به نام خدا

محمدمهدی آقاجانی

تمرین چهارم سیستم عامل

استاد طاهری جوان

بخش بن بست

تمرین اول : در مورد شکل زیر به سوالها پاسخ دهید

بله بن بست رخ داده است .فرض کنیم چهار را ه ها منبع هستند و ماشین ها فرآیند . در این صورت دو ماشین دقیقا همزمان نمیتوانند از آن عبور کنند.در این شکل دو به دو ناسازگاری طبق آنچه که توضیح دادیم وجود دارد.هم چنین ماشین ها وقتی در چهار راه منتظر میمانند در چهار راه توقف میکنند بنا براین منبع مورد نظر را رها نمیکنند ( در این حالت اگر امکان دنده عقب وجود داشته باشد این شرط ارضا نمیشود ) . هم چنین ماشینی اگر در چهار راه باشد کسی به زور نمیتواند چهار راه را از او پس بگیرد همچنین انتظار چرخشی نیز همانطور که از شکل پیدا ست وجود دارد پس هر چهار شرط کافمن بر قرار هستند و احتمال بن بست در اینجا وجود دارد.

راه کار های مقابله :

همانطور که در بالا اشاره شد میتوان امکان دنده عقب را برای ماشین ها در نظر گرفت که در این صورت دیگر بن بست رخ نمیدهد . یا اینکه در چراغ راهنمایی تا وقتی یک طرف چهار راه پشت سر هم ماشین می آید چراغ های دیگر چهار راه قرمز بمانند و اجازه عبور نیابند.

تمرین دوم : فرض کنید برای مقابله با بن بست، شش راه کار زیر وجود دارد. سوال های الف و ب را کاملا تحلیل کنید

الف )

1. در این حالت رتبه بندی به صورت زیر خواهد بود( اولین رتبه ، بدترین است ) :

* کشف بن بست و به عقب برگرداندن فرآیند ها
* کشف بن بست و از بین بردن فرآیند های دخیل
* الگوریتم بانکدار
* شروع مجدد فرآیند
* دادن شماره ترتیبی
* رزرو تمام منابع

علت اینکه کشف بن بست سنگین است این است که باید هر چند وقت یکبار الگوریتمی را اجرا کند تا بن بست را کشف کند و بعد الگوریتمی را اجرا کند تا بن بست را بر طرف کند . همچنین الگوریتم بانکدار نیز خود ناز به محاسبات ئیش بینی دارد . شروع مجدد فرآیند نیز ممکن است مدام چند فرآیند را نصفه اجرا کند و همین بهره وری را کاهش دهد. اما رزرو منابع در ابتدای اجرا شاید منابع را بیش از حد اشغال کند اما پردازنده را اشغال نمیکند.

1. رتبه بندی به صورت زیر است ( اولین رتبه ، بدترین است ) :

* رزرو تمام منابع در شروع
* دادن شماره ترتیبی
* الگوریتم بانکدار
* شروع مجدد فرآیند
* کشف بن بست

الگوریتم هایی که منابع را بیشتر اشغال میکنند در واقع جلوی بیشتر موازی اجرا شدن را میگیرند در نتیجه الگوریتمی مثل رزرو تمامی منابع در شروع خیلی مانع موازی اجرا شدن میشود.

ب)خیر!!

تمرین سوم : راه کار زیر را برای مساله فیلسوف های خورنده، هم از نظر بن بست و هم از نظر گرسنگی کاملا تحلیل کنید

احتمال گرسنگی وجود دارد بدین صورت که اگر یکی از فیلسوف ها موفق شود که دو چنگال در دست بگیرد و شروع به خوردن کند سپس به فیلسوفان دیگر هر یک، یک نوبت برسد در صورت در صورت اتمام خوردن به یک فیلسوف دیگر نوبت میرسد و اگر این فیلسوف هم خوردنش تمام شود دوباره ممکن است به همان فیلسوف اول که غذاخورده بود نوبت برسد پس احتمال گرسنگی وجود دارد. اما بن بست رخ نمیدهد زیرا یکی از شرط های کافمن را ارضا نمیکند و آن این است که یک فیلسوف اگر نتواند غذا بخورد آن قاشقی که در دستانش است را بر روی میز بر میگرداند پس در واقع هر فرآیند در حالت انتظار منابع در اختیار خود را داوطلبانه آزاد میکند.

تمرین چهارم : به نظر شما آیا یک سیستم می تواند تشخیص دهد برخی از فرآیندها دچار گرسنگی شده اند؟ اگر پاسخ شما مثبت است، راهکار خود را تشریح کنید. اگر پاسخ منفی است، سیستم چگونه باید با مساله گرسنگی کنار بیاید؟

این کار امکان پذیر است اما هزینه بر است . مثلا میتواند یک سقفی از مدت زمان سرویس ندادن به یک فرآیند را تعیین نمود سپس هر فرآیندی که در آن سقف سرویس نگرفته بود یعنی دچار گرسنگی شده است بنا براین باید الگوریتم های مناسبی در سیستم انتخاب کنیم تا با گرسنگی مواجه نشویم

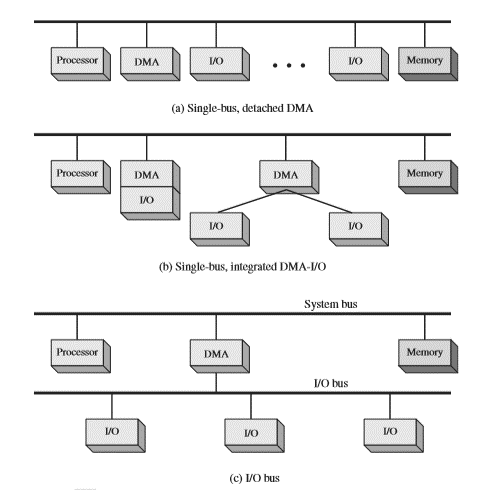
تمرین پنجم : به نظر شما ایده اصلیِ پشت الگوریتم بانکدار در زندگی روزمره هم کاربرد دارد؟ دربارۀ سودمندی )یا عدم سودمندیِ( این ایده در دنیای واقعی کمی بحث کنید و مثال بزنید

بله . میتواند سودمند باشد. وقتی انسان بدهکار است و همزمان از برخی نیز طلبکار است باید نسبت به اتفاقات مالی اطراف خود هشیارانه عمل کند مثلا در این شرایط قرض کردن از کسی میتواند شمشیر دو لبه باشد همچنین قرض دادن به کسی هم میتواند شمشیر دو لبه باشد و با یک تصمیم گیری در هر یک از شرایط بالا موجبات ضرر مالی را فراهم آورد

بخش حافظه جانبی و سیستم فایل

تمرین :1دربارۀ پیکربندی های مختلف DMAو گذرگاه سیستم تحقیق کنید.

پردازنده برای دسترسی به کنترل کننده های دستگاه ها نیاز به آدرس آن­ها دارد(چه روی حافظه نگاشت شده باشند چه خیر).کنترل کننده میتواند هر بار با نرخ یک بایت به کنترل کننده دستگاه ها درخواست بدهد که این عمل برای وسیله ای چون دیسک اصلا به صرفه نیست.پس به جای آن از DMA استفاده می­کنیم.البته برای استفاده از آن باید کنترل­کننده DMA را نیز داشته باشیم.کنترل کننده DMA می­تواند مستقل از پردازنده و بدون توجه به مکان فیزیکی اش به گذرگاه سیستم دسترسی داشته باشد.این کنترل کننده شامل چندین ثبات است(همچون ثبات آدرس حافظه، یک ثبات شمارشگر بایت ،پورت های I/O مورد استفاده جهت انتقال ، واحد انتقال و ...) که پردازنده می­تواند روی آن بنویسد یا از روی آن بخواند.در ابتدا DMA را پردازنده با استفاده از ثبات هایش برنامه ریزی می­کند تا پردازنده بداند چه چیزی را باید به کجا منتقل نماید.همچنین فرمانی را به کنترل گممده دیسک صادر می­کند تا داده ها را از بافرهای داخلی دیسک خوانده و صحیت آن را بررسی کند.هنگامی مه داده­های معتبر درون بافر کنترل کننده ریسک قرار گیرند عملیات DMA می­تواند آغاز شود.کنترل کننده DMA انتقال را با صدور یک درخواست خواندن از طریق گذرگاه به کنترل کننده دیسک آغاز می­کند(مرحله \*).این درخواست با خواندن مانند سایر درخواست­های خواندن بوده و کنترل کننده دیسک نمی­داند که آیا درخواست از پردازنده آمده است و یا از کنترل کننده DMA .معمولا آدرس خانه­ای از حافظه که باید درون آن نوشته شود برروی خطوط آدرس گذرگاه قراردارد، به طوری که وقتی کنترل کننده دیسک، کلمه بعدی را از بافر درونی اش واکشی می­کند می­داند کجا باید آن را بنویسد.نوشتن در حافظه، یک سیکل استاندارد دیگر گذرگاه می­باشد(مرحله \*\*).هنگامی که نوشتن کامل می­شود کنترل کننده دیسک یک سیگنال تصدیق به کنترل کننده DMA از طریق گذرگاه می­فرستد(مرحله \*\*\*).سپس کنترل کننده DMA آدرس حافظه مورد استفاده از افزایش می­دهد و شمارشگر بایت را کاهش می­­دهد.در صورتی که شمارشگر بایت هنوز بیشتر از صفر باشد، مراحل دو تا چهار تا صفر شدن آن تکرار می­شوند.در این مرحله یک وقفه ایجاد می­کند.هنگامی که سیستم عامل فعال می­شود مجبور نیست که یک بلوک را در حافظه کپی نماید، چون آن بلوک قبلا در حافظه قرار داده شده است. رفتار DMA به چندین روش متفاوت میتواند پیکربندی شود. برخی از امکانات در شکل زیر آمده است. در اولین مثال تمام پودمان ها از یک گذرگاه سیستم مشترک استفاده میکنند. پودمان DMA به عنوان جانشین processor ایفای نقش میکند، از ورودی خروجی برنامه ریزی شده برای تعویض داده بین حافظه و پودمان I/O. چنین پیکربندی ای با اینکه ارزان است اما بهینه نیست چراکه هر انتقال داده به دو چرخه گذرگاه نیاز دارد.در روش دوم میتوان، تعداد چرخه های مورد نیاز گذرگاه را با یک یکپارچه کردن پودمان DMA و I/O کوتاه کرد. بدین معنا که یک مسیر بین پودمان DMA و یک یا تعداد بیشتری پودمان I/O وجود دارد که گذرگاه داده را شامل نمیشود. درواقع منطق DMA ممکن است بخشی از پودمان I/O باشد و یا یک پودمان مجزا باشد که کنترل یک یا چند پودمان I/O را برعهده دارد. این مفهوم میتواند با اتصال پودمان I/O و DMA از طریق گذرگاه I/O، کمی پیشرفته تر شود.



تمرین 2 : دربارۀ طرح RAIDبرای دیسک های چندگانه تحقیق کنید. تفاوت سطوح مختلف RAIDرا بررسی کنید.

گرداننده­های دیسک هر روز ارزانتر و کوچک­تر می­شوند، لذا امروزه می­توان تعداد زیادی از دیسک­ها را به سیستم کامپیوتری وصل کرد . اگر تعداد دیسک­های موجود در سیستم زیاد باشد و این دیسک­ها به طور موازی کار کنند، نرخ نوشتن و خواندن داده-ها بهبود می­یابد. علاوه براین، منجر به قابلیت اعتماد ذخیره­سازی داده می­شود، زیرا اطلاعات اضافی را می­توان بر روی چندین دیسک ذخیره کرد . بنابراین، خراب شدن یک دیسک منجر به ازبین رفتن داده­ها نمی­شود . تکنیک­های گوناگونی از سازماندهی دیسک که آرایه­ی دیسک­های مستقل با افزونگی (RAID) نامیده می­شود، برای کارایی و قابلیت اعتماد به کار می­روند.

در گذشته، RAIDها شامل دیسک­های کوچک و ارزان بودند، که جایگزین کارآمدی برای دیسک­های بزرگ و گران محسوب می­شدند . امروزه، RAIDها به جای دلایل اقتصادی، به دلیل قابلیت اعتماد بالا و نرخ بیشتر انتقال داده­ها به کار گرفته می­شوند . در گذشته، I در RAID از Inexpensive گرفته شد که به معنای ارزان است، ولی امروزه I در RAID از Independent گرفته شد که به معنای مستقل است.

**سطوح RAID**

قابلیت اعتماد دیسک­های آینه­ای، بالاست، ولی گران است. نواربندی، منجر به نرخ بالای انتقال داده­ها می ­شود، ولی قابلیت اعتماد را بهبود نمی­بخشد. طرح­های زیادی برای تدارک افزونگی با هزینه­ی کم تدارک دیده شدند. این طرح ها از ایده­ی نواربندی دیسک همراه با بیت­های توازن استفاده می­کنند. این طرح­ها از نظر توازن بين هزینه و کارایی باهم فرق می­کنند و به سطوحی بهنام سطوح RAID دسته بندی می­شوند.

اکنون سطوح مختلف را شرح می­دهیم. در شکل زیر، P نشان دهنده­ی بیت تصحیح خطا، و C نشان دهنده­ی کپی دوم داده­ها است. در تمام مواردی که در شکل آمده است، چهار دیسک برای ذخیره­ی داده­های ارزشمند و بقیه برای ذخیره­ی داده­های افزونگی به کار می­روند. این داده­های اضافی برای ترمیم خرابی مفید هستند.

RAID سطح صفر: این سطح به آرایه­های دیسک با نواربندی سطح بلوک­ها مربوط می­شود، اما فاقد هرگونه افزونگی است (مثل آینه­ای و بیت­های توازن). قسمت الف شکل، آرایه­ای به اندازه­ی 4 را نشان می دهد.

RAID سطح۱: این سطح به آیینه­ای کردن دیسک مربوط می­شود. قسمت ب شکل، یک سازماندهی آینه­ای را نشان می­دهد که چهار دیسک، داده­های با ارزش را ذخیره می­کنند.

RAID سطح 2: این سطح به نام سازمان کد تصحیح خطای سبک حافظه (ECC) خواند می­شود. سیستم­های حافظه، با استفاده از بیت­های توازن، کشف خطا را پیاده­سازی می­کنند. هر بایت در سیستم حافظه، ممکن است دارای یک بیت توازن باشد که مشخص می­کند تعداد بیت­های یک بایت که مقدار آنها یکسان است، زوج است (توازن = صفر) یا فرد (توازن = ۱). اگر یکی از بيت­های موجود در بایت خراب شود (۱ به صفر تبدیل شود یا بر عکس)، توازن بایت تغییر می­کند و با توازن ذخیره شده تطبیق نمی­کند. متشابهاً، اگر بیت توازن ذخیره شده خراب شود، با توازن محاسبه شده تطبیق نمی­کند . بنابراین، تمام خطاهای تک بیتی توسط سیستم حافظه تشخیص داده می­شود. طرح­های تصحیح خطا دو یا چند بیت اضافی را ذخیره می­کنند و در صورتی که یک بیت خراب شود، داده­ها را می­توانند بازسازی کنند. ایده­ی ECC مستقیماً می­تواند از طریق نواربندی بایت­ها در دیسک­ها، در آرایه­های دیسک مورد استفاده قرار گیرد. به عنوان مثال، اولین بیت هر بایت می­تواند در دیسک ۱ ذخیره شود، دومی در دیسک ۲ و غیره. این روند ادامه می­یابد تا بیت هشتم در دیسک ۸ قرار گیرد و بیت­های تصحیح خطا در دیسک­های دیگری ذخیره شوند. این طرح در شکل آمده است که در آن، دیسک­هایی با برچسب P، بیت­های تصحیح ذخیره می­کنند. اگر یکی از دیسک­ها خراب شود، بیت­های باقیمانده­ی آن بایت و بیت­های ­تصحیح خطای مربوط به آن می تواند از دیسک­های دیگر خوانده شوند و برای بازسازی داده خراب شده به­کار گرفته شود. قسمت پ شکل آرایه­ای به اندازه ی ۴ را نشان می­دهد. توجه کنید که RAID سطح ۲، برای چهار دیسک از داده­ها به یک دیسک اضافه نیاز دارد. در حالی که در RAID سطح 1، به چهار دیسک اضافی نیاز است .

RAID سطح ۳: این سطح که به نام سازمان توازن جایگذاری بیت خوانده می­شود، سطح ۲ را بهبود می­بخشد. در این سطح، کنترلگرهای دیسک می­توانند تشخیص دهند که آیا سکتورها درست خوانده شدند یا خیر. درنتیجه می­توان از یک بیت توازن برای تشخیص و تصحیح خطا استفاده کرد. این کار به این صورت است: اگر یکی از سکتورها خراب شود، دقیقاً می دانیم که آن سکتور کدام است، و برای هر بیت موجود در سکتور، می­توانیم تشخیص دهیم که آیا یک یا صفر است. برای این کار، توازن بیت­های متناظر در سکتورهای توازن بیت­های دیگر را محاسبه می­کنیم. اگر توازن بیت­های باقیمانده برابر با توازن ذخیره شده باشد، بیت از دست رفته، صفراست، وگرنه یک می­باشد. RAID سطح ۳ به خوبی سطح ۲ است ولی به تعداد کمتری از دیسک­ها نیاز دارد (فقط به یک دیسک اضافی نیاز دارد). لذا، سطح ۲ در عمل به کار نمی­آید. این طرح در قسمت ت شکل آمده است.

RAID سطح 3 نسبت به سطح ۱، برتری دارد. برخلاف سطح ۱ که برای هر دیسک نیاز به یک دیسک آینه­ای بود، در سطح ۳ برای چندین دیسک معمولی نیاز به یک دیسک توازن است.

لذا، سربار ذخیره سازی کاهش می­یابد. چون خواندن و نوشتن بایت­ها با نواربندی N طرفه­ی داده­ها بر روی چندین دیسک پخش می­شود، نرخ انتقال برای خواندن با نوشتن یک بلوک N برابر سریع تر از RAID سطح ۱ با استفاده از نواربندی N طرفه است. از طرف دیگر، RAID سطح ۳، در هر ثانیه از تعداد اندکی از I/O پشتیبانی می­کند، زیرا هر دیسک باید در هر درخواست I/O شرکت کند.

مسئله­ی دیگر کارایی RAID سطح ۳ – و تمام سطوح RAID مبتنی بر توازن – گران بودن محاسبات و نوشتن توازن است. این سربار منجر به کند شدن عمل نوشتن، در مقایسه با آرایه­های RAID بدون توازن می شود. برای تعديل این جریمه­ی کارایی، بسیاری از آرایه­های حافظه­ی RAID شامل کنترلگر سخت افزاری همراه با سخت افزار توازن اختصاصی است. این کنترلگر، محاسبات توازن از CPU را به آرایه واگذار می­کند. آرایه دارای حافظه ی نهان NVRAM است که در اثنای محاسبه­ی توازن، بلوک­ها را ذخیره می­کند و نوشتن از کنترلگر به محورهای دیسک را در میانگیر قرار می­دهد. این ترکیب موجب می­شود سرعت RAID با توازن به سرعت RAID بدون توازن برسد. در حقیقت، آرایه­ی حافظه­ی نهان که RAID با توازن را پیاده سازی می­کند، نسبت به حالت بدون حافظه­ی نهان و بدون توازن، ارجح است.

RAID سطح 4: این سطح که سازمان توازن جایگذاری بلوک نام دارد، همانند RAID سطح صفر از نواربندی سطح بلوک استفاده می­کند و یک بلوک توازن را در دیسک جداگانه­ای نگه می­دارد تا با بلوک­هایی از N دیسک دیگر، تناظر برقرار شود. این طرح در قسمت ث شکل آمده است. اگر یکی از دیسک­ها خراب شود، بلوک توازن می­تواند با بلوک­های متناظر در دیسک­های دیگر به کار گرفته شود تا بلوک­های دیسک خراب شده را بازیابی کند.

خواندن بلوک فقط به یک دیسک دستیابی دارد و اجازه می­دهد که سایر درخواست­ها توسط دیسک-های دیگر پردازش شوند. لذا، نرخ انتقال داده­ها برای هر دستیابی، کندتر است، اما چندین دستیابی خواندن می­توانند به طور موازی انجام شوند و در نتیجه نرخ کلی I/O افزایش یابد. نرخ­های انتقال برای خواندن­های بزرگ، زیاد است، زیرا تمام دیسک­ها را می­توان به طور موازی خواند. نرخ انتقال نوشتن­های بزرگ نیز زیاد است، زیرا داده­ها و توازن می­توانند به طور موازی نوشته شوند.

از طرف دیگر، نوشتن­های کوچک و مستقل، نمی­توانند به طور موازی انجام شوند. نوشتن بلوک باید به دیسکی دستیابی داشته باشد که بلوک در آنجا ذخیره شده است. علاوه بر این باید به دیسک توازن نیز دستیابی داشته باشد، زیرا بلوک توازن باید نوسازی شود. مقدار قبلی بلوک توازن و مقدار قبلی بلوکی که در حال نوشتن است، برای توازن جدیدی که در حال محاسبه شدن است، باید خوانده شود. این عمل را چرخه­ی خواندن-اصلاح-نوشتن گویند. لذا، در یک عمل نوشتن چهار بار به دیسک دستیابی می­شود: دو خواندن برای دو بلوک قبلی، و دو نوشتن برای دو بلوک جدید.

WAFL از RAID سطح 4 استفاده می­کند، زیرا این سطح RAID اجازه می­دهد دیسک­ها به طور یکنواخت به مجموعه­ی RAID اضافه شوند. اگر دیسک­های اضافه شده، با بلوک­های حاوی صفر مقدار اولیه بگیرند، آنگاه مقدار توازن تغییر نمی­کند، و مجموعه­ی RAID هنوز درست است.

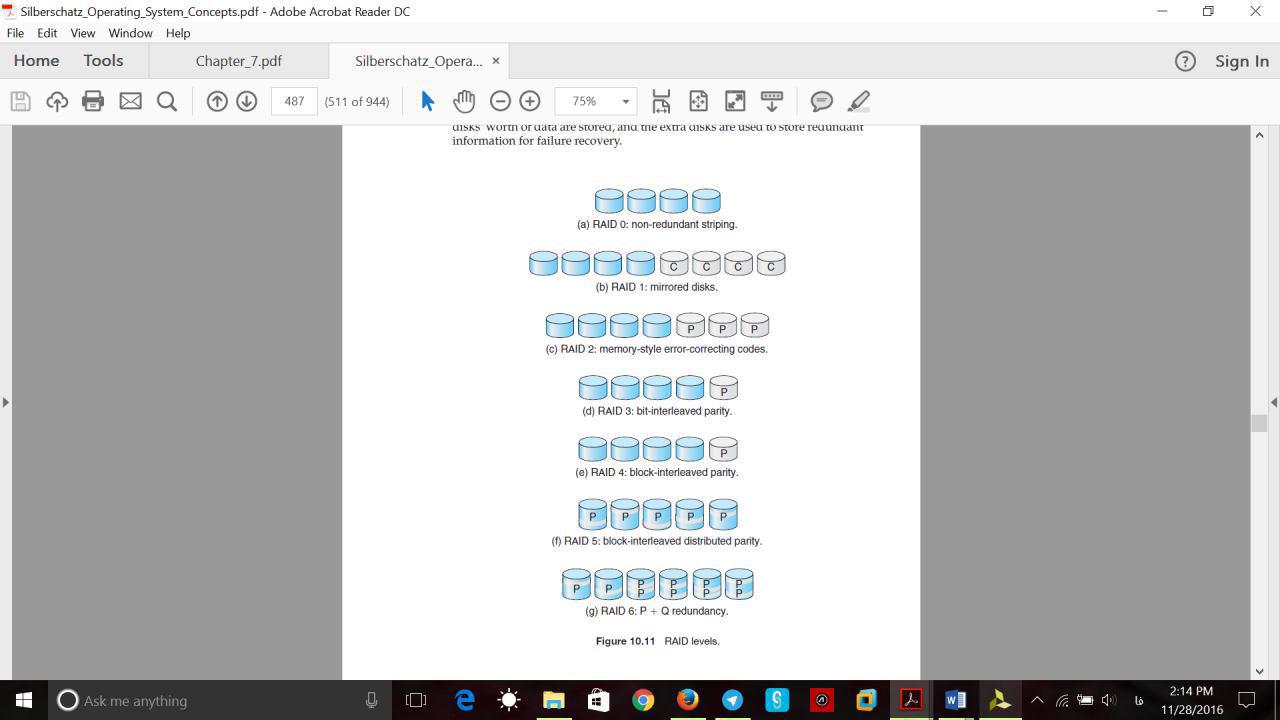
RAID سطح 5: در این سطح که توازن توزیع شده­ی جایگذاری بلوک نام دارد، از این جهت با سطح 4 متفاوت است که داده و توازن را بین N+1 دیسک پخش می­کند (در سطح 4، داده­ها در N دیسک و توازن در یک دیسک ذخیره می­شد). برای هر بلوک، یکی از دیسک­ها، توازن و سایر دیسک­ها، داده­ها را ذخیره می­کنند. به عنوان مثال، در آرایه­ای با ۵ دیسک، توازن مربوط به بلوک nام در دیسک (n mod 5) + 1 ذخیره می­شود. این سازماندهی در قسمت ج شکل نشان داده شده است. در این شکل، Pها در تمام دیسک­ها توزیع می­شوند. بلوک توازن نمی­تواند توازن مربوط به بلوک­ها را در یک دیسک ذخیره کند، زیرا خرابی دیسک منجر به مفقود شدن داده­ها و توازن می­شود. در نتیجه، قابل ترمیم نیست. RAID سطح ۵ با پخش توازن در تمام دیسک­ها، از به کارگیری یک دیسک توازن اجتناب می­کند.

RAID سطح ۶: این سطح که طرح افزونگی P+Q نام دارد، خیلی شبیه به RAID سطح 5 است، اما اطلاعات اضافی را ذخیره می­کند تا در اثر خرابی چندین دیسک، امکان بازیابی داد­ه­ها وجود داشته باشد. به جای استفاده از توازن، کدهای تصحیح خطایی مثل کدهای رید-سالامون به کار گرفته می­شوند. در طرحی که قسمت چ شکل آمده است، برای هر چهار بیت از داده­ها،دو بیت از داده­ی اضافی ذخیره شده است (بر خلاف یک بیت توازن در RAID سطح ۵)، و سیستم می­تواند خرابی دو دیسک را تحمل کند.

RAID سطح 0+1: این سطح ترکیبی از RAID سطوح صفر و یک است. RAID سطح صفر، کارایی را تدارک می­بیند، در حالی که RAID سطح 1 قابلیت اعتماد را ارائه می­کند. به طور کلی، کارایی بهتری نسبت به RAID سطح 5 ارائه می­کند. در محیط­هایی متداول است که کارایی و قابلیت اعتماد مهم باشند. متاسفانه، دیسک­های مورد نیاز برای ذخیره­سازی را دوبرابر می­کند (همانند RAID سطح 1) و در نتیجه گران است. در RAID سطح 0+1، مجموعه­ای از دیسک­ها نواربندی می­شوند، و سپس این نوار به نوار معادل دیگر آینه­ای می­شود.

گزینه­ی دیگری از RAID که در حال متداول شدن است، RAID سطح 1+0 است ک دیسک­ها به صورت جفت­هایی آینه­ای می­شوند، و سپس این جفت­های آینه­ای، نواربندی می­شوند.

این RAID از نظر تئوری نسبت به RAID سطح 0+1 امتیازاتی دارد. به عنوان مثال، اگر یک دیسک در RAID سطح 0+1 خراب شود، كل نوار غیر قابل دسترس است و فقط نوار بعدی مهیا است. اگر RAID سطح 1+0 خرابی به وجود آید، یک دیسک مهیا نیست، اما جفت آینه­ای آن همانند سایر دیسک­ها آماده است. سرانجام، یادآور می­­شویم که تغییرات زیادی در سطوح مختلف RAID به وجود آمده است. در نتیجه، ممکن است سطوح RAID دقیقاً همان چیزی نباشند که در اینجا مطرح شده­اند.



تمرین :3استفاده از حافظۀ پنهان (Cache)برای دیسک چه مزیت هایی دارد؟

**حافطه پنهان دیسک**

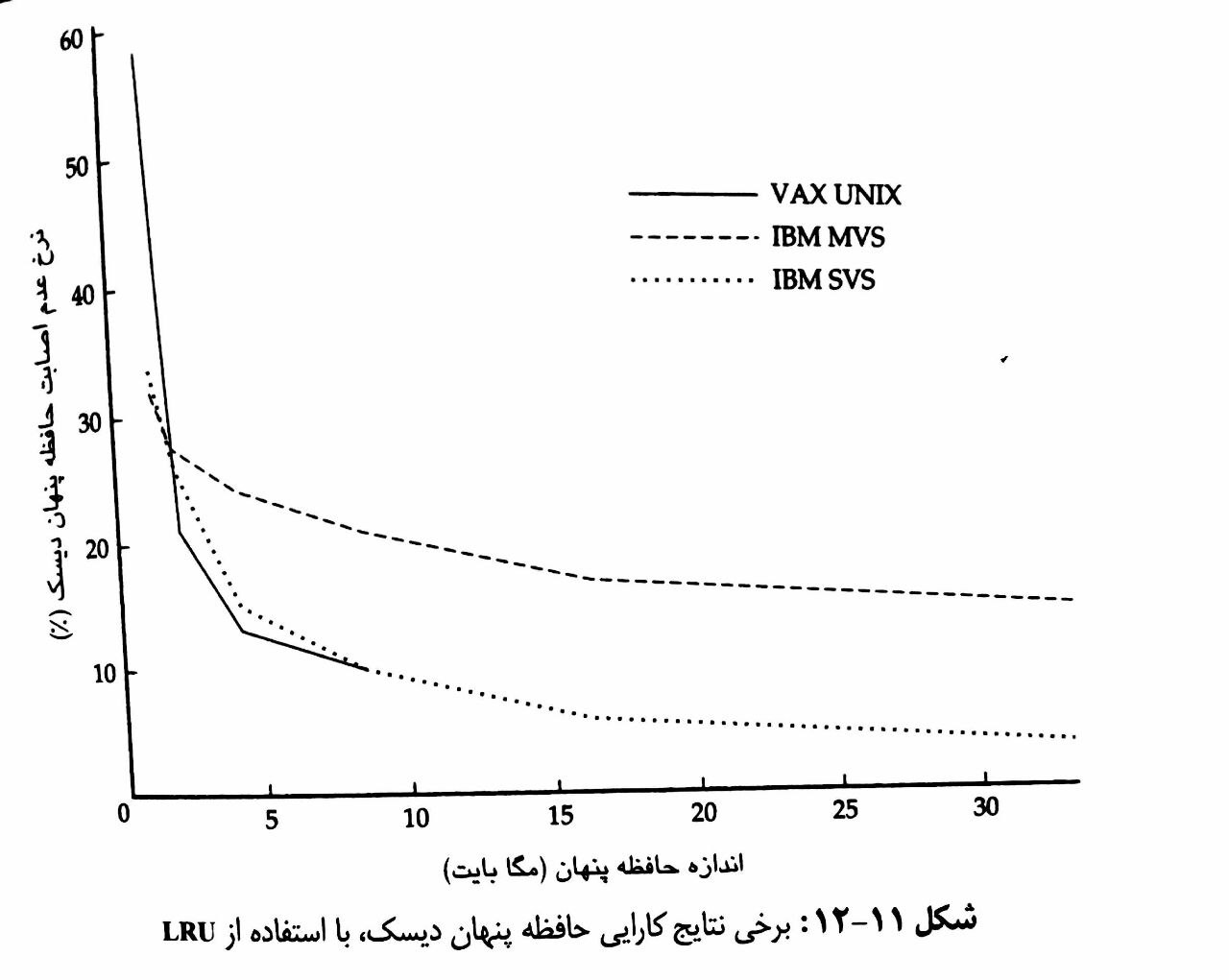
واژه حافظه نهان معمولا درباره حافظه ای بکار میرود که کوچکتر و سریعتر از حافظه اصلی باشد و در بین حافظه اصلی و پردازنده قرار گیرد.چنین حافظه پنهانی با بهره گیری از اصل محلی بودن ، متوسط زمان دسترسی به حافظه را کاهش دهد.

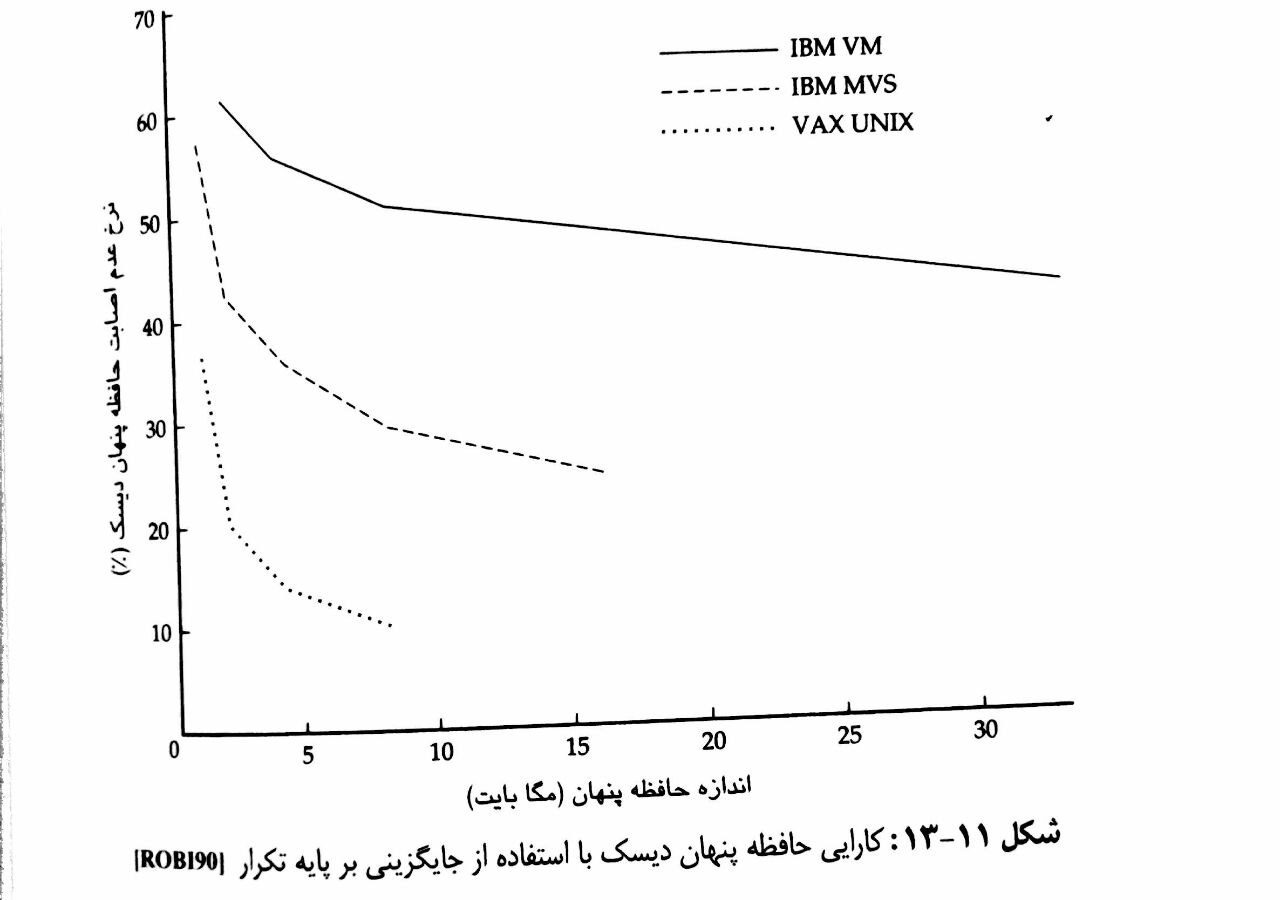
همین اصل میتواند برای حافظه دیسک هم اعمال شود. بطور مشخص حافظه پنهان دیسک میانگیری در حافظه اصلی ، برای قطاع های دیسک است . این حافظه پنهان حاوی یک کپی از برخی قطاع های دیسک است.. هنگامی که یک درخواست ورودی/خروجی برای قطاع خاصی مطرح شود وجود قطاع مزبور در حافظه پنهان بررسی می شود. اگر موجود باشد ، درخواست از طریق حافظه پنهان برآورده می شود وگرنه قطاع درخواست شده از دیسک به داخل حافظه پنهان دیسک خوانده می شود.

به دلیل پدیده محلی بودن مراجعات ، زمانی که بلوکی از داده ها برای یک درخواست ورودی/خروجی به داخل حافظه پنهان آورده می شود، به احتمال زیاد در آینده نیز به این بلوک مراجعه خواهد شد.

موضوع کارایی حتافظه پنهان به این مساله منجر می شود که آیا می توان به یک نسبت عدم اصابت موردنظر رسید؟ این مطلب به رفتار محلی بودن مراجعات به دیسک، الگوریتم جایگزینی و دیگر عوامل طراحی بستگی دارد.اصولا نسبت عدم اصابت تابعی از اندازه حافظه پنهان دیسک است.

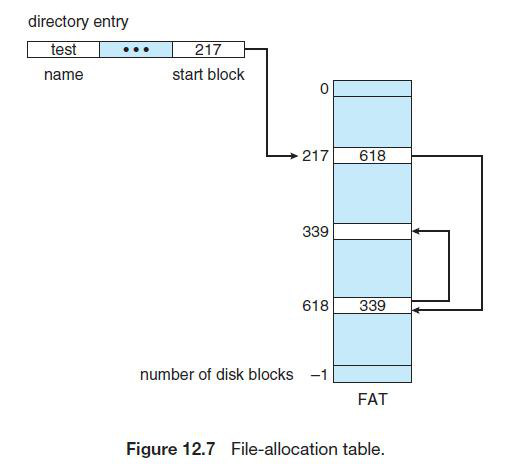
مقایسه نمودار های نتایج کارایی حافظه پنهان با استفاده از LRU (least frequently used) با کارایی آن با استفاده از الگوریتم جایگزینی بر پایه تکرار ، یکی از خطرات چنین ارزیابی کارایی را گوشزد میکند. به نظر می رسد این شکل ها مبین کارایی بهتر LRU از الگوریتم جایگزینی بر پایه تکرار است. ولی اگر الگوهای مراجعه یکسان، با بکارگیری ساختار حافظه پنهان مشابه ، مورد مقایسه قرار گیرند، الگوریتم جایگزینی بر پایه تکرار برتری دارد.بنابراین علاوه بر مسائل طراحی مرتبط مثل بلوک، ترتیب دقیق الگوهای مراجعه نیز تاثیر عمیقی بر کارایی خواهند داشت.





تمرین :4دربارۀ جدول FATدر سیستم عامل DOSتحقیق کنید

شکل دیگری از تخصیص پیوندی(در این روش هر فایل یک لیست پیوندی از بلوک های دیسک را دارد ، به طوری که بلوک های دیسک میتوانند در هر جایی از دیسک باشند دایرکتوری حاوی اشاره گر هایی به اولین و اخرین بلوک فایل است. ) استفاده از جدول ایجاد فایل (FAT) است. این روش ساده و کار آمد در سیستم عامل MS- DOS و OS/2 مورد استفاده قرار میگیرد. بخشی از دیسک در ابتدای هر پارتیشن برای نگه داری این جدول در نظر گرفته میشود. هر بلوک دیسک یک ورودی در این جدول دارد و جدول بر حسب شماره بلوک شاخص بندی شده است. این جدول همانند لیست پیوندی به کار گرفته میشود. ورودی دایرکتوری شامل شماره اولین بلوک فایل است. ورودی جدول بر حسب شماره بلوک شاخص بندی میشود و حاوی شماره بلوک بعدی موجود در فایل است. این زنجیر تا اخرین بلوک ادامه می یابد که حاوی مقدار خاصی برای نمایش انتهای فایل است. بلوک های بلا استفاده در جدول با صفر مشخص میشوند. برای تخصیص بلوک های جدید به فایل، ورودیی از جدول با مقدار صفر پیدا میشود و به جای مقدار نشانگر انتهای فایل ، ادرس بلوک جدید قرار میگیرد و مقدار انتهای فایل به جای صفر قرار میگیرد . نمونه ای از دیسک FAT برای فایل حاوی بلوک های دیسک 217 و 616 و 339 در شکل زیر آماده است .



توجه شود که استفاده از این جدول برای تخصیص فایل ، منجر به پیگرد های هد دیسک میشود ، مگر اینکه FAT در حافظه پنهان شود. هد دیسک باید به ابتدای پارتیشن منتقل شود تا FAT را بخواند و محل بلوک مطلوب را بیابد. و سپس به مخل بلوک خود برود د بدترین حالت، برای هر بلوک این دو انتقال هد صورت میگیرد . فایده اش این است که زمان دستیابی تصادفی بهبود میابد، زیرا هد دیسک میتواند محل هر بلوک را با خواندن اطلاعات موجود درFAT پیدا کند..

تمرین :5دربارۀ سیستم NTFSدر ویندوز تحقیق کنید. خصوصیات مهم آن را بیان کنید

**سیستم پرونده در ویندوز:**

W2K از چند سیستم پرونده از جمله جدول تخصیص پرونده (FAT) که روی windows95 ,MS-DOS,OS/2 اجرا می شود حمایت می کند. ولی پدید آورندگان W2K سیستم پرونده جدیدی نیز طراحی کرده اند که سیستم پرونده NTFS(new technology file system) میباشد و برای برآوردن نیازهای بالای ایستگاه های کاری و خدمتگزارها در نظر گرفته شده است.موارد زیر مثال هایی از کاربردهای با نیاز بالا هستند.

* کاربردهای مشتری/خدمتگزار مثل خدمتگذارهای پرونده ها ، خدمتگزارهای محاسبات و خدمتگزارهای پایگاه داده ها
* کاربردهای مهندسی و علمی نیازمند منابع بسیار
* کاربردهای شبکه برای سیستمهای بزرگ شرکتی

**مشخصه های اصلی NTFS**

همانگونه که خواهیم دید NTFS قابل انعطاف و پرقدرت است و بر اساس مدل سیستم پرونده ساده به زیبایی ساخته شده است. مهمترین مشخصه های قابل ذکر NTFS شامل موارد زیر است:

* **قابل ترمیم:** در بالای لیست خواسته های سیستم پرونده جدید W2K قابلیت ترمیم در مقابل فروپاشی های سیستم و شکستهای دیسک قرار دارد.در صورت وقوع چنین شکستهایی ،NTFS قابلیت بازسازی دیسک ها و بازگرداندن آنها به حالت پایدار را دارد. این عمل را با استفاده از یک مدل پردازش تراکنش برای تغییرات سیستم پرونده انجام می دهد.با هر تغییر محسوس ، به عنوان یک عمل تفکیک ناپذیر رفتار میشود که یا تمامی آن انجام می پذیرد یا اصلا انجام نمی گیرد.هر تراکنشی که در زمان شکست در حال انجام باشد یا کنار گذاشته میشود یا تکمیل می گردد. علاوه بر آن ،NTFS برای داده های بحرانی سیستم پرونده از ذخیره سازی اضافی استفاده می کند،در نتیجه شکست یک قطاع دیسک موجب از دست رفتن داده هایی که ساختار و وضعیت سیستم پرونده را توصیف میکنند،نمیشود.
* **امنیت:**NTFS از مدل شئ W2K برای اعمال امنیت استفاده می کند. پرونده باز به عنوان یک شئ پرونده با توصیفگر امنیتی که خصیصه های امنیتی آن را تعریف می کند، پیاده سازی میشود.
* **دیسک ها و پرونده های بزرگ:** NTFS کاراکتر از بسیاری از سیستمهای پرونده (ازجمله FAT) از دیسک ها و پرونده های بسیار بزرگ حمایت می کند.
* **جریان های داده ای چندگانه:** با محتوای واقعی پرونده مانند جریانی از بایت ها رفتارمیگردد. در NTFS امکان تعریف جریانهای داده ای چندگانه برای پرونده واحد وجود دارد. مثالی از سودمندی این خصوصیت آنست که W2K اجازه می دهد توسط سیستمهای macintosh هر پرونده دارای دو مولفه است : داده های پرونده و منابع fork که حاوی اطلاعاتی در مورد پرونده است. NTFS با این دومولفه مانند دو جریان داده رفتار می نماید.
* **امکان شاخص بندی عام:** NTFS مجموعه ای از خصیصه ها را به هر پرونده نسبت می دهد.مجموعه توصیفات پرونده ها در سیستم مدیریت پرونده به صورت یک پایگاه داده رابطه ای سازمان یافته اند،در نتیجه پرونده ها میتوانند با هر خصیصه ای شاخص بندی شوند.

تمرین :6فرمت سطح پایین یا فرمت فیزیکی دیسک چیست؟

دیسک مغناطیسی یک لوح خالی است ، یعنی صفحاتی از ماده ضبط مغناطیسی است. قبل از ذخیره داده ها بر روی دیسک ، باید به قطاعهایی تبدیل شود که کنترل کننده ی دیسک بتواند آنها را بخواند و بنویسد . این فرآیند را فرمت سطح پاییم یا فرمت فیزیکی گویند. فرمت سطح پایین هر قطاع را با ساختمان داده خاصی پر میکند .ساختمان داده هر قطاع معمولا شامل سرایند ،یک ناحیه داده معمولا512 بایت و یک پسایند است. سرایند و پسایند حاوی اطلاعاتی اند که توسط کنترل کننده ی دیسک مورد استفاده قرار میگیرد. مثل شماره قطاع و کد تصحیح خطا . اغلب دیسک های سخت در کار خانه سازنده فرمت سطح پایین میشوند.

تمرین :7بلاک بد در دیسک چیست؟ این بلاک ها در استراتژی های متفاوت، چگونه مدیریت می شوند؟

چون دیسک ها دارای قطعات متحرک با میزان تحمل پذیری خطای پایینی هستند، احتمال خرابی آن ها وجود دارد. گاهی اوقات خرابی کامل است، در این حالت بایستی دیسک جایگزین شده و محتویات آن از رسانه های پشتیبان بر روی دیسک جدید بازیابی شود. در بیشتر موارد، یک یا چندین قطاع معیوب می شوند. حتی اکثر دیسک هایی که از کارخانه می آیند بلاک های بد دارند. با این بلاک ها، بسته به دیسک و کنترلر مورد استفاده به شیوه های مختلفی برخورد می شود.

در دیسک های ساده، نظیر برخی دیسک ها با کنترلرهای IDE می توان بلاک های بد را به صورت دستی رفع کرد. یک راهبرد، پویش دیسک برای یافتن بلاک های بد در زمان فرمت دیسک است. هر بلاک بدی که پیدا می شود، پرچم غیر قابل استفاده می خورد تا آن که سیستم فایل آنها را تخصیص ندهد. اگر بلاک ها در طی عملیات عادی خراب شوند، بایستی برنامه خاصی (همچون فرمان badblocks در لینوکس) به صورت دستی اجرا شود تا بلاک های بد را جست و جو کرده و همانند حالت قبل، آن ها قفل کرده و از رده خارج کند. معمولا داده هایی که مقیم بلاک های بد بودند، گم می شوند.

دیسک های پیچیده تر در رابطه با ترمیم بلاک ای بد هوشمندانه عمل می کنند. کنترلر، لیستی از بلاک های بد روی دیسک را نگه می دارد. این لیست در کارخانه طی فرمت سطح پایین مقدار اولیه می گیرد و در طول حیات دیسک به روز می گردد. فرمت سطح پایین، قطاع هایی را هم به عنوان یدک به دور از چشم سیستم عامل کنار می گذارد. می توان از کنترلر خواست تا هر یک از بلاک های بد را به صورت منطقی با یکی از قطاع های یدک جایگزین کند. این طرح به یدک گذاری یا روانه ساختن قطاع معروف است. تراکنش قطاع بد نوعی می تواند به صورت زیر باشد:

- سیستم عامل سعی دارد تا بلاک منطقی 87 را بخواند.

- کنترلر، ECC را محاسبه کرده و در می یابد که قطاع بد است. این یافته را به سیستم عامل گزارش می کند.

- در نوبت بعدی که سیستم مجددا راه اندازی شود، فرمان خاصی اجرا می شود تا به کنترلر بگوید که قطاع بد را با یک قطاع یدک جایگزین کند.

- بعد از آن، هر بار که سیستم، بلاک منطقی 87 را درخواست کند، این درخواست توسط کنترلر به آدرس قطاع جایگزین ترجمه می شود.

توجه داشته باشید که با چنین تغییر مسیری که توسط کنترلر صورت می گیرد ممکن است هر گونه بهینه سازی الگوریتم زمان بندی دیسک سیستم عامل بی اعتبار شود! به همین دلیل بیشتر دیسک ها طوری فرمت می شوند که تعدادی قطاع یدک نیز در هر استوانه باقی بماند و یک استوانه را هم برای یدک نگه می دارند. زمانی که بلاک بعدی تجدید نگاشت می شود، کنترلر در صورت امکان، قطاعی از همان استوانه را به عنوان یدک به کار می برد.

برخی کنترلرها را می توان دستور داد تا به عنوان روش دیگری برای یدک گذاری قطاع، بلاک بد را با لغزاندن قطاع جایگزین کنند. این مثال را ببینید: فرض کنید که بلاک منطقی 17 معیوب می شود و اولین یدک موجود بعد از قطاع 202 قرار دارد. در این جا لغزاندن قطاع، نگاشت تمامی قطاع ها را از 17 تا 202 با انتقال یک قطاع به پایین تجدید می کند. یعنی قطاع 202 به یدک کپی می شود، سپس قطاع 201 به 202 و پس از آن 200 به 201 و الی آخر، تا این که قطاع 18 به قطاع 19 کپی شود. به این ترتیب با لغزاندن قطاع ها، فضای قطاع 17 آزاد می شود، لذا قطاع 17 می تواند در آن نگاشته شود.

در حالت کلی، جایگزینی بلاک بد کالا خودکار نیست، چون معمولا داده های بلاک های بد گم می شوند. فرایند در مواجهه با خطاهای نرم می تواد بلاک داده ها را کپی کرده و برای این کار، از یدک گذاری یا لغزاندن بلاک استفاده کند. با این وجود، وقوع یک خطای دشوار غیر قابل ترمیم می تواند باعث از بین رفتن داده ها شود. هر فایلی که از آن بلاک استفاده می کرد بایستی ترمیم یابد (برای نمونه، با بازیابی از نوار پشتیبان) و این کار نیازمند مداخله دستی است.

تمرین :8کاربرد بلاکِ بوت در دیسک و سیستم عامل چیست؟ دراین راستا تعاریفی مانند «بوت پارتیشن»، «بوت سکتور» و MBRرا بررسی کنید.

برای اینکه کامپیوتر بتواند اجرای خود را شروع نماید ، برای نمونه ، بازدن دکمه ی power یا با راه اندازی مجدد بایستی یک برنامه ی اولیه برای اجرا داشته باشد. این برنامه ی اولیه بالا آورنده ( bootstrap) تمایل به سادگی دارد . تمامی جوانب سیستم را از ثبات های پردازنده گرفته تا کنترلر دستگاه ها و محتویات اصلی مقدار دهی می نماید و سپس اجرای سیستم عامل را شروع میکند. برای انجام این کار، برنامه ی بالاآورنده هسته ی سیستم عامل را از دیسک می یابد، آن را داخل حافظه بار گذاری میکند و به یک آدرس اولیه پرش دارد تا اجرای سیستم را شروع کند . بالاآورنده در اکثر کامپیوتر ها در ROM ذخیره میشود. این مکان مناسب است چون مقدار دهی اولیه برای ROM لازم نبوده و در یک مکان ثابتی قرار دارد که پردازنده میتواند با زدن power یا reset اجرای آن را شروع کند و از آنجایی که ROM فقط خواندنی است ، لذا ویروس کامپیوتری نمیتواند روی آن تاثیر گذار باشد . تنها مساله ی آن این است که تغییر کد این بالا آورنده مستلزم تغییر تراشه های سخت افزاری ROM خواهد بود . به این دلیل ، بیشتر سیستم ها یک برنامه ی بالا آورنده ی بسیار کوچکی در ROM راه انداز ذخیره میکنند که کار آن ، تنها آوردن برنامه ی بالا آورنده ی کامل از دیسک به حافظه است . برنامه ی راه انداز کامل را میتوان به آسانی تغییر داد: نسخه ی جدید آن به سادگی روی دیسک نوشته میشود. برنامه ی بالاآورنده کامل در **بلاک های بوت** در مکانی ثابت از دیسک ذخیره میشود. هر دیسک یک پارتیشن بوت دارد، دیسک بوت یا دیسک سیستم ( boot disk or system disk) نامیده میشود . کد موجود در ROM راه انداز به کنترلر دیسک دستور میدهد تا بلاک های بوت را به داخل حافظه بخواند ( در اینجا ، هیچ درایور دستگاهی بار گذاری نشده است ) و سپس اجرای آن کد را شروع میکند . برنامه ی بالاآورنده ی کامل ، پیچیده تر از بار گذار بالاآورنده ی موجود در ROM راه انداز است. آن قادر است کل سیستم عامل را از یک مکان غیر ثابت دیسک باگذاری کرده و اجرای سیستم عامل را شروع نماید. اگر چه کد بالاآورنده ی کامل هم میتواند کوچک باشد.

اکنون به عنوان مثالی در این زمینه ، نگاهی به فرایند راه انداز ی ویندوز می اندازیم . در ابتدا توجه شود که ویندوز به دیسک سخت اجازه میدهد تا به پارتیشن هایی تقسیم شود و یک پارتیشن که به عنوان پارتیشن راه انداز (boot partition) شناخته میشود، شامل سیستم عامل و درایور دستگاه هاست. سیستم ویندوز کد راه انداز خود را در اولین قطاع دیسک سخت ، که تغییر آن ، رکورد راه انداز اصلی یا MBR است قرار میدهد . راه اندازی با اجرای کدی که مقیم حافظه ی ROM سیستم است شروع میشود. این کد سیستم را هدایت میکند تا کد راه انداز را از MBR بخواند . علاوه بر کد راه انداز، MBR شامل یک جدول لیست برای پارتیشن های دیسک سخت و یک پرچم که نشانگر پارتیشن راه انداز را شناسایی کرد، اولین قطاع ( که قطاع راه اندازه ( boot sector) نامیده میشود ) آن پارتیشن را خوانده و باقی فرایند راه اندازی را تداوم میبخشد که شامل بارگذاری زیر سیستم ها و سرویس های مختلف سیستم میباشد.