی‌ثانیه و زمان پردازش آن برابر 1 میلی‌ثانیه است. پس از خواندن بلاک B0 ، ۱ میلی‌ثانیه طول می‌کشد تا بلاک B0 پردازش شود. در این مدت هِد از روی بلاک‌های B1 و B2 عبور کرده و به بلاک B3 رسیده است. حال اگر بخواهیم بلاک را به ترتیب منطقی بخوانیم پس مجبور خواهیم بود تا یک دور زده و هِد خواندن نوشتن را بر روی بلاک B1 قرار دهیم (جهت چرخش از چپ به راست است).

در واقع برای خواندن بلاک B1 باید هِد خواندن/نوشتن ۸ بلاک را خوانده تا به سر بلاک B1 برسد و سپس بلاک B1 را در 0.5 میلی‌ثانیه بخواند. این مشکل در مورد خواندن بلاک B2 نیز وجود دارد. بنابراین می‌توانیم بگوییم که برای خواندن هر بلاک با توجه به مفروضات فوق، زمان 0.5 میلی‌ثانیه برای خواندن، زمان 1 میلی ثانیه برای پردازش در بافر و زمان 8×0.5 میلی‌ثانیه برای رفتن به سر بلاک بعدی در ترتیب منطقی نیاز دارد. بنابراین برای خواندن هریک از بلاک‌های B0 تا B8 زمان 0.5*8 + 1 + 0.5 میلی‌ثانیه نیاز خواهد بود بجز بلاک آخر B9 که نیاز به زمان رفتن به سر بلاک بعدی ندارد و زمان آن برابر 1 + 0.5 میلی‌ثانیه می‌باشد.

زمان لازم برای پردازش کل بلاک‌ها = 8*(0.5 + 1 + 8*0.5) + (0.5 + 1) = 45.5 ms

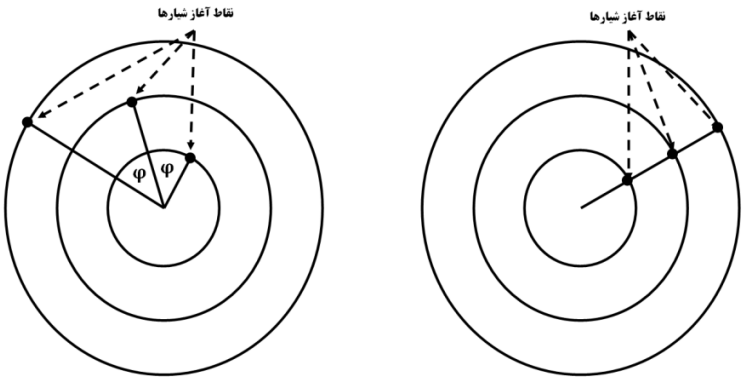
اگر بخواهیم بعد از پردازش بلاک B0، بلاک B1 بلافاصله در زیر نوک هِد خواندن/نوشتن قرار داشته باشد، باید بین بلاک B0 و B1 ۲ بلاک وجود داشته باشد در اینصورت بعد از خواندن هر بلاک در 0.5 میلی ثانیه و پردازش آن در بافر در زمان 1 میلی‌ثانیه، بلاک بعدی در زیر نوک خواندن/نوشتن وجود دارد و دیگر نیاز به صرف زمانی برای رفتن به سر بلاک بعدی نیست.



زمان لازم برای پردازش کل بلاک‌ها بعد از درهم‌چینی = 9*(0.5 + 1) = 13.5 ms

ی) تغییر نقطه آغازین شیارها

در حالت معمولی نقطه آغاز شیارها به‌صورتی است که در یک راستا بوده و یک شعاع صفحه را تشکیل می‌دهند. اما در این روش نقطه آغاز هر شیار نسبت به شیار قبلی زاویه **φ** می‌سازد. بدین ترتیب در بلاک‌های متوالی در شیارهای متعدد ذخیره شده و در هنگام حرکت از یک شیار به شیار دیگر هِد خواندن/نوشتن در سر بلاک بعدی قرار می‌گیرد.



فصل پنجم :

انواع ساختارهای فایل‌ها

در هر سیستم فایل اهداف اصلی عبارتند از:

- سرعت عملیات در بازیابی و ذخیره سازی
- صرفه جویی در حافظه

برای رسیدن به این دو هدف در طراحی سیستم های ذخیره و بازیابی باید به معیارهایی همچون حداقل بودن میزان افزونگی داده-ها، دستیابی سریع، سهولت در عملیات بهنگام‌سازی، سهولت درنگهداری سیستم، قابلیت اطمینان بالا، امنیت داده‌ها و ... توجه داشته باشیم.

برای رسیدن به این اهداف در سیستم فایل روشهای مختلف ذخیره سازی فایل ها را بررسی خواهیم نمود. انواع فایل‌ها و هر کدام با روش سازماندهی خاصی که دارند ما را برای رسیدن به این اهداف یاری می‌کنند.

انواع ساختارهای فایل عبارتند از:

۱-فایل بی‌نظم (Pile)

۲- فایل ترتیبی(Sequential)

۳- فایل ترتیبی شاخص‌دار(Indexed Sequential)

۴- فایل چند شاخصی یا شاخص بندی شده(Indexed file)

۵- فایل مستقیم (Direct file)

۶- فایل چندحلقه (Multi Ring file)

در ابتدا به معرفی و بیان خصوصیات و شرح تکنیکی هر کُدام از این فایل ها خواهیم پرداخت و سپس ساختار و ویژگی‌های هریک از آنها را تشریح خواهیم کرد.

برای ارزیابی ساختارهای مختلف فایل باید پارامترهای زیر را در نظر بگیرید :

۱ – T_F (Fetch) : زمان لازم برای واکشی یک رکورد از فایل .

۲ – T_N (Get Next) : زمان لازم برای واکشی رکورد بعدی .

۳ – T_I (Insert) : زمان لازم برای درج یک رکورد به درون فایل با توجه به ساختارها .

۴ – T_U (Update) : زمان لازم برای بهنگام سازی یک رکورد .

۵ – T_X (Exhaustive read) : زمان لازم برای خواندن تمام رکوردهای فایل .

۶ – T_Y (Restructuring) : زمان لازم برای سازمان دهی مجدد فایل .

دقت داشته باشید که عمل حذف حالت خاصی از بهنگام سازی است .

به طور کلی می توان فایل ها را به دو دسته ی متراکم (Dense) و پراکنده (Sparse) تقسیم بندی کرد .

- **فایل متراکم :** فایلی است که تمام رکوردها در تمام فیلدها دارای مقدار باشند.
- **فایل پراکنده :** فایلی است که بعضی از رکوردها در بعضی از فیلدها، مقدار نداشته باشند.

بدیهی است که اگر فایل را با رکوردهایی با قالب غیرثابت مکانی، طراحی کنیم حالت پراکنده (غیرمتراکم) پدید نمی‌آید. پس هنگامی فایل پراکنده است که رکوردهایی با طول ثابت و قالب ثابت مکانی داشته باشیم و طبعاً حافظه هرز ایجاد می‌شود.

افزونگی (Redundancy) :

افزونگی به معنای تکرار بعضی از صفات خاص فایل بیش از یک بار در محیط فیزیکی می‌باشد . افزونگی به دو دسته تقسیم می-شود .

۱) **افزونگی طبیعی** : صفت خاصه (فیلد) به صورتی است که یک مقدار مشخص از آن در تعدادی از نمونه ها (رکوردها) وجود دارد . مثلاً : شماره ی درس برای تمام دانشجویانی که آن درس را برداشته اند، تکرار می شود .

۲) **افزونگی تکنیکی** : تکرار بعضی یا تمام مقادیر یک یا چند صفت خاصه به خاطر ایجاد یک استراتژی خاص در دستیابی به فایل . مثل کلید خارجی .

این افزونگی را می‌توان با تکنیک‌هایی کاهش داد، به‌ویژه اگر صفت خاصه خود چند مقداری هم باشد(یعنی در یک نمونه از رکورد، چند مقدار از آن صفت خاصه وجود داشته باشد). چند مقداری موجب متغیر شدن طول رکوردها می‌شود، حال اگر وجود این پدیده با پدیده افزونگی نیز همراه باشد(که معمولاً نیز چنین است) افزونگی تشدید می‌شود. زیرا مقادیر یک صفت خاصه در تعدادی فیلد از یک نمونه رکورد وجود دارند و ضمناً در نمونه‌های دیگر نیز تکرار همان مقادیر دیده می‌شود.

برای کاهش مصرف حافظه در حالت وجود افزونگی طبیعی می‌توان از یکی از دو روش متدوال زیر استفاده کرد:

- طراحی فایل با ساختار مناسبتری
- استفاده از یک تکنیک فشرده‌سازی

عملیات ششگانه قابل انجام بر روی انواع ساختارهای فایل

اساساً شش نوع عمل روی فایل‌ها توسط سیستم‌فایل انجام می‌شود که عبارتند از:

۱- **واکشی رکورد دلخواه (Fetch) :** واکشی یک رکورد یک عمل محتوایی است که در آن یکی یا تعداد بیشتری از صفات خاصه رکورد به عنوان آرگومان جستجو مدنظر قرار داده می‌شود که لازمه این امر جستجو در فایل، دستیابی به بلاک حاوی رکورد موردنظر و خواندن آن می‌باشد، که رسیدن به بلاک مورد نظر بستگی به ساختار فایل دارد.

انواع درخواست های واکشی یک رکورد عبارتند از

- **درخواست ساده (Single request)** : درخواستی است که در آن عمل جستجو بر اساس یک صفت از رکورد انجام گیرد. مانند : جستجو با شماره دانشجوئی
- **درخواست محدوده‌ای (Range request)** : درخواستی است که در آن عمل جستجو در محدوده ای از مقادیر کلید(اصلی یا ثانویه) انجام می‌شود مانند : جستجوی دانشجویان با شماره دانشجوئی بین x و y .
- **درخواست تابعی (Functional request)** : درخواستی است که در آن جواب با انجام محاسبات و پردازش‌هایی بر روی فیلدها انجام می‌شود . به عنوان مثال برای بدست آوردن معدل باید بر روی نمرات دانشجو محاسباتی انجام شود.(با فرض اینکه معدل جزو فیلدها نیست)
- **درخواست بولین (Boolean request)** : درخواستی است که در آن صفات همراه با عملگرهای منطقی (AND و OR و XOR) در بازیابی رکوردها نقش دارند . مانند : دانشجویانی که قدشان x و ورودی سال y باشند .
- **درخواست مرکب (Composite request)** : درخواستی است که در آن عمل جستجو بر اساس چندین فیلد انجام می شود و در واقع حالت خاصی از درخواست بولین است

۲- **واکشی رکود بعدی (Get Next) :** بدست آوردن رکورد بعدی عملی است که وابسته به ساختار فایل می‌باشد در حالیکه واکشی یک رکورد (Fetch) عملی است محتوائی. موقعیت رکورد بعدی نسبت به رکورد فعلی می‌تواند یکی از سه حالت زیر باشد.

- با رکورد فعلی ارتباطی ندارد یعنی باید بر روی رکورد بعدی دوباره عمل جستجو انجام شود
- آدرس رکورد بعدی از رکورد فعلی بدست می‌آید(یعنی از رکورد فعلی به بعدی اشاره‌گر وجود دارد).
- رکورد بعدی همجوار فیزیکی با رکورد فعلی است

۳- **درج یک رکورد (Isert) :** عمل درج یک عمل نوشتن نیست بلکه باید در هنگام درج، ساختار منطقی نیز حفظ گردد. به عنوان مثال در فایل درهم یا بی‌نظم در هر کجای فایل عمل درج می‌تواند انجام گیرد ولی در فایلی با ساختار ترتیبی (منظم) رکورد باید در جای منطقی‌اش اضافه گردد.

- برای درج یک رکورد باید عملیات زیر انجام گیرد
- بازیابی بلاکی که رکورد باید در آن درج شود.
- نوشتن رکورد بر روی بلاکی که در بافر قرار دارد.
- بازنویسی بلاک
- انجام عملیات مربوط به تنظیم ساختار فایل(در صورت نیاز)

۴- **به‌هنگام‌سازی یک رکورد (Update) :** برای این عمل ابتدا باید رکورد مورد نظر بازیابی و پس از اعمال تغییرات بر روی آن دوباره در فایل نوشته شود. به طور کلی عمل بهنگام‌سازی به دو صورت انجام می‌گیرد.

- **بهنگام سازی درجا (Inplace) :** در این نوع بهنگام‌سازی رکورد به‌هنگام شده در جای قبلی نوشته می‌شود.
- **به‌هنگام سازی برون از جا (Out place) :** در این نوع به‌هنگام‌سازی رکورد به‌هنگام‌سازی شده در جای دیگری غیر از محل اصلی بازنویسی می شود

۵- **خواندن تمام فایل (Exhaustive read) :** : خواندن تمام فایل به دو صورت امکان پذیر است.

- **پی‌درپی :** یعنی بلاکها به ترتیب از اول تا آخر پشت سر هم خوانده می‌شوند.
- **سریال :** عمل خواندن بر اساس ترتیب مقادیر یکی از صفات خاصه (معمولاً کلید اصلی) انجام می‌پذیرد

خواندن سریال در واقع یک سلسله عملیات Get Next است. بنابراین هرگاه در ساختاری عمل بازیابی رکورد بعدی ناممکن باشد، خواندن سریال هم ناممکن است.

۶- **سازماندهی مجدد فایل (Restructuring) :** هر فایل پس از لود اولیه به علت انجام عملیات بازیابی یا ذخیره‌سازی، دستخوش تغییراتی می‌شود که موجب کاهش کارآیی اولیه آن خواهد شد.

دلایل سازمان‌دهی مجدد فایل عبارتند از

- احیاء نظم ساختار اولیه فایل
- از بین بردن حافظه‌های هرز موجود
- اصلاح استراتژی دستیابی (در فایل‌های شاخص‌دار)

جهت سازمان‌دهی مجدد فایل، ابتدا فایل باید به طور کامل خوانده شده(به صورت سریال یا پی‌درپی، بسته به نوع ساختار فایل) پس از حذف رکوردهای حذف‌شدنی، باقیمانده رکوردها را با توجه به ساختار فایل بازنویسی کرد و در آخر در صورت نیاز باید بازسازی ساختار مربوط به استراتژی دستیابی را انجام دهیم.

زمان بازنویسی بلاک (T_{RW}) :

همانگونه که گفته شد در عملیات درج، بهنگام‌سازی و سازمان‌دهی مجدد بعد از خواندن بلاک‌ها و انجام عملیات بر روی آنها عمل بازنویسی بلاک را خواهیم داشت . زمان بازنویسی یک بلاک همان انتقال محتوای بافر به دیسک است که می‌توان برابر زمان خواندن بلاک در نظر گرفت ($S + r + b_{tt}$). در صورتی که انتقال به بلاک به بافر سریع انجام گیرد و زمان لازم برای انجام عملیات در بافر کوتاه باشد ($C_B \ll 2r$). می توان بلاک را در همان دور جاری در دیسک بازنویسی کرد. در اینصورت

ولی اگرعملیات بافر به موقع انجام نشود آن گاه ممکن است سیستم یک دور را از دست بدهد و تا دور بعدی منتظر بماند. به عنوان مثال :

در بحث‌های آتی، غالباً فرض بر این است که $C_B \ll 2r$

فایل بی‌نظم، پایل (Pile) :

رکوردها در این نوع فایل بر اساس هیچ فیلد یا فیلدهایی مرتب نیستند. در واقع در این نوع ساختار که ساده‌ترین ساختار جهت پیاده‌سازی می‌باشد، رکوردها دارای طول متغیر هستند. تعداد فیلدها و همچنین مکان آنها نیز در نمونه‌های مختلف رکوردها متفاوت می‌باشد. برای ساختن این فایل رکوردها بخش بندی نشده و هیچ استراتژی مشخصی برای دستیابی به رکوردها وجود ندارد. ساختمان رکورد در این فایل به صورت زیر می‌باشد.

$$\boxed{A_1 = V_1 \ , \ A_2 = V_2 \ , \quad \ldots}$$

که در ساختمان فوق A_i نام فیلد یا اسم صفت و V_i مقدار آن فیلد می باشد.
دقت داشته باشید تعداد رکوردها و محل قرار گرفتن آنها برای نمونه‌های مختلف متفاوت است. با توجه به ساختار ذخیره‌سازی رکوردها در این شیوه برای دستیابی به رکوردها هیچگونه استراتژی خاصی نداریم و باید رکوردها را به صورت ترتیبی (یعنی همه را برای پیدا کردن یک رکورد خاص) جستجو کنیم و از طرفی نمی‌توانیم از ابتدای یک رکورد بر سر رکورد بعدی پرش کنیم چرا که اندازه هر رکورد متغیر بوده و به ناچار مجبور به خواندن کلیه مقادیر رکوردها می‌باشیم .

کاربرد و موارد استفاده فایل pile :

۱) در محیط‌های عملیاتی که داده‌ها اساساً نظم‌پذیر نباشند(وجود نظم درجه ایمنی فایل را کاهش می‌دهد).
۲) در محیط‌هایی که فقط عملیات خواندن از ابتدا به انتهای فایل نیاز است .

ارزیابی کارایی فایل Pile :

۱- **زمان واکشی یک رکورد (T_F) :** همانگونه که گفته شد برای دستیابی به رکوردها به دلیل عدم وجود ساختار مشخص و ثابت باید کلیه رکوردهای فایل خوانده شود. در حالت کلی می‌توان گفت برای واکشی یک رکورد باید نصف رکوردها بررسی گردد چرا که به دلیل بی‌نظم بودن رکوردها ممکن است که رکورد در اولین بلاک فایل و یا آخرین بلاک فایل باشد، پس به طور متوسط باید نصف بلاک‌ها جستجو شوند. در صورتی‌که تعداد بلاک‌های فایل را **b** بلاک فرض کنیم که هر کدام **B** بایت حافظه اشغال کنند و تعداد کل رکوردها برابر **n** درنظر بگیریم انگاه زمان واکشی T_F از فرمول‌های زیر بدست می آید .

$$T_F = \frac{1}{2} b \frac{B}{\acute{t}} \qquad\qquad T_F = \frac{1}{2} n \frac{R}{\acute{t}}$$

چون تعدادی بلاک باید خوانده شود لذا نرخ انتقال انبوه ($\acute{t}$) در نظر گرفته می‌شود.
البته برای رسیدن به ابتدای فایل، زمان S+r نیز صرف می‌گردد که قابل صرف‌نظر کردن می‌باشد.

۲- **زمان بازیابی رکود بعدی (T_N) :** در این ساختار رکورد بعدی مفهومی ندارد چرا که هیچ گونه ارتباطی بین رکورد فعلی و رکورد بعدی برقرار نیست . بنابراین برای پیدا کردن آن باید کل فایل مورد جستجو قرار گیرد(این در صورتی است که بدانیم رکورد بعدی چه رکوردی خواهد بود).

$$T_N = T_F$$

۳- **زمان درج یک رکورد (T_I) :** با توجه به ساختار بی‌نظم فایل pile، برای درج‌کردن یک رکورد در این فایل باید به انتهای فایل رفته و عمل درج انجام گیرد. بنابراین ابتدا باید بلاک آخر فایل خوانده شده و بعد از درج رکورد در بلاک موجود در بافر، بلاک را بازنویسی کنیم. برای رفتن به انتهای فایل و خواندن بلاک انتهائی زمان $S+r+b_{tt}$ صرف می شود و بازنویسی بلاک را نیز T_{RW} در نظر می گیریم . بنابراین

$$T_I = S + r + b_{tt} + T_{RW}$$

دقت کنید در این فرمول، از زمان اضافه کردن رکورد در بلاک موجود در بافر به علت اینکه عملیات مربوط به آن در حافظه صورت می گیرد و بسیار سریع می‌باشد، صرف‌نظر کرده‌ایم.
همانگونه که گفته شد اگر فرض کنیم $C_B \ll 2r$ یعنی اضافه‌کردن رکورد به بلاک موجود در بافر خیلی کمتر از یک دور دیسک باشد آن گاه خواهیم داشت:

$$T_I = S + r + b_{tt} + 2r = S + 3r + b_{tt}$$

دقت داشته باشید که در فرمول‌های فوق فرض کرده‌ایم که آدرس آخرین بلاک فایل را در اختیار داریم در غیر اینصورت مجبوریم کل فایل را از ابتدا به انتها خوانده تا به آخرین بلاک برسیم

۴- **زمان به‌هنگام‌سازی یک رکورد (T_U) :** همانگونه که قبلاً گفته شد عمل به‌هنگام‌سازی به دو صورت انجام می پذیرد: درجا و برون از جا. ابتدا بهنگام‌سازی درجا را در این ساختار بررسی می‌کنیم. اگر طول رکورد قبل و بعد ازعمل بهنگام‌سازی تغییر نکند پس می‌توانیم رکورد را در همان جای قبلی‌اش بازنویسی کنیم. در این حالت زمانی را که جهت خواندن رکورد به حافظه صرف خواهد شد. در صورتیکه فرض کنیم زمان عمل بازنویسی و تغییر رکورد در حافظه کمتر از یک

دور چرخش دیسک است ($C_B \ll 2r$) آنگاه در دور بعدی هِد دیسک، می‌توانیم آن را بازنویسی کنیم که این زمان برابر $2r - b_{tt}$ و برای بازنویسی رکورد نیز زمان b_{tt} را صرف خواهیم کرد در نتیجه خواهیم داشت:

$$T_{U_{Inplace}} = T_F + 2r - b_{tt} + b_{tt} = T_F + 2r$$

ولی در صورتیکه طول رکورد عوض شود، دیگر نمی‌توانیم در جای قبلی آن عملیات درج را انجام دهیم. بنابراین مجبوریم که رکورد را به عنوان یک رکورد جدید در انتهای فایل درج کرده و آن را از جای قبلی‌اش حذف کنیم، در نتیجه خواهیم داشت:

$$T_{U_{Outplace}} = T_F + T_{RW} + T_I = T_F + 2r + T_I$$

۵– زمان حذف یک رکورد (T_D) : برای حذف یک رکورد ابتدا باید آن را واکشی کرد و سپس علامت حذف منطقی را در آن قرار داده و دوباره رکورد را بازنویسی کنیم چون در این حالت اندازه رکورد تغییری نمی‌یابدT بنابراین همان بهنگام‌سازی درجا خواهد بود. در نتیجه:

$$T_D = T_F + T_{RW} = T_F + 2r$$

۶– زمان خواندن تمام فایل (T_X) : : همانگونه که گفته شد خواندن فایل به دو صورت پی‌درپی و سریال انجام می‌پذیرد که هر کدام را بررسی می‌کنیم.

در حالت بازیابی فایل به صورت پی‌درپی تمام رکوردهای فایل از ابتدا تا انتها بدون در نظر گرفتن هیچ گونه ترتیب منطقی خاصی خوانده می‌شود بنابراین خواهیم داشت:

$$T_{X_{seq}} = b \times \acute{b}_{tt} = 2 \times \left(\frac{1}{2} \times b \times \acute{b}_{tt}\right) = 2T_F$$

دقت داشته باشید فایل pile را نمی‌توان بصورت سریال خواند چرا که بازیابی رکورد بعدی در این فایل عملی نمی‌باشد . برای این کار باید ابتدا فایل را بر حسب فیلد خاصی مرتب کرده و سپس خوانده شود، که زمان مرتب سازی فایل را برابر $T_{sort}(n)$ برای n رکورد در نظر می گیریم . بنابراین

$$T_{X_{ser}} = T_{sort}(n) + T_{X_{seq}}$$

۷– زمان سازماندهی مجدد فایل (T_Y) : برای سازماندهی مجدد فایل ابتدا باید فایل خوانده و رکوردهایی را که به صورت منطقی حذف شده‌اند را و نوعی حافظه هرز محسوب می شوند را حذف کرد، بنابراین برای این عمل ابتدا باید فایل را خوانده و پس از حذف رکوردهای حذف شده دوباره بازنویسی کرد.

$$T_Y = (n + i)\frac{R}{\acute{t}} + (n + i - d)\frac{R}{\acute{t}}$$

در فرمول فوق مقدار کل رکوردها در لود اولیه برابر **n** فرض شده که از این رکوردها **d** تا رکورد، حذف منطقی و تعداد **i** رکورد جدید نسبت به حالت اولیه در انتهای فایل درج شده‌اند

$(n + i)\frac{R}{\acute{t}}$: خواندن کل فایل(فایل اولیه به همراه رکوردهای اضافه شده جدید)

$(n + i - d)\frac{R}{\acute{t}}$: بازنویسی فایل بدون در نظرگرفتن رکوردهای حذف شده منطقی و همراه با رکوردهای درج شده بعد از لود اولیه.

مثال۱ : در یک فایل پایل شامل 2000 رکورد با طول 160 بایت. مطلوبست بدست آوردن زمان لازم برای واکشی رکورد و زمان یافتن رکورد بعدی در این فایل درصورتیکه نرخ انتقال انبوه برابر 800 بایت بر میلی‌ثانیه باشد.

مثال۲ : در یک فایل پایل مشخصات زیر را داریم

$S = 8.5(ms) \qquad r = 2.5(ms) \qquad B = 250(B) \qquad t = 1000\left(\frac{B}{ms}\right)$

مطلوبست یافتن زمان لازم برای بهنگام‌سازی در این فایل.

مثال۳ : در یک فایل پایل اطلاعات زیر را داریم

$T_N = 125(\text{ms}) \qquad \acute{t} = 3000\left(\frac{B}{ms}\right) \qquad t = 500\left(\frac{B}{ms}\right)$

$S = 6.5(ms) \qquad r = 1.5(ms) \qquad b = 600$

مطلوبست یافتن زمان لازم برای بهنگام‌سازی این فایل.

مثال۴ : اگر زمان حذف در فایل پایل برابر 18.5 ms و زمان درنگ دورانی برابر 6.5 ms باشد، مطلوبست زمان لازم برای انجام عمل خواندن کل فایل موردنظر به صورت پی‌درپی

مثال۵ : زمان درج و یافتن رکورد بعدی در یک فایل پایل به‌ترتیب برابر 18 و 12.5 میلی‌ثانیه است. اگر تعداد رکوردهای این فایل برابر 450 رکورد فرض شود با توجه به داده‌های زیر، نرخ انتقال انبوه را در این فایل محاسبه نمائید.

$t = 300\left(\frac{B}{ms}\right) \qquad B_f = 8 \qquad r = 3.4(ms) \qquad S = 5.5(ms)$

مثال۶ : دریک فایل پایل، اگر زمان واکشی برای فایلی با 1200 رکورد برابر 18 ms باشد و نیز نرخ انتقال انبوه برابر $\frac{2}{3}$ نرخ انتقال اسمی و زمان استوانه‌جویی $\frac{2}{3}$ زمان درنگ دورانی آن باشد، با توجه به آنکه زمان بازنویسی برابر 8.5 ms می‌باشد، مطلوبست یافتن زمان لازم برای بهنگام‌سازی در این فایل ($B_f = 12$ فرض شده و شرط $C_B \ll 2r$ برقرار است)

مثال۷ : در یک فایل پایل، زمان بهنگام‌سازی $\frac{7}{5}$ زمان حذف از آن فایل و همچنین زمان حذف، خود $\frac{8}{3}$ زمان لازم برای انتقال بلاک به بافر می‌باشد. اگر زمان استوانه‌جویی و درنگ دورانی به ترتیب برابر 5.5 و 8.5 میلی‌ثانیه باشد، چه زمانی صرف انتقال هر بلاک به بافر خواهد شد.

مثال۸ : زمان حذف و بهنگام‌سازی در یک فایل پایل به‌ترتیب برابر 22.5 و 37 میلی‌ثانیه است. با وجود اطلاعاتی به‌شرح زیر مطلوبست زمان بازنویسی یک بلاک

$t = 2190\left(\frac{B}{ms}\right) \qquad B = 730(B) \qquad S = 6.4(ms)$

مثال۹ : دریک فایل پایل با زمان بهنگام‌سازی برابر 35 ms، زمان لازم برای درج و حذف برابر می‌باشد. اگر زمان درنگ دورانی برابر 7.5 ms باشد، زمان خواندن کل فایل به صورت پی‌درپی چند میلی‌ثانیه خواهد بود.

فایل با ساختار ترتیبی (Sequential) :

در این ساختار تمام رکوردها دارای ساختار و قالب یکسانی می‌باشند. بدین مفهوم که در این ساختار برای هر فیلد میزان فضای گرفته شده مشخص است. بنابراین دیگر نیازی به ثبت نوع رکورد و سایز آن همانند فایل pile نخواهد بود. یکی از ویژگیهای مهم این ساختار مرتب بودن آن بر اساس کلید اصلی (شامل فیلد یا فیلدها) می‌باشد. ویژگی کلید اصلی یکتایی مقدار آن است. بنابراین در این ساختار رکوردهایی با مقدار یکسان کلید نخواهیم داشت. این ویژگی یعنی یکتایی مقدارکلید اصلی موجب کاهش افزونگی در فایل خواهد شد، ولی آن را از بین نخواهد برد. در مقابل با داشتن طول ثابت برای فیلدها و در نتیجه رکوردها میزان انعطاف پذیری کاهش خواهد یافت چرا که دیگر این اجازه را در ضمن عملیات بهنگام‌سازی طول رکود تغییر کند، از ما سلب خواهد شد و از طرفی با ثابت بودن مدل رکوردها در صورتی که بخواهیم بعضی از فیلدها را مقداردهی کنیم به علت ثابت بودن ساختار مجبوریم فضایی برای آن فیلدها در نظر بگیریم هر چند که مقداری در آنها نباشد(NULL باشند).

V_1	V_2	V_3	----	V_n

این ساختار نسبت به ساختار فایل پایل دارای مزایایی است که عبارتند از:

۱) صرفه‌جویی در مصرف حافظه به خاطر عدم ذخیره‌سازی اسم صفت در نمونه رکورد
۲) ساده‌تر بودن قالب رکورد، به‌نحوی که رکورد ذخیره شده عملاً نگاشتی از آنچه در برنامه پردازشگر است، می‌باشد
۳) نرم‌افزار ساده‌تر برای ایجاد، مدیریت و پردازش فایل
۴) وجود یک استرتژی دستیابی،که در این نوع ساختار شیوه دسترسی ترتیبی است(رکوردها بر اساس کلید اصلی مرتب شده‌اند)
۵) پردازش سریال رکوردها سریع‌تر و راحتر انجام می‌پذیرد

معایب این ساختار عبارتند از:

۱) مصرف حافظه بیشتر به خاطر شکل ثابت و مکانی رکوردها (ممکن از در بعضی از نمونه‌ها، بعضی از فیلدها مقداردهی نشوند)
۲) وجود پدیده عدم تقارن (Asymmetry) : زیرا فقط دسترسی اساس فیلد یا فیلدهای کلید اصلی انجام می‌شود و سایر فیلدها نقشی ندارند و درواقع استراتژی دستیابی متکی به کلید اصلی است.
۳) کاهش انعطاف‌پذیری ساختار : نخست ازنظر طول رکورد، بدین مفهوم اگر بخواهیم طول بعضی از فیلدهای رکوردها تغییر کند، این امر امکان‌پذیر با توجه به ساختار نیست و باید از همان اول برای هر فیلد سایر حداکثری را که نیاز خواهیم داشت درنظر بگیریم و دیگر از نظر انجام عملیات فایلی (درج، حذف و بهنگام‌سازی) که این ساختار فاقد انعطاف‌پذیری است. مثلاً در مورد درج، رکورد باید در نقطه خاصی درج شود تا نظم ساختار فایل ترتیبی حفظ شود که این امر بدلیل شیفت زمان‌بر است.

دقت داشته باشید در عمل درج یک رکورد در این فایل دیگر نمی‌توان این عمل را با اضافه کردن رکورد به انتهای فایل انجام داد، چرا که در اینصورت ساختار و ترتیب منطقی رکوردها به هم خواهد خورد. پس باید آن را در جای منطقی‌اش درج کنیم که این مستلزم زمان زیادی به علت شیفت‌های مربوط به رکورد می‌باشد. بنابراین از فایلی به نام T.L.F استفاده می‌کنیم .

فایل ثبت تراکنش‌ها T.L.F (Transaction Log File) :

جهت بالا بردن سرعت عملیات و پردازش فایل، عمل درج در فایل اصلی انجام نمی‌شود بلکه در یک فایل کمکی به نام ثبت تراکنش‌ها انجام می‌پذیرد که در آن رکوردهای اضافه شونده‌ی جدید درج خواهند شد، چرا که اگر قرار باشد رکورد در جای اصلی خود در فایل اولیه ذخیره گردد مجبوریم تعدادی عمل شیفت انجام دهیم که بسیار زمان‌بر است. مخصوصاً وقتی که حجم فایل زیاد باشد. بنابراین رکوردهای جدید را در انتهای فایل T.L.F ذخیره می‌کنیم(دقت داشته باشید که فایل T.L.F مرتب نمی باشد). سپس بعد از هر چند وقت فایل اصلی را سازمان دهی مجدد کرده و آنگاه است که رکوردهای فایل T.L.F در جای اصلی خود قرار می‌گیرد .

ارزیابی کارآیی فایل ترتیبی :

با توجه به مطالب گفته شده حال به بررسی پارامترهای موجود برای فایلی با ساختار ترتیبی می‌پردازیم.

۱- **زمان واکشی یک رکورد (T_F) :** با توضیح بالا، در هنگام جستجوی یک رکورد بعد از جستجو در فایل اولیه فایل T.L.F به صورت خطی جستجو خواهد شد(چرا که مرتب نبوده)، حال اگر فرض کنیم که فایل T.L.F دارای i رکورد می‌باشد آنگاه زمان بازیابی یک رکورد برابر خواهد بود با :

$$T_F = \frac{1}{2}(n+d)\frac{R}{t}$$

دقت داشته باشید که در اینجا فرض براین است که جستجو براساس فیلد یا فیلدهایی غیر ازکلید اصلی انجام شده است. اگر بخواهیم بر روی فیلدی غیر از کلید اصلی جستجو انجام دهیم،باید فایل را از ابتدا تا انتها همانند فایل pile پردازش کنیم، چرا که فایل براساس سایر فیلدها مرتب نبوده و بی‌نظم است.

در صورتیکه بخواهیم عملیات جستجو را بر اساس کلید اصلی انجام دهیم در این صورت زمان جستجو بسته به نوع الگوریتمی است که ما استفاده می‌کنیم. به عنوان نمونه دو روش جستجوی عمومی بر روی این نوع فایل که عبارتند از جستجوی باینری و جستجوی بلاکی را در زیر شرح می‌دهیم.

الف- جستجوی باینری : در این روش جستجو بر اساس کلید اصلی و در دو سطح انجام می پذیرد . در ابتدا جستجو برای پیدا کردن بلاک حاوی رکورد مورد نظر انجام گرفته و سپس در بلاک خوانده شده جستجو برای پیدا کردن رکورد انجام شد

$T_{F_{Binary\ Search}} = ($

$S + r + b_{tt}$: زمان خواندن یک بلاک

C_B : زمان پردازش یک بلاک در بافر که می‌توان بعلت کم بودن از آن صرف‌نظر کرد

log_2^b : تعداد دفعات مراجعه به فایل برای خواندن بلاک

$\frac{1}{2} \cdot i \cdot \frac{R}{t}$: زمان جستجوی خطی در فایل T.L.F (i تعداد رکوردهای فایل T.L.F)

برای اینکه بتوانیم الگوریتم جستجوی دودویی را پیاده‌سازی کنیم باید فایل در یک فضای پیوسته در دیسک ذخیره شده باشد چرا که در صورت تکه‌تکه بودن فایل در نقاط مختلف روی دیسک، پیداکردن آدرس بلاک وسطی فایل به آسانی انجام نمی‌گیرد.

ب- جستجوی بلاکی : این جستجو نیز مانند جستجوی قبلی است با این تفاوت که جستجو در داخل بلاک انجام می‌گیرد. کلید رکورد مورد نظر با کلید آخرین رکورد بلاک مقایسه می‌گردد، در صورتیکه از آن بزرگتر باشد به بلاک بعد پرش کرده و در صورت کوچکتر بودن محتویات همان بلاک را برای پیدا کردن رکورد مورد جستجو قرار می‌دهد. زمان اجرای این روش رابطه مستقیمی با فاکتور بلاک‌بندی دارد، بنابراین محاسبه مقدار بهینه B_f کمکی برای به حداقل رساندن زمان جستجو می‌باشد.

در این روش به طور متوسط می‌توانیم مقدار بهینه ی B_f را محاسبه کنیم.

اگر N تعداد رکوردهایی باشد که باید بررسی شوند، می‌دانیم که بطور متوسط $\frac{b}{2}$ (نصف بلاکهای فایل) پردازش می‌شوند تا بلاک حاوی رکورد موردنظر پیدا گردد. بعد از پیدا کردن بلاک، داده‌های درون آن مورد جستجوی خطی قرار می‌گیرند، بنابراین خواهیم داشت:

$$N = \frac{b}{2} + \frac{B_f}{2} = \frac{1}{2}\left(\frac{n}{B_f} + B_f\right) \qquad , \qquad n = b \cdot B_f$$

برای آنکه N حداقل شود باید مشتق رابطه بالا را نسبت به B_f بدست آوریم

$$\frac{d(N)}{d(B_f)} = 0 \qquad \Rightarrow \quad \frac{1}{2}\left(\frac{-n}{B_f^2} + 1\right) = 0 \qquad \Rightarrow \qquad B_f = \sqrt{n}$$

بنابراین اگر $B_f = \sqrt{n}$ باشد، آنگاه تعداد رکوردهایی که باید بررسی شوند تا به رکورد مورنظر برسیم، حداقل خواهند بود.

۲- زمان بدست آوردن رکورد بعدی (T_N) : در این ساختار رکورد بعدی معمولاً بلافاصله بعد از رکورد فعلی قرار دارد ولی امکان دارد که در بلاک بعدی قرار داشته باشد با این توضیح احتمال اینکه رکورد بعدی در همان بلاک خوانده شده که در بافر موجود است قرار داشته باشد برابر $1 - \frac{1}{B_f}$ است و به احتمال $\frac{1}{B_f}$ نیز رکورد فعلی آخرین رکورد بلاک است که برای دستیابی به رکورد بعدی نیاز داریم که بلاک بعدی را بخوانیم بنابراین

$$T_N = \left(1 - \frac{1}{B_f}\right) \times 0 + \frac{1}{B_f}(r + b_{tt}) = \frac{r + b_{tt}}{B_f}$$

در فرمول فوق زمان لازم برای خواندن بلاک بعدی را برابر $r + b_{tt}$ در نظر می‌گیریم و زمان S را برابر صفر، چرا که هِد خواندن/نوشتن بر روی بلاک قرار دارد، بنابراین دیگر زمانی را جهت پیدا کردن استوانه بلاک بعدی (S) صرف نخواهیم کرد.

در صورتیکه رکورد بعدی در فایل T.L.F باشد به علت عدم ارتباط منطقی بین فایل اصلی و فایلT.L.F مجبور به یک جستجوی خطی در فایل T.L.F خواهیم بود که این جستجو امکان پذیر نیست مگر آن که کلید رکورد بعدی را داشته باشیم

۳- زمان درج یک رکورد (T_I) : در هنگامی که فایل اصلی کوچک باشد، رکورد را در جای منطقی‌اش درج می کنیم ولی در مورد فایل‌هایی با حجم زیاد عمل شیفت بسیار زمان‌گیر می‌باشد و باید در فایل T.L.F ذخیره گردد. در دو حالت فوق زمان درج را بررسی خواهیم کرد:

در حالت اول رکورد باید در سر جای خود درج شود و سپس کلیه رکوردهای بعد از آن یک واحد به سمت انتهای فایل شیفت داده شوند، چون دقیقاً مشخص نیست که رکورد در کجا باید درج شود، به طور متوسط نصف رکوردهای فایل شیفت داده می شوند بنابراین خواهیم داشت

$$T_I = T_F + \frac{1}{2}b(b_{tt} + T_{RW}) \qquad b_{tt} = \frac{B}{t}$$

T_F : زمان پیدا کردن محل منطقی رکورد جدید جهت درج

$b_{tt} + T_{RW}$: زمان شیفت یک بلاک

در حالت دوم یعنی حالتی که حجم فایل زیاد می‌باشد بعلت زمان‌گیر بودن عمل شیفت، عمل درج در انتهای فایل T.L.F انجام می‌پذیرد تا بعداً در سازمان‌دهی مجدد فایل به سر جای اصلی خود باز گردد.

$$T_I = S + r + b_{tt} + T_{RW} = S + 3r + b_{tt}$$

$S + r + b_{tt}$ زمان صرف‌شده برای خواندن آخرین بلاک فایل ثبت تراکنش‌ها T.L.F (البته به‌شرط داشتن آدرس آخرین بلاک) است و در گردش بعدی دیسک یعنی 2r آنرا در سر جای خود (بعد از اضافه کردن رکورد جدید در بلاک خوانده شده و موجود در بافر) بازنویسی می‌کند.

البته برای اطمینان بیشتر از اینکه شرط یکتائی کلید اصلی رعایت شود ابتدا زمان T_F بر روی فایل اصلی صرف می‌گردد تا مشخص شود که آیا رکورد جدید دارای مقدار کلید تکراری است یا خیر و در صورت تکراری نبودن عملیات درج انجام می پذیرد.

۴- **زمان حذف یک رکورد (T_D) :** برای حذف یک رکورد از دو روش حذف منطقی وحذف فیزیکی استفاده می شود. در روش فیزیکی باید کلید رکوردهای بعد از آن یک واحد به ابتدای فایل شیفت داده شوند که این امر زمان‌گیر است به همین علت از حذف منطقی استفاده می‌شود که برای انجام دادن آن ابتدا رکورد خوانده شده و سپس فیلد مربوط به حالت حذف آن مقدار دهی شده و دوباره رکورد را بازنویسی می کنیم. بنابر این

$$T_D = T_F + T_{RW} = T_F + 2r$$

البته دقت داشته باشید که در اینجا نیز زمان پردازش بلاک در بافر را کمتر از یک دور دیسک درنظر گرفتها یم $C_B \ll 2r$

۵- **زمان بهنگام‌سازی رکورد (T_U) :** در عمل بهنگام سازی دو حالت پیش می آید:

اول اینکه بهنگام‌سازی برروی فیلد یا فیلدهایی غیر از کلید اصلی انجام پذیرد، بنابراین بعد از انجام عمل بهنگام‌سازی رکورد در همان جای قبلی‌اش بازنویسی می‌شود چرا که کلید اصلی تغییر نکرده، پس مکان منطقی فایل تغییری نمی-کند بنابراین

$$T_U = T_F + T_{RW} = T_F + 2r$$

ولی در حالت دوم بهنگام‌سازی برروی کلید اصلی انجام می‌شود، بنابراین دیگر در هنگام بازنویسی نمی‌توانیم رکورد را در سر جای قبلی‌اش بازنویسی کنیم چراکه ترتیب رکوردها بر هم خواهد خورد. بنابراین ابتدا رکورد را خوانده و به صورت منطقی حذف می‌کنیم و بعد از آن رکورد جدید را با کلید تغییر یافته در انتهای فایل اضافه خواهیم کرد .بنابراین

$$T_U = T_D + T_F$$

۶- **زمان خواندن کل فایل (T_X) :** در صورتی که بخواهیم کل فایل را به صورت پی‌درپی بخوانیم خواهیم داشت

$$T_{X_{Seq}} = (n + i)\frac{R}{t}$$

که در فرمول فوق، n تعداد رکوردهای فایل در لود اولیه، i تعداد رکوردهای فایل T.L.F می‌باشد.

ولی در صورتیکه بخواهیم فایل را به صورت سریال بخوانیم ابتدا باید بخشT.L.F را مرتب‌سازی کنیم.بنابراین

$$T_{X_{Ser}} = T_{sort}(i)$$

البته در این عمل کل رکوردها بصورت مرتب خوانده نمی‌شوند، بلکه هر قسمت (فایل اصلی و بخش T.L.F) مرتب خوانده می‌شوند. در صورتیکه بخواهیم کل رکوردها را به‌صورت مرتب بازیابی کنیم باید فایل را سازماندهی مجدد کنیم.

۷- **زمان سازماندهی مجدد فایل (T_Y) :** برای سازماندهی مجدد فایل ترتیبی ابتدا بخش فایل تراکنشی (T.L.F) مرتب‌سازی می‌شود وسپس با خواندن کلیه رکوردهای فایل اصلی و فایل تراکنشی و حذف رکوردهای حذف شدنی منطقی، فایل سازماندهی شده جدید را به صورت مرتب و بدون حافظه‌های هرز خواهیم داشت.

$$T_Y = T_{sort}(i) + n\frac{R}{t} + i\frac{R}{t} + (n + i - d)\frac{R}{t}$$

$T_{sort}(i)$: زمان مرتب‌سازی فایل T.L.F با i رکورد

$n\frac{R}{t}$: زمان خواندن فایل اصلی با n رکورد لود اولیه

$i\frac{R}{t}$: زمان خواندن فایل T.L.F با i رکورد

$(n + i - d)\frac{R}{t}$: زمان بازنویسی دوباره رکوردها به همراه حذف d رکورد حذف‌شده منطقی

موارد استفاده ی فایل ترتیبی :

۱) کاربردهای تجاری
۲) وقتی پردازش به صورت دسته ای یا Batch باشد.
۳) هنگامی که نیاز به رکوردی با طول ثابت داریم.

مثال۱ : زمان مرتب‌سازی یک فایل ترتیبی با 1024 رکورد و فایل T.L. F با 86 رکورد، که در آن هر رکورد دارای سایزی برابر ۱۲۰ بایت می‌باشد، 10.5 ms است. با توجه به آن که نرخ انبوه مربوط به این فایل 3000 بایت بر میلی‌ثانیه است. مطلوبست زمان خواندن کل فایل به صورت سریال

مثال۲ : اگر بخواهیم با جستجو به روش پرش بلاکی در یک فایل ترتیبی با حداقل تعداد مقایسه‌ها برای یافتن رکورد اقدام کنیم با وجود 625 رکورد تعداد مقایسه ها چقدر خواهد بود ؟

مثال۳ : در یک فایل ترتیبی از الگوریتم پرش بلاکی استفاده می‌شود. اگر در این فایل 200 رکورد وجود داشته باشد که طول هر یک برابر 75 بایت و طول هر بلاک 300 بایت باشد. تعداد مقایسه های لازم برای یافتن یک رکورد چقدر است

مثال۴ : برای انجام عمل واکشی با استفاده از تخمین و کاوش باید 120 بلاک خوانده شود تا رکورد موردنظر یافته شود. اگر اطلاعات زیر از این فایل موجود باشد، زمان موردنیاز برای واکشی یک رکورد را پیدا کنید.

$S = 6(ms) \qquad r = 10.5(ms) \qquad B = 620(B) \qquad t = 1500\left(\frac{B}{ms}\right) \qquad \acute{t} = 4000\left(\frac{B}{ms}\right)$

مثال۵ : در یک فایل ترتیبی 400 رکورد در فایل اصلی و 50 رکورد در فایل تغییرات وجود دارد. اگر طول هر بلاک برابر 110 بایت و نرخ انتقال انبوه برابر 3100 بایت بر میلی‌ثانیه باشد، با در اختیار داشتن T_{RW} برابر 14 میلی‌ثانیه و b_f برابر 8، مطلوبست زمان عمل درج در این فایل.

مثال۶ : یک فایل حاوی 100 بلاک هر یک به طول 240 بایت قرار است یک رکورد را در بلاک 20 این فایل اضافه کنیم در صورتی که بخواهیم درج در جای صحیح و در فایل اصلی انجام گیرد با فرض اینکه زمان شیفت هر بلاک 0.05 میلی‌ثانیه و نرخ انتقال انبوه 1200 بایت بر میلی‌ثانیه باشد. زمان درج در این فایل را بدست آورید

مثال۷ : مطلوبست یافتن طول رکورد در صورتی که تمام فایل ترتیبی به صورت پی‌درپی خوانده شود. با فرض اینکه تعداد رکوردهای فایل اصلی 420، تعداد رکوردهای فایل T.L.F 80 ، نرخ انتقال انبوه 2900 بایت بر میلی‌ثانیه و زمان خواندن کل فایل به صورت پی‌درپی 100 میلی ثانیه باشد .

مثال۸ : اگر دریک فایل ترتیبی، مجموع تعداد رکوردهای فایل اصلی و تغییرات 300 باشد و زمان انتقال رکورد به بافر برابر 0.4 میلی‌ثانیه فرض شود، مدت زمانی که برای خواندن سریال این فایل صرف می‌شود 59 میلی‌ثانیه درنظر گرفته می‌شود. مدت زمانی که برای مرتب‌سازی فایل T.L.F صرف می‌شود را محاسبه نمائید.

مثال۹ : اگر در یک فایل ترتیبی بدون حذف هیچ رکوردی، تعداد رکوردهای موجود 150 و تعداد رکوردهای درجی 25 باشد، درصورتیکه زمان انتقال رکورد به بافر 0.32 ثانیه فرض شود و 50 میلی‌ثانیه زمان صرف مرتب‌سازی فایل T.L.F نیاز باشد، زمان سازماندهی مجدد در این فایل چقدر خواهد بود.

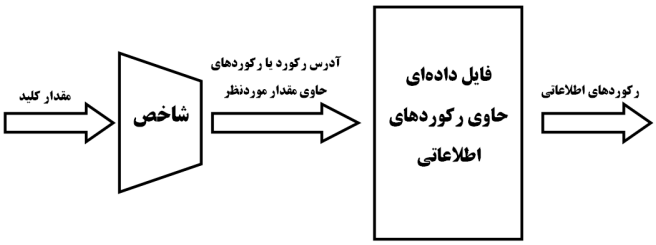
مثال۱۰ : در یک فایل ترتیبی که شامل 2500 بلاک به طول 1000 بایت می باشد . طول هر رکورد چقدر باشد تا تعداد کل مقایسه ها با روش پرش بلاکی به حداقل میزان خود برسد.

فایل با ساختار شاخص‌دار (Indexed) :

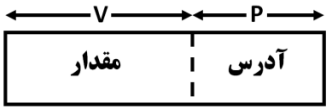
یکی از مشکلات فایل‌های ترتیبی زمانگیر بودن عمل واکشی رکورد بود که برای بهبود این مشکل از تکنیکی بنام شاخص‌بندی استفاده می‌کنیم. بدین مفهوم که فایل در لود اولیه یک فایل ترتیبی است . دستیابی بر اساس ترتیب کلید انجام می‌پذیرد. بعلاوه اینکه براس تسریع در عمل واکشی رکوردها، ساختار شاخص به آن اضافه می‌شود.

شاخص :

شاخص ساختمان داده‌ای است که مقدار کلیدی از رکورد را به عنوان ورودی می‌پذیرد و رکوردهایی با آن مقدار کلید (درصورت وجود) را سریعاً پیدا می‌کند.



تعریف: عبارتست از مجموعه‌ای از تعدادی مدخل که هر مدخل یک رکورد با طول V+P است. در این ساختار V میزان فضای فیلد یا فیلدهایی است که شاخص بر اساس آن ساخته شده است و P آدرس مستقیم رکورد یا مجموعه‌ای از رکوردها در فایل اصلی می‌باشد.



درصورتیکه فایل شاخص خود بزرگ باشد، جستجوی ترتیبی در آن زمانگیر خواهد بود. برای رفع این مشکل دو روش زیر پیشنهاد می‌گردد:

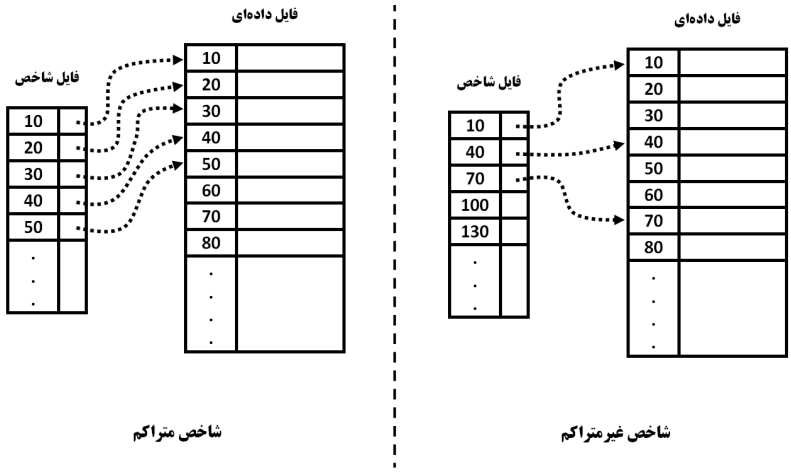
۱) استفاده از یک الگوریتم مناسب جهت جستجو مانند جستجوی باینری یا جستجوی بلاکی
۲) استفاده از سطوح مختلف شاخص (شاخص چند سطحی) که رایج‌تر می‌باشد.

لنگره‌گاه (Anchor opint) :

نقطه‌ای از فایل داده‌ای که از مدخل بخش شاخص به آن اشاره‌گر (نشانگر) داریم. از این نقطه نظر دو نوع شاخص داریم:

۱- **شاخص متراکم (Dense Index)** : اگر نقطه موردنظر در فایل یک رکورد باشد به آن شاخص متراکم می‌گوییم یا به بیان دیگر اگر هر مدخل فایل شاخص به یک رکورد اشاره کند.
اگر شاخص متراکم باشد، لازم نیست فایل داده‌ای بر حسب کلید مرتب باشد.

۲- **شاخص پراکنده یا غیرمتراکم (Non Dense Index)** : درصورتیکه هر مدخل فایل شاخص به گروهی از رکوردها در فایل داده‌اس اصلی اشاره کند، شاخص را غیرمتراکم می‌گویند. در این نوع شاخص فایل داده‌ای باید مرتب باشد چراکه در واقع عمل دسته‌بندی داده‌ها انجام می‌شود و یک نمونه از هر دسته در مدخل فایل شاخص قرار می‌گیرد.



بطور کلی جستجو بر اساس شاخص بسیار کارآمدتر از جستجو ترتیبی می‌باشد چراکه

- معمولاً فایل شاخص دارای تعداد بلاکهای کمتری نسبت به فایل داده‌ای می‌باشد بنابراین زمان واکشی کمتری دارد.
- چون کلیدها در فایل شاخص مرتب هستند، برای یافتن کلید از روش جستجوی دودویی استفاده می‌کنند. اگر تعداد بلاک‌های فایل شاخص برابر n درنظر بگیریم، با استفاده از جستجوی دودویی فقط log_2^n بلاک را جستجو خواهیم کرد.
- ممکن است بعلت کوچک بودن فایل شاخص آنرا در حافظه اصلی نگهداری کنیم که دراینصورت دیگر برای دسترسی به شاخص نیازی به دستیابی به دیسک نیست و این امر موجب افزایش سرعت در کار با فایل خواهد شد.

شاخص اصلی و شاخص(های) ثانویه :

در صورتیکه شاخص را بر روی کلید اصلی بنا کنیم، شاخص را شاخص اصلی می‌نامیم. اما هر فیلد دیگری نیز می‌تواند به عنوان فیلد مدخل ورودی شاخص انتخاب شود که در این حالت شاخص(های) ثانویه خواهیم داشت.

ظرفیت نشانه‌روی شاخص (Index fanout) :

فایل شاخص نیز همانند فایل اصلی دارای بلاک‌بندی می‌باشد و در هر بلاک تعدادی از سطرهای فایل شاخص قرار می‌گیرد. تعداد مدخل‌های یک بلاک شاخص را ظرفیت نشانه‌روی بلاک شاخص می‌گویند و با y نمایش می‌دهند.

$$y = \left\lfloor \frac{\text{اندازه بلاک شاخص}}{\text{طول مدخل شاخص}} \right\rfloor = \left\lfloor \frac{B}{V+P} \right\rfloor$$

ظرفیت نشانه‌روی در فایل شاخص همانند فاکتور بلاک‌بندی در فایل اصلی می‌باشد. در هر دو سایز بلاک برابر B می‌باشد ولی در فایل شاخص هر رکورد دارای سایز V+P می‌باشد ولی در فایل داده‌ای برابر R می‌باشد.

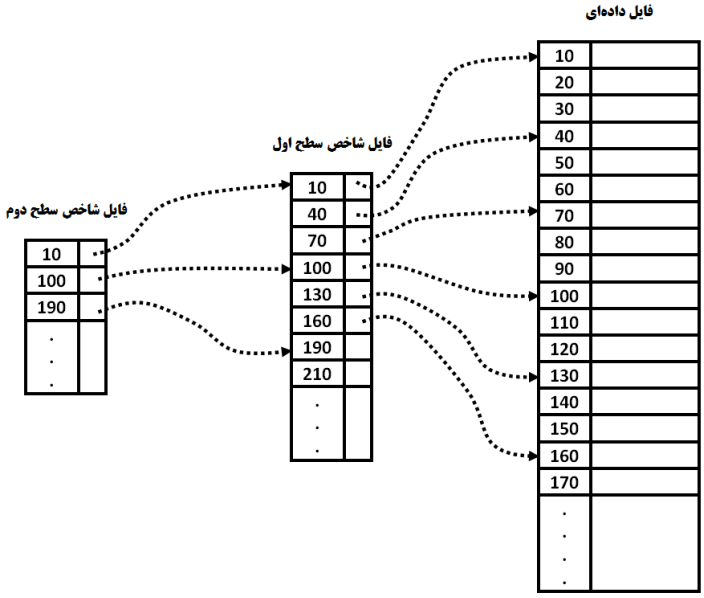
شاخص چندسطحی (Muli Level Index) :

همانگونه که دیدید شاخص می‌تواند چندین بلاک را اشغال کند. اگر این بلاک‌ها در جای مشخصی از دیسک (مثلاً در سیلندرهای خاصی) وجود نداشته باشند که بتوان براحتی به آن دسترسی داشت، برای یافتن آنها نیاز به ساختمان داده دیگری است. حتی اگر

بتوان براحتی آنها را پیدا کرد و سپس با استفاده از جستجوی دودویی به کلیدی در شاخص دست یافت باز هم در شرایط زیر نمی‌توان به یک سطح از شاخص اکتفا کرد

- شاخص بزرگ باشد بطوریکه استفاده از آن منجر به کارآیی زیادی نباشد (قطعه قطعه بودن فایل شاخص یا زیاد بودن زمان واکشی برای پیدا کردن کلید موردنظر)
- در شاخص پراکنده تعداد رکوردهای موجود در بلوک یا گروهی از رکوردها بسیار زیاد باشد.

در شاخص چندسطحی می‌توان شاخصی برای شاخص دیگر انتخاب کرد. به عنوان مثال در شاخص دو سطحی، شاخص جدیدی ایجاد می‌شود که به عنوان شاخصی برای سطح اول عمل می‌کند. این روند می‌تواند ادامه داشته باشد و شاخص‌ها به چندین سطح گسترش پیدا کنند. اما این روش محدودیت‌هایی دارد که به جای آن معمولاً از درخت‌های B استفاده خواهد شد.



در شاخص چندسطحی، شاخص سطح اول می‌تواند پراکنده یا متراکم باشد ولی شاخص‌های سطوح بالاتر حتماً پراکنده هستند.

تعداد سطوح را با x نمایش می‌دهیم. شاخص سطح x از نظر حافظه چنان است که در حافظه قرار دارد و به آن سرشاخص (Master Index) گفته می‌شود(بهتر است شاخص سطح x کمتر از یک بلاک باشد)

تعداد مدخل‌های سطح اول $b_1 = \frac{n}{B_f}$ سایز فایل شاخص در سطح اول $= b_1 \times (V + P)$

تعداد مدخل‌های سطح دوم $b_2 = \frac{b_1}{B_f}$ سایز فایل شاخص در سطح دوم $= b_2 \times (V + P)$

و به همین صورت الی آخر

تعداد سطوح شاخص $x = \left\lceil log_y^{b_1} \right\rceil = \left\lceil log_y^{\left(n/B_f\right)} \right\rceil$

هرچه تعداد سطوح بیشتر شود، دفعات دسترسی برای واکشی رکورد بیشتر خواهد شد و در نتیجه سرعت دسترسی کم خواهد شد، برای کاهش تعداد سطوح باید ظرفیت نشانه‌روی شاخص را افزایش داد که لازمه اینکار این است که با اندازه بلاک را زیاد کنیم یا طول مدخل شاخص را کاهش دهیم.

مثال : فرض کنید که یک فایل بزرگ با 10^7 رکورد داریم. اگر طول هر رکورد 200 باید درنظر بگیریم و طول بلاک در این سیستم ذخیره‌سازی 2000 بایت باشد. با فرض اینکه طول فیلد شاخص 12 بایت و طول فیلد آدرس 8 بایت باشد، مطلوبست اندازه فایل شاخص در هر سطح به همراه تعداد مدخل‌های هر سطح.

معایب ساختار چندسطحی:

۱- مصرف حافظه اضافی برای نگهداری سطوح شاخص
۲- فزونکاری در عملیات ذخیره‌سازی (به عنوان مثال در عملیات درج و حذف ممکن است لازم باشد تمام سطوح شاخص تغییر یابند)

شاخص چندسطحی بیشتر برای محیطهای عملیاتی مورد استفاده قرار می‌گیرد که ایستا باشند و عملیات درج و حذف و یا بهنگام-سازی در آن خیلی کم اتفاق بیفتد.

فایل با ساختار ترتیبی شاخص‌دار

با توجه به مطالب گفته شده اکنون می‌توانیم ساختاری کامل برای چنین فایلی ارائه دهیم.
در این ساختار علاوه بر یک فایل ترتیبی که فایل اصلی داده‌ای می‌باشد و در آن داده‌ها براساس کلید مرتب شده‌اند، جهت واکشی سریعتر رکوردها از فایل یا فایل‌هایی که شامل جداول شاخص بر روی فایل اصلی می‌باشد، نیز استفاده می‌شود. شاخص در این ساختار می‌تواند متراکم یا پراکنده باشد.
دراین ساختار از یک ناحیه سرریز (overflow area) نیز جهت انجام عملیات ذخیره‌سازی برای داده‌های جدید پس از لود اولیه فایل داده‌ای اصلی، استفاده می‌شود.
نکته: شاخص در این ساختار فقط به داده‌های موجود در فایل اصلی اشاره می‌کند و هیچ اشاره‌ای به داده‌های موجود در بخش سرریز نخواهد داشت. این داده‌ها در هنگام بازسازی مجدد فایل به داخل فایل اصلی انتقال داده می‌شوند و در بخش شاخص نیز وارد می‌شود.
نکته: در این ساختار، شاخص بصورت ایستا (Static) می‌باشد بنابراین در هنگام عملیات ذخیره‌سازی در فایل تغییری برروی شاخص انجام نخواهد گرفت.
نکته: از شاخص برای تسریع واکشی رکوردها استفاده می‌شود بنابراین در عملیاتی مانند خواندن تمام فایل بصورت پی‌درپی یا سریال، چندان کاربرد ندارد.

تکنیک‌های خواندن فایل ترتیبی شاخص‌دار

درهنگام پردازش فایل ترتیبی شاخص‌دار به دو صورت عمل می‌کنیم

۱- **خواندن فایل بصورت پی‌درپی :** که برای اینکار داده‌های موجود در ناحیه اصلی و ناحیه سرریز بصورت بلاک به بلاک خوانده می‌شوند.
۲- **خواندن فایل بصورت سریال :** برای خواندن فایل به این روش، باید فایل بر روی کلید اصلی مرتب باشد. مشکلی که در اینجا وجود دارد مربوط به ناحیه سرریزی است که داده‌های آن مرتب نیستند. برای اینکار رکوردها را از فایل اصلی بترتیب می‌خوانیم تا به مکانی برسیم که یک اشاره‌گر به ناحیه سرریزی دارد. این بدان معنی است که رکورد بعدی در ناحیه سرریزی و بعد از لود اولیه فایل اضافه شده، در نتیجه به ناحیه سرریزی مراجعه کرده و شروع به خواندن رکورد

می‌کنیم تا به اشاره‌گر به فایل اصلی برسیم. دوباره به سرجای خود در فایل اصلی بازگشته و این عمل را تا خواندن کل فایل اصلی و سرریز ادامه می‌دهیم (این تکنیک درجلوتر توضیح داده خواهد شد).

انتخاب فضا برای درج رکوردهای سرریزی

برای اضافه کردن رکوردهای جدید، همانگونه که اشاره شد باید یک ناحیه سرریزی داشته باشیم که به سه روش متداول می‌توانیم این ناحیه سرریزی را ایجاد کنیم:

۱) **درنظر گرفتن یک فایل جداگانه برای ناحیه سرریزی :** این روش بدلیل زمانگیربودن ارتباط بین فایل اصلی و فایل سرریزی چندان مناسب نمی‌باشد.

۲) **درنظر گرفتن فضایی از بلاک در لود اولیه برای رکوردهای جدید :** این روش هرچند که لوکالیتی را بهتر خواهد کرد ولی امکان توزیع نامناسب داده‌ها و درنتیجه پرشدن سریع بعضی از این حافظه و خالی بودن بقیه وجود دارد. درصورتیکه بتوانیم محدوده داده‌های سیستم را حدس بزنیم می‌توانیم تا حدودی از بوجود آمدن این مشکل جلوگیری کنیم.

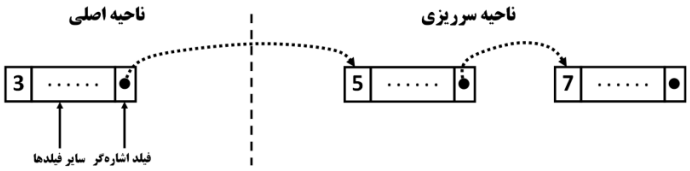
۳) **انتخاب فضایی از فایل اصلی برای ناحیه سرریزی :** که مناسب‌ترین روش می‌باشد که خود این روش به دو سورت قابل پیاده‌سازی است:

- اختصاص ناحیه‌ای از شیارهای انتهایی هر استوانه به ناحیه سرریزی که باعث قرارگرفتن رکوردهای جدید مربوط به هر استوانه در همان استوانه می‌شود و باعث کاهش زمان استوانه‌جویی نیز خواهد شد.
- اختصاص استوانه‌هایی در انتهای فایل برای ناحیه سرریزی که موجب کاهش لوکالیتی و افزایش S خواهد شد.

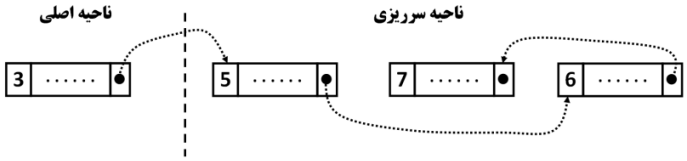
روش‌های درج رکوردهای در ناحیه سرریزی

دو تکنیک متداول وجود دارد که عبارتند از:

۱) **درج در اولین بلاک جا دار در ناحیه سرریزی :** در این روش رکورد در اولین بلاک جا دار در ناحیه سرریزی اضافه می‌شود و از رکوردی که منطقاً (در مرتب‌سازی منطقی) قبل از آن قرار دارد، اشاره‌گری به رکورد جدید ایجاد می‌شود. در این تکنیک برای هر رکورد ناحیه اصلی و ناحیه سرریزی فیلد اشاره‌گر درنظر می‌گیریم (هرچند که امکان null بودن آن وجود دارد).



همانگونه که می‌بینید درصورتیکه فیلد اشاره‌گر رکورد منطقی قبل از رکورد درج شده در ناحیه سرریزی به رکورد دیگر اشاره کند، زنجیره اشاره‌گر را دنبال می‌کنیم تا به محلی برسیم که رکورد منطقاً باید در آنجا درج شود و سپس اشاره‌گرها را تصحیح می‌کنیم. دقت داشته باشید که در انتهای هر زنجیره، رکوردی با فیلد اشاره‌گر null وجود خواهد داشت.
به عنوان مثال در شکل فوق، اگر بخواهیم رکوردی با کلید 6 را درج کنیم، بعد از تحصیحات مربوط به اشاره‌گرها خواهیم داشت.



۲) **درج با جابجایی (Push through) :** در این روش، رکورد جدید در بلاک مربوطه در ناحیه اصلی (درمحلی که باید منطقاً درج شود) قرار می‌گیرد و رکوردهای بعدی همان بلاک، به سمت انتهای بلاک شیفت داده می‌شوند و آخرین رکورد بلاک به اولین بلاک جادار در ناحیه سرریزی منتقل می‌شود. در این روش نیز زنجیره رکوردهای سرریزی هم ایجاد می‌شود ولی تفاوت اصلی آن این است که در این روش برای هر بلاک از ناحیه اصلی، یک اشاره‌گر به ناحیه سرریزی وجود دارد و نه برای هر رکورد یک اشاره‌گر به ناحیه سرریز(در روش قبل).

نکته : در روش درج با جابجایی، لوکالیتی رکوردهای بیشتر حفظ می‌گردد و همچنین پردازش فایل بصورت سریال در روش دوم ساده‌تر است.

نکته : در هیچ‌کدام از این دو تکنیک فایل شاخص تغییر نمی‌کند زیر در این ساختار، شاخص ایستا است و در واقع فقط یک ناظر به ناحیه اصلی داده‌ای می‌باشد.

موارد استفاده و کاربرد ساختار ترتیبی شاخص‌دار

بیشتر در محیط‌هایی که نیاز به پردازش سریال فایل برروی یک فیلد (کلید اصلی) مطرح باشد، مورد استفاده قرار می‌گیرد. همچنین در سیستم‌هایی که عمل واکشی رکوردها از طریق مقدار کلیدها انجام می‌شود.

ارزیابی کارآیی فایل ترتیبی شاخص‌دار:

۱) **زمان واکشی یک رکورد (T_F) :** برای واکشی یک رکورد در این ساختار، باید ابتدا سرشاخص (Master Index) که در حافظه اصلی قرار دارد را بررسی کنیم، سپس در سطوح شاخص آنقدر عمل بررسی و رفتن به سطح بعدی را انجام می‌دهیم تا به مدخل مربوط به سطح اول برسیم. بعد از پیدا کرد آدرس بلاکی که باید از ناحیه اصلی خوانده شود (این آدرس در بررسی سطح اول شاخص بدست می‌آید) بلاک را از ناحیه فایل داده‌ای اصلی به حافظه می‌آوریم. در مرحله بعد احتمال دارد که رکورد در بلاک موردنظر نباشد و مجبور به رجوع به ناحیه سرریزی و خواندن بلاک‌های این ناحیه (دنبال کردن زنجیره‌های) تا پیدا نمودن رکورد موردنظر ادامه می‌دهیم.

۲) **زمان بدست آوردن رکورد بعدی (T_N) :** برای بازیابی رکورد بعدی، باید از آخرین رکورد واکشی شده شروع کنیم، ببینیم آیا رکورد بعدی در بلاکی از ناحیه اصلی است یا در بلاکی از ناحیه سرریزی. بنابراین می‌توان گفت که به احتمال $\frac{1}{B_f}$ رکورد بعدی دربلاکی از ناحیه اصلی است، پس زمان خواندن آن برابر $\frac{1}{B_f} \cdot b_{tt}$ می‌باشد. و احتمال اینکه در ناحیه سرریزی باشد برابر $\frac{i}{n+i}$ می‌باشد، بنابراین خواهیم داشت.

$$T_N = \frac{1}{B_f} \cdot b_{tt} + \frac{i}{n+i} \cdot (r + b_{tt})$$

اما برای ارزیابی دقیقتر این زمان، باید مکان رکورد بعدی را دقیقتر مشخص کرد. شش حالت وجود دارد و هریک از حالات به احتمالی ممکن است بروز کند. پس از درنظر گرفتن احتمالات و ساده‌کردن آنها خواهیم داشت.

$$T_N = \frac{n + i \cdot B_f}{(n+i) \cdot B_f}$$

۳) **زمان درج یک رکورد (T_I) :** برای درج یک رکورد، ابتدا باید بلاکی را که رکورد در آنجا باید درج شود را پیدا کنیم، سپس رکورد موردنظر را در آن بلاک (بعد از خواندن از دیسک) بازنویسی کنیم و رکورد آخر بلاک را در بافر قرار دهیم (که این زمان را بعلت کمی آن صفر درنظر می‌گیریم).
بعد از بازنویسی رکورد جدید، بلاکی را از ناحیه سرریزی خوانده و رکورد موجود در بافر (رکورد آخر بلاکی که عمل درج در آن انجام شده است) را در آن نوشته و دوباره بلاک را در ناحیه سرریزی بازنویسی می‌کنیم. بنابراین خواهیم داشت:

$$\boldsymbol{T_I = T_F + T_{RW} + r + b_{tt} + T_{RW}}$$

T_F : زمان خواندن بلاکی که عمل درج باید در آن انجام گیرد.
T_{RW}: زمان بازنویسی بلاکی که رکورد جدید به آن اضافه شده است (2r)
$r + b_{tt}$: زمان لازم جهت خواندن یک بلاک از ناحیه سرریزی (S را بعلت اینکه ناحیه سرریزی در همان استوانه قرار گرفته است، صفر فرض کرده‌ایم)
T_{RW} : زمان لازم جهت بازنویسی بلاک خوانده شده از ناحیه سرریزی (2r)

$$\boldsymbol{T_I = T_F + 2r + r + b_{tt} + 2r = T_F + 5r + b_{tt}}$$

دقت کنید که در محاسبه روابط بالا، زمان پردازش برروی بلاک در بافر را کوچک درنظر گرفته و قابل صرف‌نظر کردن می‌باشد $C_B \ll 2r$

۴) **زمان بهنگام‌سازی یک رکورد (T_N) :** دربهنگام‌سازی، همانگونه که در ساختارهای قبلی مشاهده کردید، دو حالت پیش می‌آید :
اگر بهنگام‌سازی بر روی فیلدی غیر از کلید اصلی انجام پذیرد، پس رکورد را می‌توان پس از انجام تغییرات دوباره در همان محل قبلی‌اش بازنویسی کرد (بهنگام‌سازی درجا)

$$\boldsymbol{T_{U_{Inplace}} = T_F + T_{RW} = T_F + 2r}$$

ولی درصورتیکه تغییر بر روی فیلد کلید اصلی انجام پذیرد، دیگر نمی‌توان رکورد را در همان جای قبلی‌اش بازنویسی کرد. برای اینکار باید ابتدا رکورد را بازیابی کرده و پس از تنظیم فیلد مربوط به حذف منطقی و دوباره نویسی آن، رکورد تغییر یافته با مقدار کلید جدید را در فایل درج کنیم (بهنگام‌سازی برون‌ازجا)

$$\boldsymbol{T_{U_{Outplace}} = T_F + T_{RW} + T_I = T_F + 2r + T_F + 5r + b_{tt} = 2T_F + 7r + b_{tt}}$$

۵) **زمان حذف یک رکورد (T_D) :** برای حذف در این ساختار از حذف منطقی استفاده می‌کنیم (بعلت زمانگیر بودن حذف فیزیکی) که برای این عمل ابتدا رکورد را واکشی نموده و پس از تنظیم فیلد حذف منطقی، آنرا بازنویسی می‌کنیم. این رکوردهایی که بصورت حذف منطقی، حذف شده‌اند، نوعی حافظه هرز در فایل محسوب می‌شوند که پس از سازماندهی مجدد فایل بطور فیزیکی حذف خواهند شد (حذف حالت خاصی از بهنگام‌سازی درجا می‌باشد).

$$\boldsymbol{T_D = T_F + T_{RW} = T_F + 2r}$$

۶) **زمان خواندن تمام فایل (T_X) :** همانند ساختارهای قبلی، در اینجا نیز می‌توانیم فایل را به دوصورت پی‌درپی و سریال بخوانیم.

برای خواندن فایل بصورت سریال رکوردها، با خواندن اولین رکورد، بقیه از روی زنجیره‌های موجود و توسط عملیات بازیابی رکورد بعدی بدست خواهند آمد. بنابراین خواهیم داشت (i تعداد رکوردهای بخش سرریزی و n تعداد رکوردهای بخش فایل اصلی داده‌ای است)

$$\boldsymbol{T_{X_{ser}} = T_F + (n + i - 1) \cdot T_N}$$

درصورتیکه بخواهیم فایل را بصورت پی‌درپی بخوانیم، ابتدا رکوردهای ناحیه اصلی داده‌ای فایل و سپس رکوردهای بخش سرریزی پشت سرهم خوانده می‌شوند. بنابراین خواهیم داشت:

$$\boldsymbol{T_{X_{seq}} = (n + i) \cdot \frac{R}{t}}$$

۷) **زمان سازماندهی مجدد فایل (T_Y) :** هنگامی که ناحیه سرریزی پرشود و یا طول زنجیره‌ها طولانی گردد، فایل را سازماندهی مجدد می‌کنیم. برای اینکار فایل را باید بصورت سریال خوانده و رکوردهایی که بصورت منطقی حذف شده‌اند را بصورت فیزیکی حذف کنیم (ننوشتن در فایل سازماندهی جدید) و سپس فایل سازماندهی شده را بازنویسی کنیم و بعد از آن شاخص را بر روی فایل جدید بازسازی نمائیم.

$$\boldsymbol{T_Y = T_{X_{seq}} + (n + i - d) \cdot \frac{R}{t} + \frac{S_I}{t}}$$

$T_{X_{seq}}$: زمان خواندن فایل بصورت سریال

n : تعداد کل رکوردهای فایل در لود اولیه

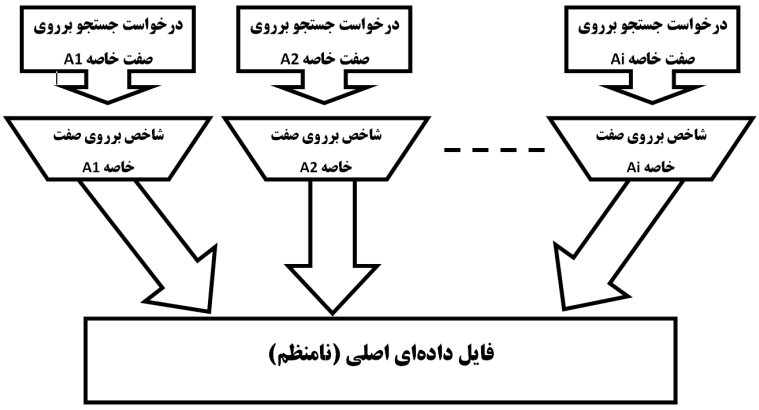
i : تعداد رکوردهای موجود در ناحیه سرریزی

d : تعداد رکوردهای حذف‌شده منطقی

$\frac{S_I}{t}$: زمان بازنویسی بلاک‌های شاخص (بازنویسی شاخص جدید)

فایل چندشاخص (Multi Index) :

فایل چندشاخصی از نظر ساختار همانند فایل ترتیبی شاخص‌دار است با این تفاوت که در این ساختار روی تعدادی و یا حتی تمامی فیلدهای رکورد می‌توان شاخص داشت. این روش بعلت برطرف کردن پدیده عدم تقارو در ساختار قبلی پیشنهاد گردید. چرا که در روش قبلی دسترسی به یک رکورد براساس فیلد خاصی (کلید اصلی فایل) انجام می‌گرفت و از طرفی در این ساختار مشکل سرریزی بصورتی که در ساختار ترتیبی شاخص‌دار وجود داشت، وجود ندارد و خود ساختار شاخص نیز وضعیتی پویا دارد و همگام با تغییرات فایل داده‌ای، بهنگام می‌شود.



فایل اصلی داده‌ای در این ساختار می‌تواند حتی بصورت بی‌نظم‌ترین شکلش (پایل) وجود داشته باشد و در این ساختار چندین فایل شاخص (حتی می‌توان به تعداد فیلدهای فایلی شاخص داشت) وجود دارد و بدین ترتیب کاربر می‌تواند بر اساس هریک از صفات خاصه (فیلدهای رکورد) جستجو را بر روی فایل انجام دهد.

استراتژی دستیابی به رکورد در این ساختار توسط شاخص‌ها مشخص می‌گردد. چراکه فایل داده‌ای بی‌نظم بوده و فاقد هرگونه استراتژی دستیابی است. لذا آنچه در این ساختار بسیار مهم است، ساختار درونی فایل‌های شاخص می‌باشد که باید در پیاده‌سازی انها از روش‌هایی استفاده کرد که رفتاری پویا، انعطاف‌پذیر باشند و عملیات بر روی فایل اصلی را تسریع کنند، همانند ساختارهایی نظیر B-tree که در ادامه به تشریح آنها خواهیم پرداخت.

نکته : درصورتیکه بر روی تمامی صفات، شاخص ایجاد کرده باشیم، فایل را وارون (Invert File) می‌گوییم

همانگونه که گفته شد تنها استراتژی دسترسی به فایل چندشاخصی (باتوجه به اینکه فایل از نوع بی‌نظم است) توسط شاخص انجام می‌پذیرد. با توجه به اینکه در تعریف فایل چند شاخصی، امکان ایجاد شاخص بر روی هرکدام از صفات خاصه وجود دارد، ما دارای تعدادی فایل شاخص هستیم که برای دسترسی به رکوردها بکار برده می‌شوند. ساختار شاخص باید بگونه‌ای باشد که همگام به عملیات بر روی فایل اصلی (درج، حذف یا بهنگام‌سازی) بطور پویا قوبلیت بهنگام شدن را داشته باشد. ساختار مناسبی که در فایل‌های چندشاخصی استفاده می‌شود مدل B-tree است.

قبل از تشریح درختان B، ابتدا به معرفی درخت‌هاس جستجوی mمی‌پردازیم.

درخت‌های جستجوی m طرفه

همانطور که می‌دانید هدف اصلی شاخص‌بندی این است که تعداد دستیابی‌ها به دیسک کاهش یابد و درخت‌ها ساختارهای مناسبی برای این منظور هستند.

یک درخت جستجوی m طرفه، یا تهی است یا خواص و شرایط زیر را دارا می‌باشد:

۱- درخت جستجوی m طرفه T درختی است که درجه تمام گره‌های آن کوچکتر یا مساوی m است

۲- هرگره درخت شامل صفات زیر است:

P_0	K_1	P_1	K_2	P_2	- - - -	K_n	P_n

که در آن

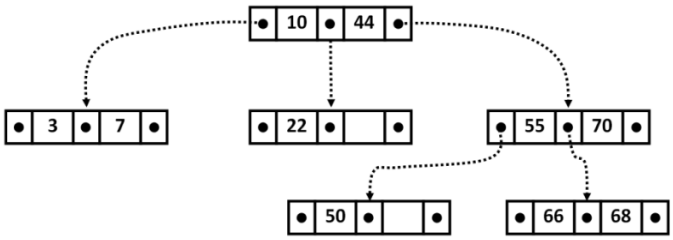
- $1 \le n \le m$
- k_i مقادیر کلید در گره هستند $(1 \le i \le n)$
- P_i اشاره‌گره‌هایی به زیردرخت‌های T هستند $(1 \le i \le n)$

۳- به ازای هر $1 \le i \le n$ خواهیم داشت $k_i \le k_{i+1}$

۴- مقادیر موجود در زیر درختی که P_i به آن اشاره می‌کند، کوچکتر از مقادیر کلید k_{i+1} هستند $(1 \le i \le n)$

۵- مقادیر کلید موجود در زیر درختی که P_n به آن اشاره می‌کند، بزرگتر از k_n هستند.

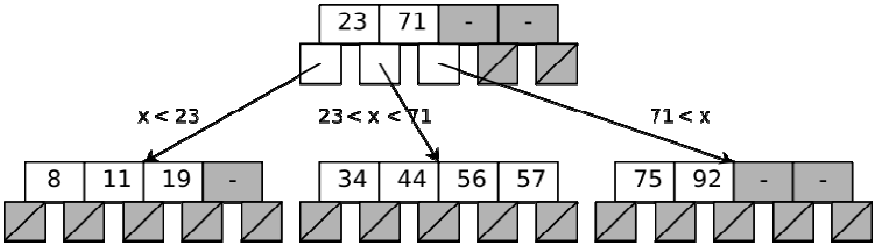
۶- تمام زیردرخت‌هایی که P_i $(1 \le i \le n)$ به آن اشاره می‌کند، درخت‌های جستجوی m طرفه هستند.



دو نوع درخت جستجوی m طرفه‌ی معروف عبارتند از درخت $B-tree$ و درخت B^+-tree می‌باشند.

جستجوی در یک درخت جستجوی m طرفه

فرض کنید خواسته باشیم درخت جستجوی m طرفه‌ای مانند T را برای کلید x مورد جستجو قرار دهیم و همچنین فرض کنید که T بر روی دیسک قرار داشته باشد. در ابتدا شروع به بازیابی گره ریشه درخت T از دیسک خواهیم نمود. با جستجو کلیدها در ریشه، i را بگونه‌ای تعیین می‌کنیم که $k_i \le x \le k_{i+1}$ باشد. اگر $x = k_i$ باشد، آنگاه جستجو به اتمام می‌رسد، درغیراینصورت براساس تعریف درختان جستجوی m طرفه، چنین استنباط می‌گردد که که x در زیر درختی است که به آن اشاره می‌کند و این جستجو در زیر درختها را تا آنجا ادامه می‌دهیم که کلید x پیدا شود.



بطور کلی می‌توان گفت که تمام درخت‌ها مناسب شاخص نیستند، بلکه درخت‌هایی برای این کار مناسب هستند که

- گره‌های آن‌ها تقریباً پر باشند
- متوازن باشند

درختی که این دو ویژگی را داشته باشد، کوتاه است. درخت‌های $B-tree$ و B^+-tree دارای چنین ویژگی‌هایی هستند که در ادامه به بررسی آنها خواهیم پرداخت

درخت B $(B-tree)$

درخت B در واقع یک درخت جستجوی m طرفه است. مشکل درخت جستجوی m طرفه عدم توازن آن بود که درخت B این مشکل را مرتفع نموده است.

هر گره درخت B یک رکورد شاخص است و هرکدام از این رکوردها دارای تعدادی از جفت‌های کلید-آدرس هستند که مرتبه درخت نام دارد. معمولاً رکوردها دارای حداقلی از جفت‌های کلید-آدرس هستند که برابر نصف مرتبه درخت است. مثلاً اگر درخت B با مرتبه 50 داشته باشیم، هر رکورد حداقل دارای 25 کلید و حداکثر 50 کلید است. ریشه استثناء است و می‌تواند حداقل 2 فرزند داشته باشد.

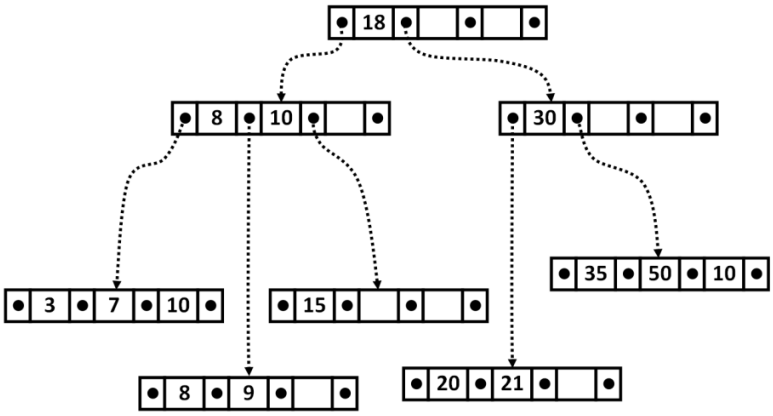
تعریف درخت B : درخت B مرتبه m ، درخت جستجوی m طرفه‌ای است که یا خالی باشد یا خواص ریز را برآورده سازد

۱- گره ریشه حداقل دارای دو فرزند باشد.

۲- تمام گره‌ها غیر از ریشه حداقل دارای $\left\lceil \frac{m}{2} \right\rceil$ فرزند باشند (و یا به مفهوم دیگر حداقل $\left\lceil \frac{m}{2} \right\rceil - 1$ کلید دارد)

۳- تمام گره‌های برگ در سطح مشابه و یکسانی قرار دارند.

۴- در سطح برگ‌ها، تمام کلیدها بصورت مرتب قرار دارند.

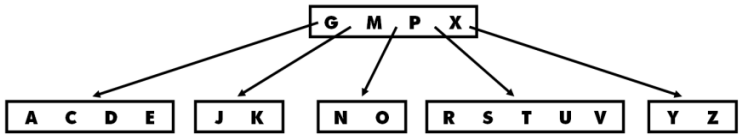


درج در درخت B :

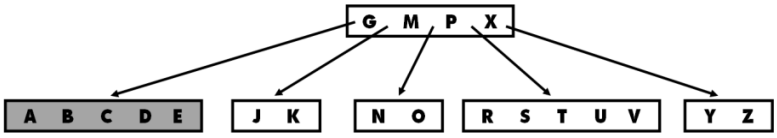
برای درج یک رکورد در درخت B، ابتدا باید درخت را جستجو کنیم که آیا کلید موردنظر ما قبلاً در درخت وجود داشته است یا خیر؟ درصورتیکه وجود نداشته باشد، این جستجو ما را به گره برگی خواهد رساند که رکورد جدید باید در آنجا درج شود. در صورتیکه بعد برگ فضا برای اضافه کردن کلید جدید داشته باشد با مشکلی مواجه نخواهیم بود. ولی اگر برگ پُر باشد، مجبوریم که رکوردهای موجود در برگ را به دوقسمت تقسیم کنیم (البته بعد از اضافه نمودن کلید جدید درمحل منطقی صحیحاش). در این

تقسیم دقت داشته باشید که هر دو گره جدید دارای $\left\lceil \frac{m}{2} \right\rceil$ کلید خواهند بود و کلید وسط را به گره پدر منتقل کرده و اشـاره-گرهای گره پدر به این دو گره جدید را تنظیم می‌کنیم. حال درصورتیکه گره پدر پُر باشد، دوباره همین روال را بـرای ان انجام می‌دهیم. این روال تا آنجا که نیازی به شکستن گره وجود نداشته باشد ادامه خواهیم داد (این امکان وجود دارد که این روال تا سطح ریشه پیش رفته و حتی خود ریشه نیز شکسته شود).

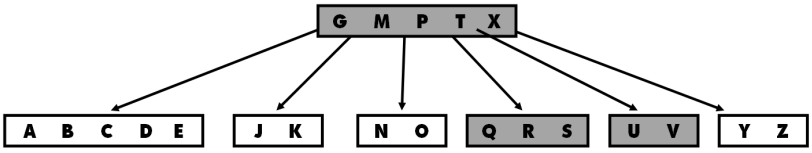
فرض کنید که درخت B (m=5) زیر مفروض است. تغییرات درخت را در هر مرحله بعد از اضافه کردن یـک کلیـد جدیـد مطابق الگوریتم گفته شده را مشاهده می‌کنید.



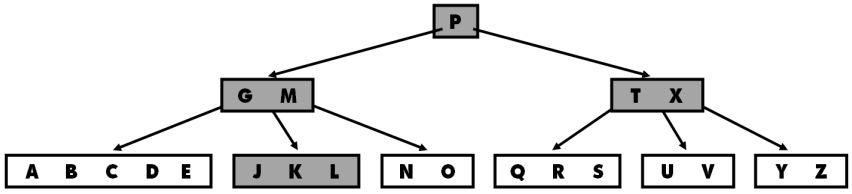
اضافه کردن کلید B



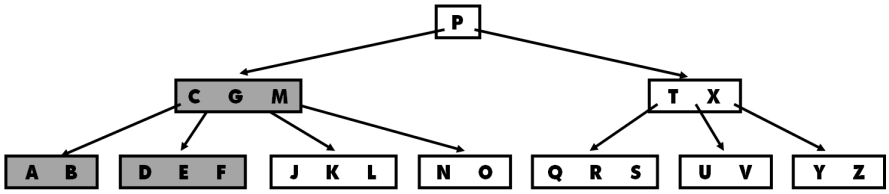
اضافه کردن کلید Q



اضافه کردن کلید L



اضافه کردن کلید F



مثال۱ : درخت B از مرتبه ۳ را با استفاده از مقادیر زیر که از چپ به راست وارد می‌شوند، ایجاد نمائید

10 , 20 , 30 , 40 , 50 , 60 , 70 , 80 , 90

مثال۲ : درخت B مرتبه ۵ با استفاده از مقادیر زیر که از چپ به راست وارد می‌شوند را بسازید.
C , N , G , A , H , E , K , Q , M , F , W , L , T , Z , D , P , R , X , Y , S

حذف از درخت B :

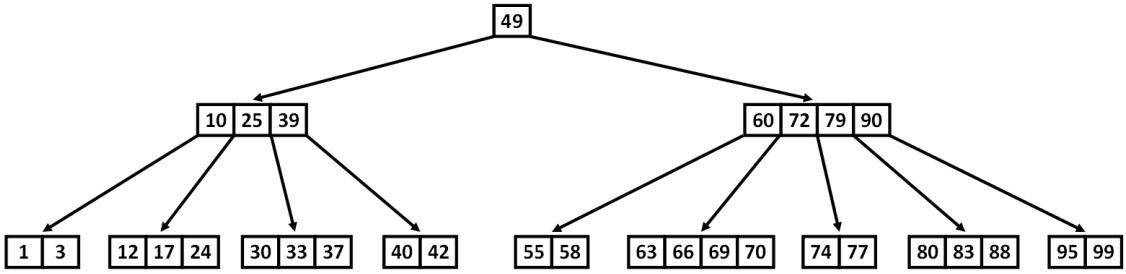
در حذف، بخلاف درج در درخت B حالات مختلفی پیش خواهد آمد که هر حالت قوانین خاص خود را دارد ولی در همه حالات باید خواص درخت B حفظ گردد یعنی :

- ریشه حداقل باید دارای یک مقدار کلید باشد
- بقیه گره‌ها حداقل باید $\left\lceil \frac{m}{2} \right\rceil - 1$ کلید داشته باشند.

حالت اول : حذف کلید از گره‌های برگ

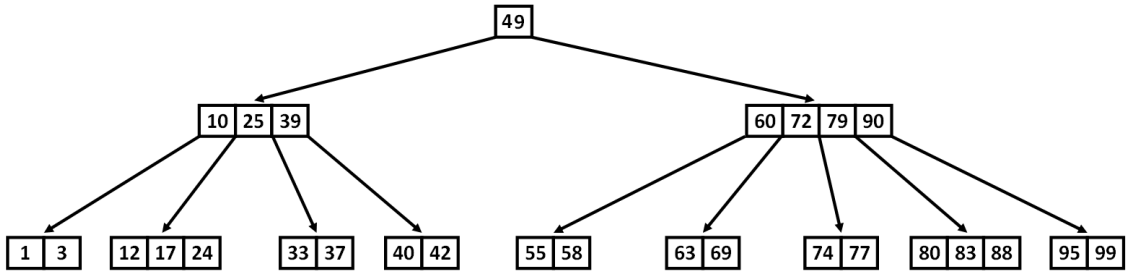
درصورتیکه با حذف کلید، تعداد کلیدهای باقیمانده در برگ موردنظر بیشتر یا مساوی $\left\lceil \frac{m}{2} \right\rceil - 1$ باقی بماند (راحترین حالت حذف) کلید را حذف و بقیه درخت تغییر نخواهد کرد.

به عنوان مثال، در درخت B زیر از مرتبه 5 را در نظر می‌گیریم.



در درخت فوق هر گره حداقل باید دارای $\left\lceil \frac{m}{2} \right\rceil - 1 = \left\lceil \frac{5}{2} \right\rceil - 1 = 2$ کلید داشته باشد.

درصورتیکه بخواهیم کلیدهای 33 و 66 و 70 را حذف کنیم، شرط دوم که هر گره حداقل بعد از حذف باید دارای 2 کلید باشد نقض نخواهد شد بنابراین درخت فوق بعد از حذف گره‌ها به شکل زیر در خواهد آمد.

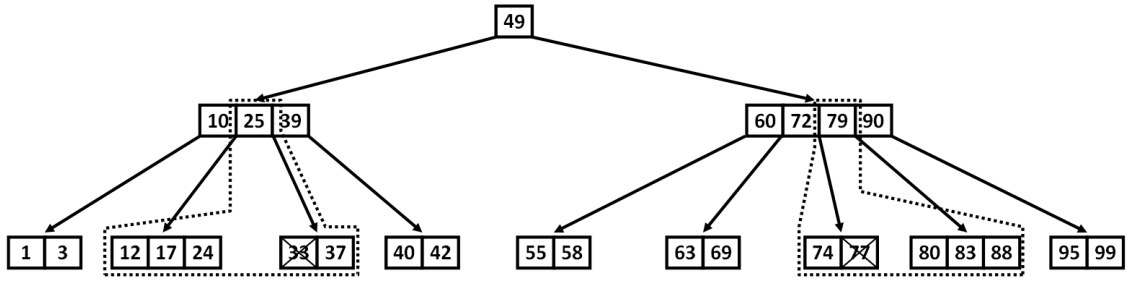


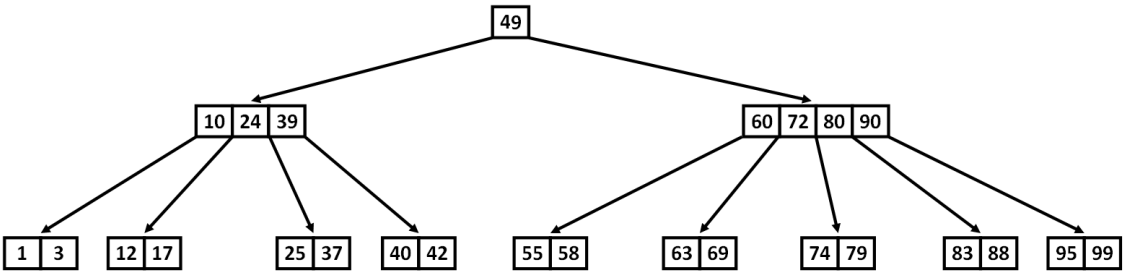
ولی اگر بعد از حذف کیلدهای باقیمانده در گره کمتر از $\left\lceil \frac{m}{2} \right\rceil - 1$ باشد آنگاه باید کلیدی را از گره همزاد آن به این گره منتقل کنیم و یا آنرا با گره‌های همزاد آن ترکیب کنیم. سه وضعیت ممکن است رخ دهد

۱) نزدیک‌ترین همزاد راست بیش از $\left\lceil \frac{m}{2} \right\rceil - 1$ کلید دارد.
۲) نزدیک‌ترین همزاد چپ بیش از $\left\lceil \frac{m}{2} \right\rceil - 1$ کلید دارد.
۳) هیچ کدام از نزدیک‌ترین همزاد چپ یا راست بیش از $\left\lceil \frac{m}{2} \right\rceil - 1$ کلید ندارند.

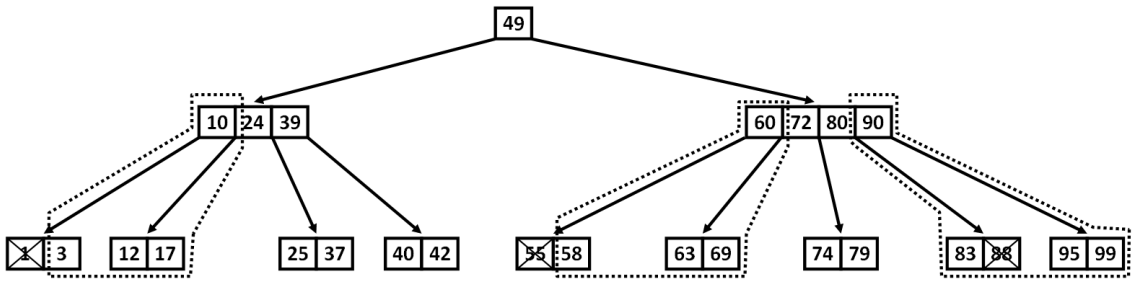
در حالت‌های ۱ و ۲ یکی از کلیدها را از همزادی با بیش از $\left\lceil\frac{m}{2}\right\rceil - 1$ کلید به گره پدر انتقال می‌دهیم و سپس یک کلید از گره پدر را به گره برگی که کلیدی را از آن حذف کرده‌ایم؛ منتقل می‌کنیم.

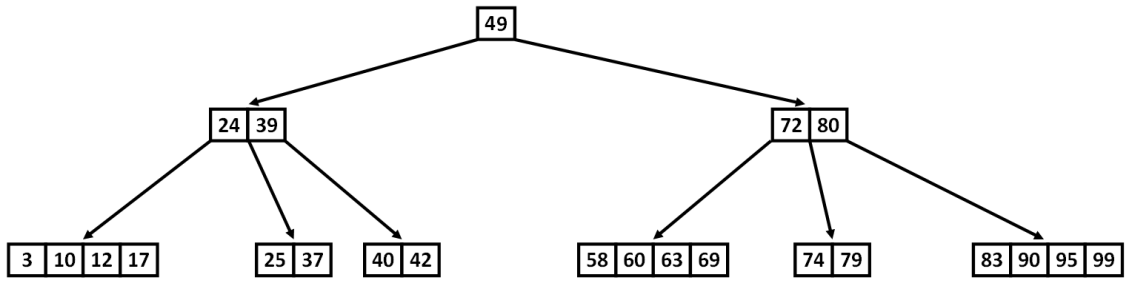
فرض کنید که در مثال فوق، می‌خواهیم گره 33 و 77 را حذف کنیم، خواهیم داشت





در حالت ۳ که هیچ همزادی بیش از $\left\lceil\frac{m}{2}\right\rceil - 1$ کلید ندارد، گره برگی را که کلیدی از آن حذف شده است را با یکی از همزادهایش ترکیب می‌کنیم و کلیدی را از گره پدر به این گره جدید (ترکیب شده برگی که کلیدی از آن حذف شده و همزادش) انتقال می‌دهیم. اگر تعداد کلیدهای گره ریشه کمتر از $\left\lceil\frac{m}{2}\right\rceil - 1$ باشد، روند فوق را برای این گره تکرار می‌کنیم. در صورتیکه بترتیب کلیدهای 88 و 1 و 55 را حذف کنیم خواهیم داشت.





حالت دوم : حذف کلید از گره‌های غیربرگ

اگر کلیدی که باید حذف شود، در برگ نباشد، کلید پیشین یا بعدی آن در برگ وجود خواهد داشت. بنابراین با تبدیل ساده‌ای می‌توان حذف کلید از گره غیر برگ را به حذف کلید از گره برگ تبدیل کرد. فرض کنید k_i کلیدی در گره P_i باشد که باید حذف شود. فرض کنید گره P به صورت زیر باشد.

K_1	K_2	---	K_i	---	K_n

P_0	P_1	P_2	----	P_i	---	P_n

چونP گره برگ نیست بنابراین $P_i\ != null$ است. حال به جای کلید k_i که می‌خواهیم حذف کنیم، می‌توانیم کوچک‌ترین مقدار زیر درختی که P_i به آن اشاره می‌کند را قرار دهیم. این مقدار که در سمت چپ ترین برگ زیر درخت P_i وجود دارد را جایگزین k_i کرده و سپس آنرا از برگ همانند حذف یک کلید از برگ، حذف می‌کنیم.

بررسی ویژگی‌های درخت B

بعد از بررسی اعمال ممکن بر روی درخت B، حال بهش شرح بعضی از ویژگی‌های این درخت می‌پردازیم.

- درخت B همواره از نظر ارتفاع متوازن است.
- درجه درخت B مرتبه m، برابر با m است. یعنی حداکثر تعداد فرزندان یک گره برابر m است.
- در درخت B با مرتبه m با ارتفاع h داریم

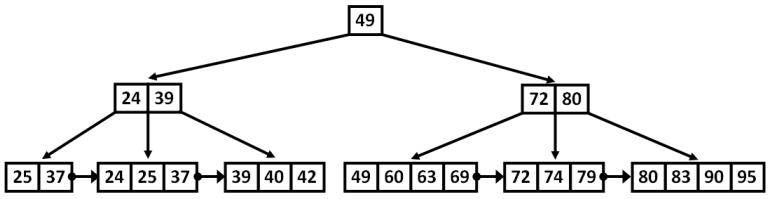
حداکثر تعداد گره‌های ممکن $=\sum_{i=0}^{h-1} m^i = \frac{m^h-1}{m-1}$

- حداکثر تعداد مقادیر کلید در هر گره درخت B مرتبه m، برابر با (m-1) است.
- حداکثر تعداد مقادیر کلید در درخت B مرتبه m برابر است با

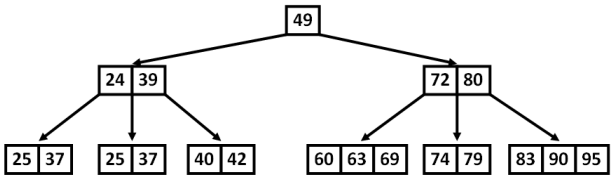
- گره ریشه حداقل دو فرزند دارد
- تمام گره‌ها بجز گره ریشه حداقل می‌تواند $\left\lceil \frac{m}{2} \right\rceil$ فرزند داشته باشد.
- حداقل تعداد مقادیر کلیدها در گره ریشه برابر با یک است (اگر درخت خالی نباشد)
- حداقل تعداد مقادیر کلید در هر گره غیر رشته برابر با $\left\lceil \frac{m}{2} \right\rceil - 1$ است.
- هر مقدار کلید فقط یک بار در درخت ظاهر می‌شود.

درخت B^+

درخت B^+ از تغییر کوچکی در درخت B بدست می‌آید. تفاوت اصلی در بین این دو نوع درخت، تکرار تمامی مقادیر کلید در سطح برگ در درخت B^+ است.



B^+-tree



B-tree

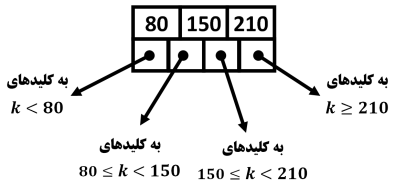
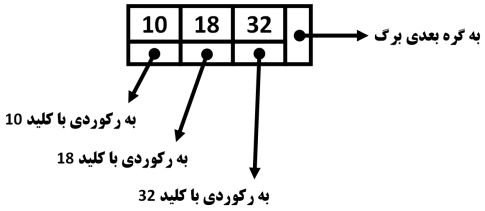
علاوه بر این تفاوت، در سطح برگ بین گره‌های برگ پیوندی وجود دارد که می‌توان کلیدهای سطح برگ را پشت‌سرهم و بودن نیاز به پیمایش درخت، خواند (البته این خواندن به صورت صعودی یا نزولی است و در آن تمام کلیدها شرکت دارند).

در درخت B^+ آدرس رکورد داده‌ها در فایل اصلی فقط در سطح برگها ذخیره می‌شوند و در گره‌های داخلی درخت B^+ فقط کلیدها ذخیره می‌شوند. در واقع گره‌های داخلی فقط برای انجام عمل جستجو و رسیدن به گره برگ مناسب مورد استفاده قرار می‌گیرند. اگر داده موردنظر کوچکتر از کلیدی در یک گره داخلی باشد، اشاره‌گر سمت چپ و به تبع آن وارد زیر درخت سمت چپ می‌شویم ولی اگر بزرگتر باشد اشاره‌گر سمت راست و درنتیجه وارد زیردرخت سمت راست خواهیم شد. ولی در عمل جستجو همانگونه که قبل گفتیم، از کلید سمت چپ گره شروع می‌کنیم و آنقدر جلو می‌رویم تا به کلیدی در گره برسیم که از داده مورد جستجو بزرگتر باشد، آنگاه اشاره‌گره سمت چپ کلید را دنبال می‌کنیم و اگر هیچ کدام از کلیدهای موجود در گره از داده مورد جستجو کوچکتر نبودند، آخرین اشاره‌گر گره (سمت راست‌ترین) را دنبال می‌کنیم.

بعضی از ویژگی‌های درخت B^+ عبارتند از:

- اشاره‌گرهای مربوط به رکورد داده‌ها (یا خود رکورد داده‌ها) فقط در برگ‌ها ذخیره می‌شوند.
- گره‌های داخلی (غیربرگ) را گره‌های شاخص، و گره‌های برگ را گره‌های داده‌ها می‌نامند.
- گره‌های داخلی فقط شامل کلیدها و اشاره‌گرها درخت هستند
  - می‌توان اشاره‌گرهای بیشتری را در گره‌های داخلی ذخیره کرد
  - به دلیل سطوح کمتر درخت، جستجو با سرعت بیشتری انجام می‌شود.
- فضایی به‌خاطر اشاره‌گرهای تهی در گره‌های برگ به‌هدر نمی‌رود

ساختار گره‌های داخلی درخت B^+ دقیقاً مثل ساختار گره‌های درخت B است.



ساختار گره‌های برگ درخت B^+ به‌صورت زیر است