دانشگاه صنعتی شریف
دانشکده مهندسی کامپیوتر

عنوان:

پیاده‌سازی ویژگی‌های پیشرفته برای یک فایل‌سیستم

(پروژه شماره ۱)

اعضای گروه

هلیا اخترکاویان

کیان عمومی

امیرصدرا عبدالهی

نام درس

سیستم‌های عامل

نیم‌سال اول ۱۴۰۱-۱۴۰۲

نام استاد درس

دکتر اسدی

۱ کلیات

با توجه به ویژگی هایی که پیاده سازی کردیم، به توضیح نحوه پیاده سازی هر یک می پردازیم و سعی می کنیم مراحل پیاده سازی را با توجه به تابع هایی که تغییر دادیم شفاف سازی کنیم تا درک آن برای خواننده راحت تر شود.

۲ ویژگی ها

ویژگی هایی که پیاده سازی کرده ایم به همراه توضیح به شرح زیر می باشند:

۲-۱ عدم تغییر فایل‌هایی که با lock شروع می‌شوند

برای اینکار تابع راis_important_file پیاده سازی کردیم. این تابع چک می کند که آیا رشته filename با lock شروع شده است یا خیر. در توابع write و delete نیز در ابتدای آنها از این تابع برای چک کردن این ویژگی استفاده کرده ایم و در صورت وجود مشکل، ارور مناسب را چاپ می کنیم

۲-۲ تصحیح نحوه محاسبه offset

در قسمت write مشکلی که وجود دارد این است که وقتی میخواهد از آخرین بلاک فایل شروع کند و بافر را از آنجا بنویسد، دیتای بلاک آخر که از پیش در فایل بوده است را overwrite می کند و با bounce_buff جایگزین می کند که در ابتدا یک بافر خالی است. با این کار در واقع دیتای بلاک آخر از دست می رود. برای اینکه به این مشکل بر نخوریم باید در ابتدای نوشتن یک کپی از بلاک آخر بگیریم و در bounce_buff بریزیم و سپس بافر اصلی را به bounce_buff اضافه کنیم. پس از آن می توانیم bounce_buff را با دیتای آخرعوض کنیم یا به نوعی overwrite کنیم. کدی که لازم است اضافه شود به صورت زیر است که فقط به ازای بلاک آخر فایل اجرا میشود(که همان اولین بلاک در هنگام نوشتن است)

```
if(i==0)
    block_read(curr_fat_index + superblock->data_start_index,bounce_buff); //correct offset
```

۲-۳ استفاده از حافظه نهان

در این قسمت همانطور که از ما خواسته شده بود حافظه را به سه قسمت تقسیم کردیم که یکی برای حافظه نهان نوشتن ، دیگری برای حافظه نهان خواندن و آخری هم برای خود دیسک می باشد. ابتدا به فرآیند استفاده از حافظه نهان نوشتن می پردازیم. برای اینکار از تابع fs_write_cache استفاده می کنیم که مسئولیت نوشتن بر روی حافظه نهان را دارد. این تابع به این صورت عمل می کند که پس از پیدا کردن دیتا بلاک های خالی، به تفکیک این که هرکدام در کدام قسمت قرار دارند به نوشتن دیتا روی بلاک ها می پردازد. به این صورت که اگر دیتای خالی در حافظه نهان وجود داشت که کار خاصی نمی کند و مستقیم روی آن بلاک دیتا را می نویسد. اما اگر دیتا بلاک خالی در دیسک بود، آنگاه یک دیتا بلاک به طور رندوم از حافظه نهان انتخاب می کند و اگر block dirty بود آن را ابتدا روی دیسک خالی می نویسد و سپس دیتای جدید را روی آن دیتا بلاک می نویسیم که درون حافظه نهان است. توجه داشته باشید که در حین این مراحل آرایه FAT_table نیز به درستی باید آپدیت شود و تغییرات روی اندیس دیتا بلاک ها باید روی این آرایه اعمال شود تا زنجیره دیتا بلاک های فایل ها خراب نشود.

برای حافظه نهان خواندن نیز تنها کاری که نیاز بود انجام دهیم این بود که در تابع fs_read اگر دیدیم دیتایی که می خواهیم از آن بخوانیم در حافظه نهان نیست، آنگاه با استفاده از تابع bring_to_read_cache آن را پس از خواندن به یک دیتا بلاک خالی در حافظه نهان بیاوریم و یا اگر دیتا بلاک خالی وجود نداشت با استفاده از عملیات جایگزینی، یک دیتا بلاک را به دیسک ببریم و از آن برای ریختن دیتای جدید استفاده کنیم.

برای آپدیت کردن اندیس دیتا بلاک ها در آرایه FAT_table از دو تابع swap_fat_i_and_j و move_fat_i_to_j استفاده می کنیم که همانطور که از اسم هایشان مشخص است اولی برای جا به جایی دو دیتا بلاک و دومی برای انتقال یک دیتا بلاک به دیتا بلاک دیگر استفاده می شود.

توابع is_in_disk و is_in_read_cache و is_in_write_cache برای مشخص کردن این است که آیا اندیس مورد نظر در حافظه نهان ذکر شده وجود دارد یا نه. یا اینکه اصلا در دیسک هست یا نه.

برای جایگزینی دیتا بلاک ها نیز در هر دو حافظه نهان از الگوریتم رندوم استفاده کردیم به این نحو که یک دیتا بلاک رندوم را از بین دیتا بلاک های واجد شرایط انتخاب کردیم و عملیات های مربوط به جایگزینی را روی آن انجام دادیم.

توجه داشته باشید که پیاده سازی به صورتی انجام شده است که می توان با استفاده از متغیر cache_is_on مشخص کرد که آیا از حافظه نهان استفاده کنیم یا خیر.

در مقام مقایسه نیز وقتی که از حافظه نهان استفاده کردیم، زمان اجرای تست های یکسان در حدود چند صد میکرو ثانیه کم شد. مقایسه را نیز به این صورت انجام دادیم که اول و آخر تابع main در فایل test از clock استفاده کردیم و در نهایت زمان اجرا را با استفاده از اختلاف دو متغیر start و end به دست آوردیم. همین تست ها را نیز یک بار با استفاده از حافظه نهان اجرا کردیم و اعداد به دست آمده را با یکدیگر مقایسه کردیم.

امتیازی: یک مورد اضافه تری که نسبت به خواسته سوال پیاده سازی کردیم قسمت استفاده از الگوریتم جایگزینی در حافظه نهان نوشتن است. به طوری که وقتی می یابیم که دیتا بلاک خالی در حافظه نهان وجود ندارد، همه بلاک های کثیف را روی دیسک به یک باره نمی نویسیم و فقط در هر بار یک دیتا بلاک را با الگوریتم رندوم انتخاب می کنیم و اگر کثیف بود آن را روی دیسک می نویسیم و عملیات جایگزینی را روی آن اجرا می کنیم.

۲-۴ نوشتن در حین ساختن فایل و پاک کردن دیتا بلاک ها در حین حذف فایل

برای اینکه بتوانیم محتویات فایل را در حین ساختن در دیسک بنویسیم، مقداری تعریف تابع fs_create را تغییر دادیم به طوری که سایز و بافری که قرار است نوشته شود را هم به آن می دهیم. پس از اینکه عملیات های مربوط به خود create انجام شد، ابتدا فایل را با استفاده از fs_open در آرایه fd_table قرار می دهیم و سپس با استفاده از تابع ،fs_write بافری که در ابتدا به fs_create داده شده است را در دیتا بلاک های مربوط به فایل باز شده می نویسیم. در تمام موارد حواسمان به ارور ها و اختلال هایی که ممکن است پیش بیاید بوده است و ارور های مناسب را چاپ می کنیم. در کد اولیه، هنگامی که فایلی پاک می شود محتویات آن از دیتا بلاک های متناظرش پاک نمی شود و فقط descriptor file به حالت اولیه باز می گردد و FAT آن نیز empty میشود. در اینجا ما در هنگام پاک کردن FAT درهمان حین به پاک کردن دیتا بلاک متناظرش نیز می پردازیم. به طوری که یک بافر خالی به اندازه size block در آن بلاک write میکنیم. در این صورت محتویات بلاک های اختصاص یافته به آن فایل را نیز پاک کرده ایم.

۲-۵ برگرداندن descriptor file ها به حالت اول هنگام unmount کردن

این ویژگی از پیش در پروژه موجود بوده است. در تابع umount در انتهای کد این تابع یک for روی fd_table زده شده است و همه descriptor file ها را به حالت اولیه و init خود بر میگرداند. فی الواقع نیاز به تغییر کد نیست و این ویژگی از قبل پیاده سازی شده است.

۶-۲ استفاده از لاک برای کنترل تغییرات همزمان چندین کاربر

وقتی چندین کاربر بخواهند همزمان روی سیستم تغییر اعمال کنند، ممکن است منجر به بروز رفتارهای پیش‌بینی نشده در سیستم شود. به همین دلیل ما با استفاده از mutex و استفاده کردن از توابع wait و post در جاهای موردنیاز رفتار درست سیستم را تضمین می‌کنیم. برخی از جاهایی که استفاده از لاک ضروری است، عبارت است ازfs_create، fs_write، fs_delete و ...