project3

تمرین گروهی ۳ - مستند طراحی

گروه

رضا یاربخش ryarbakhsh@gmail.com
علیرضا طباطبائیان tabanavid77@gmail.com
روزبه مشکین نژاد rouzyd@gmail.com
علی بهجتی bahjatia@gmail.com

مقدمات

اگر نکات اضافه‌ای در مورد تمرین یا برای دستیاران آموزشی دارید در این قسمت بنویسید.

لطفا در این قسمت تمامی منابعی (غیر از مستندات Pintos، اسلایدها و دیگر منابع درس) را که برای تمرین از آن‌ها استفاده کرده‌اید در این قسمت بنویسید.

بافر کش

داده‌ساختارها و توابع

در این قسمت تعریف هر یک از `struct`ها، اعضای `struct`ها، متغیرهای سراسری یا ایستا، `typedef`ها یا `enum`هایی که ایجاد کرده‌اید یا تغییر داده‌اید را بنویسید و دلیل هر کدام را در حداکثر ۲۵ کلمه توضیح دهید.

در فایل inode.h:

```
struct list LRU_list;
struct lock LRU_lock;

struct cache_block{
  bool is_dirty;
  block_sector_t sector;
```

```
  struct list_elem elem;
  struct lock cb_lock;

  void* block_data[BLOCK_SECTOR_SIZE];
};

void cached_block_read(
  struct block *block, block_sector_t sector, void *buffer,
    int offset, int length);

void cached_block_write(
  struct block *block, block_sector_t sector, const void *buff
er,
    int offset, int length);

bool free_LRU_block();

void free_all_cache();
```

برای این بخش، برای جابجایی بلاک‌های cache از LRU استفاده خواهد شد. برای پیاده‌سازی این replacement policy از یک لیست به نام `LRU_list` استفاده می‌کنیم که در front یا ابتدای آن داده‌هایی هستند که اخیرا به آن‌ها دسترسی داشته‌ایم و در back یا انتهای آن داده‌هایی هستند که چند وقتی بدون استفاده مانده‌اند.

- هر node از این list یک cache_block است. این cache_block شامل شماره sector، کثیف یا dirty بودن و یک قفل می‌شود.
- تابع `cached_block_read` از روی block مشخص شده (دستگاه فایل سیستم)، sector داده شده را می‌خواند و در cache ذخیره می‌کند (اگر در cache نباشد). بعد از `sector+offset` تا `sector+offset+length` روی `buffer` تا `buffer+length` کپی می‌شود.
- تابع `cached_block_write` ابتدا sector خواسته شده را در صورت نبودن در cache، از block (دستگاه فایل سیستم) می‌خواند و به cache می‌آورد. بعد از `buffer` تا

`buffer+length` را روی `sector+offset` تا `sector+offset+length` کپی می‌کند و `is_dirty` را برای آن 1 می‌کند.
- تابع `free_LRU_block` آخرین cache_block موجود در LRU_list را در صورت dirty بودن روی disk می‌نویسد و آن را از لیست حذف می‌کند.
- تابع `free_all_cache` در `filesys_done` صدا زده می‌شود و تمام cache_blockهای dirty را روی disk می‌نویسد.

الگوریتم‌ها

> *توضیح دهید که الگوریتم مورد استفاده‌ی شما به چه صورت یک بلاک را برای جایگزین شدن انتخاب می‌کند؟*
> *روش پیاده‌سازی `read-ahead` را توضیح دهید.*

به طور کلی، دو تابع `cached_block_read` و `cached_block_write` به ترتیب جایگزین تمام referenceها به توابع `block_read` یا `block_write` می‌شوند. در این دو تابع ابتدا در لیست‌های cache برای شماره sector خواسته شده جستجو می‌شود. برای این کار، ابتدا قفل مربوط به لیست (LRU_lock) گرفته می‌شود. اگر cache_block مورد نظر پیدا شد، آن block به ابتدای LRU_list اضافه می‌شود و بعد LRU_lock آزاد می‌شود. بعد قفل مربوط به cache_block گرفته و عملیات خواندن و نوشتن روی cache_block انجام می‌شود.

اگر cache_block مورد نظر پیدا نشود، در صورت پر بودن cache، ابتدا باید cache به اندازه یک block خالی شود. بنابراین سعی می‌کنیم cache_block انتهایی را بیرون کنیم. برای این کار، تابع `free_LRU_block` صدا زده می‌شود. (همچنان LRU_lock را داریم.) این تابع ابتدا سعی می‌کند cb_lock مربوط به cache_block انتهایی را بگیرد (تا مطمئن شویم در حال خواندن یا نوشتن روی آن نیستیم) و بعد محتوای cache_block را در صورت dirty بودن روی disk بنویسد. بعد از نوشتن محتوای cache_block انتهایی روی disk، قفلِ cb_lock آزاد می‌شود و کار تابع تمام می‌شود. بعد از اتمام تابع `free_LRU_block` می‌توانیم یک cache_block جدید را به LRU_list اضافه کنیم.

برای اضافه کردن یک cache_block جدید نیز ابتدا LRU_lock را می‌گیریم (اگر تابع `free_LRU_block` را صدا زده باشیم هنوز آن را داریم) و cache_block جدید را به ابتدای لیست اضافه می‌کنیم. بعد LRU_lock را آزاد می‌کنیم و cb_lock را می‌گیریم تا بتوانیم داده مورد نیاز را از روی disk بخوانیم و روی block_data بنویسیم. بعد از این که خواندن داده تمام شد، cb_lock را آزاد می‌کنیم.

زمانی که سیستم shut_down می‌شود، تابع filesys_done صدا زده می‌شود. در این تابع، `free_all_cache` را صدا می‌زنیم تا تمام cache_blockهای dirty را به disk منتقل کند.

- پیاده سازی `read-ahead`:
برای پیاده‌سازی read-ahead کافیست هر بار که تابع `inode_read_at` صدا زده می‌شود، تابع دیگری مثل `cached_block_read_ahead` صدا زده شود، که بلوک‌های بعدی data را می‌گیرد و یک thread جدید می‌سازد که تابع اجرایی آن، `cached_block_read` است.
آرگومان‌های تابع اجرایی نیز بلوک‌های بعدی data هستند. البته این تابع باید بعد از `cached_block_read` برای data خواسته شده صدا زده شود تا اول داده مورد نیاز تامین شود و بعد داده‌های مربوط به read-ahead خوانده شود.

- پیاده‌سازی `write-behind`:
برای پیاده‌سازی write-behind کافیست یک ترد بسازیم که در آن کد زیر اجرا می‌شود:

```
while(true)
{
  write_dirty_blocks_to_disk();
  timer_sleep(100);
}
```

همگام سازی

> *هنگامی که یک پردازه به طور مستمر در حال خواندن یا نوشتن داده در یک بلاک بافرکش می‌باشد به چه صورت از دخالت سایر پردازه‌ها جلوگیری میشود؟*
> *در حین خارج شدن یک بلوک از حافظه‌ی نهان، چگونه از پروسه‌های دیگر جلوگیری می‌شود تا به این بلاک دسترسی پیدا نکنند؟*

برای هر cache_block یک قفل تعبیه شده است. بنابراین هر عملی اعم از خواندن، نوشتن یا خارج کردن تنها به یک process محدود می‌شود.
زمانی که می‌خواهیم یک cache_block را از LRU_list خارج کنیم، ابتدا قفل مربوط به آن را می‌گیریم. به این ترتیب process دیگری نمی‌تواند روی آن خواندن یا نوشتن انجام دهد.
همچنین در این حین، کل لیست قفل شده و processهای دیگر نمی‌توانند حتی برای این block جستجو کنند.

منطق طراحی

> *یک سناریو را توضیح دهید که بافر کش، از `read-ahead` و یا از `write-behind` استفاده کند.*

به عنوان مثال اولین sector از یک فایل که در 10 بخش است را می‌خوانیم. در ادامه می‌خواهیم به بقیه فایل دسترسی داشته باشیم ولی در این بین، ترد مربوط به `read-ahead` کمی کار کرده و در نتیجه sector دوم از فایل در cache موجود است. به این ترتیب، زمانی که در حال پردازش اطلاعات اولین sector بوده‌ایم، sector دوم به شکل read-ahead لود شده و در زمان صرفه جویی کرده‌ایم.

فایل‌های قابل گسترش

داده‌ساختارها و توابع

> *در این قسمت تعریف هر یک از `struct` ها، اعضای `struct` ها، متغیرهای سراسری یا ایستا، `typedef` ها یا `enum` هایی که ایجاد کرده‌اید یا تغییر داده‌اید را بنویسید و دلیل هر کدام را در حداکثر ۲۵ کلمه توضیح دهید.*

ابتدا برای پشتیبانی از ساختار چندسطحی، باید موارد زیر را (به جای `start`) به ساختار `inode` اضافه کنیم:

```
block_sector_t direct[16]
block_sector_t indirect
block_sector_t double_indirect
```

برای این که اندازه این ساختار را تغییر ندهیم، باید تعداد درایه‌های `unused` را به 107تا تغییر دهیم:

```
uint32_t unused[107]
```

در نهایت برای جلوگیری از race در خواندن و نوشتن، قفل زیر را در ساختار `file` میگذاریم:

```
struct lock *file_lock
```

با توجه به ساختار چندسطحی، نیاز داریم که توابع `free_map_allocate` و `free_map_release` را به صورتی تغییر دهیم که نیازی به گرفتن سری sectorها نداشته‌باشند.
همچنین نیاز است که تابع `file_write` طوری تغییر کند که اگر به EOF رسید، فایل را گسترش دهد.
برای جلوگیری از race در free_map، متغیر زیر را اضافه می‌کنیم:

```
static struct lock *free_map_lock
```

برای پشتیبانی از `SYS_INUMBER`، کافی است تابع `get_file_safe` را صدا زده و سپس به کمک ساختار `inode` موجود در ساختار `file` بدست‌آمده، تابع `inode_get_inumber` را صدا می‌زنیم.

> *بیشترین سایز فایل پشتیبانی شده توسط ساختار inode شما چقدر است؟*

بیشترین اندازه فایل کمی بیشتر از 8MB است و بنابراین محدودیت ذکرشده در صورت پروژه را رعایت می‌کند.

همگام سازی

> *توضیح دهید که اگر دو پردازه بخواهند یک فایل را به طور همزمان گسترش دهند، کد شما چگونه از حالت مسابقه جلوگیری می‌کند.*
> *فرض کنید دو پردازه‌ی A و B فایل F را باز کرده‌اند و هر دو به end-of-file اشاره کرده‌اند.*
> *اگر همزمان A از F بخواند و B روی آن بنویسد، ممکن است که A تمام، بخشی یا هیچ چیز از اطلاعات نوشته شده توسط B را بخواند. همچنین A نمی‌تواند چیزی جز اطلاعات نوشته شده توسط B را بخواند. مثلا اگر B تماما ۱ بنویسد، A نیز باید تماما ۱ بخواند. توضیح دهید کد شما چگونه از این حالت مسابقه جلوگیری می‌کند.*
> *توضیح دهید همگام سازی شما چگونه "عدالت" را برقرار می‌کند. فایل سیستمی "عادل" است که خواننده‌های اطلاعات به صورت ناسازگار نویسنده‌های اطلاعات را مسدود نکنند و برعکس. بدین ترتیب اگر تعدادی بسیار زیاد پردازه‌هایی که از یک فایل می‌خوانند نمی‌توانند تا ابد مانع نوشته شدن اطلاعات توسط یک پردازه‌ی دیگر شوند و برعکس.*

این مشکلات با استفاده از قفل تعبیه‌شده در ساختار `file` پیش نخواهد آمد زیرا این قفل در هنگام خواندن و نوشتن می‌گیریم.

منطق طراحی

> *آیا ساختار `inode` شما از طبقه‌بندی چند سطحه پشتیبانی می‌کند؟ اگر بله، دلیل خود را برای انتخاب این ترکیب خاص از بلوک‌های مستقیم، غیر مستقیم و غیر مستقیم دوطبقه توضیح دهید.*
> *اگر خیر، دلیل خود برای انتخاب ساختاری غیر از طبقه‌بندی چند سطحه و مزایا و معایب ساختار مورد استفاده خود نسبت به طبقه‌بندی چند سطحه را توضیح دهید.*

در طراحی ما ساختار `inode` از طبقه‌بندی چندسطحه پشتیبانی می‌کند چرا که به این ترتیب می‌توان فایل‌های موجود را به سادگی گسترش داد. در این ترکیب خاص (با توجه به این‌که اندازه blockها برابر 512بایت است) در حالت غیرمستقیم 2طبقه 8MB، در حالت غیرمستقیم 64KB و در حالت مستقیم 8KB را پشتیبانی می‌کند، زیرا بیشترین تعداد فایل‌ها اندازه‌ای بین 128B تا 32KB دارند و در حالت مستقیم اکثر این فایل‌ها پشتیبانی می‌شوند. همچنین در حالت غیرمستقیم دوطبقه، 8MB پشتیبانی می‌شود که حداکثر اندازه فایل در این قسمت از پروژه است.

زیرمسیرها

داده‌ساختارها و توابع

> *در این قسمت تعریف هر یک از `struct`ها، اعضای `struct`ها، متغیرهای سراسری یا ایستا، `typedef`ها یا `enum`هایی که ایجاد کرده‌اید یا تغییر داده‌اید را بنویسید و دلیل هر کدام را در حداکثر ۲۵ کلمه توضیح دهید.*

در فایل `node.c` به `struct inode_disk` موارد زیر را اضافه میکنیم:
`bool is_dir` : جهت مشخص کردن پوشه بودن یک inode
`block_sector_t parent` : جهت نگهداری پدر یک inode
به `struct thread` در `thread.c` جهت نگهداری پوشه کار فعلی `struct dir *cwd` را اضافه میکنیم.
به `struct dir` در `directory.c` جهت جلوگیری از مسابقه `struct lock dir_lock` را اضافه میکنیم.
به دو تابع `inode create` و `filesys_create` جهت تشخیص پوشه بودن `bool is_dir` را اضافه میکنیم.

الگوریتم‌ها

> *کد خود را برای طی کردن یک مسیر گرفته‌شده از کاربر را توضیح دهید.*
> *آیا عبور از مسیرهای **absolute** و **relative** تفاوتی دارد؟*

ابتدا حرف اول را بررسی میکنیم در صورتی که / بود از root شروع میکنیم و در غیر اینصورت از cwd شروع میکنیم. سپس حروف را تا زمانی که به / برسیم خوانده و ادامه میدهیم. در صورتی که پوشه ای با این نام وجود داشت آن را پیمایش کرده , در صورتی که .. بود به سمت پوشه پدر رفته و در صورتی که نام فایل بود بایستی آخرین قسمت آدرس باشد.

همگام سازی

> *چگونه از رخ دادن **race-condition** در مورد دایرکتوری ها پیشگیری می‌کنید؟*
> *برای مثال اگر دو درخواست موازی برای حذف یک فایل وجود داشته باشد و تنها یکی از آنها باید موفق شود یا مثلاً دو ریسه موازی بخواهند فایلی یک اسم در یک مسیر ایجاد کنند و مانند آن.*

با اضافه شدن قفل به `struct dir` کارهایی مثل حذف فایل و یا اضافه کردن فایل به هر پوشه را امن ریسه میکنیم.

> *آیا پیاده سازی شما اجازه می‌دهد مسیری که **CWD** یک ریسه شده یا پردازه‌ای از آن استفاده می‌کند حذف شود؟ اگر بله، عملیات فایل سیستم بعدی روی آن دایرکتوری چه نتیجه‌ای می‌دهند؟ اگر نه، چطور جلوی آن را می‌گیرید؟*

هنگامی که `cwd` یک ریسه را تعیین میکنیم, پوشه را باز میکنیم و `open_cnt` آن یکی زیاد میشود. در صورتی که پوشه‌ای `open_cnt` بیشتر از 0 داشته باشد اجازه حذف آن را نمیدهیم.

منطق طراحی

> *توضیح دهید چرا تصمیم گرفتید **CWD** یک پردازه را به شکلی که طراحی کرده‌اید پیاده‌سازی کنید؟*

به جای استفاده از `inode` برای `cwd` از `dir` استفاده کردیم. چرا که استفاده از api برای `dir` ساده تر است و باز و بسته کردن آن ساده تر از `inode` میباشد.

سوالات نظرسنجی

پاسخ به این سوالات دلخواه است، اما به ما برای بهبود این درس در ادامه کمک خواهد کرد.
نظرات خود را آزادانه به ما بگوئید—این سوالات فقط برای سنجش افکار شماست.
ممکن است شما بخواهید ارزیابی خود از درس را به صورت ناشناس و در انتهای ترم بیان کنید.

به نظر شما، این تمرین گروهی، یا هر کدام از سه وظیفه آن، از نظر دشواری در چه سطحی بود؟ خیلی سخت یا خیلی آسان؟

چه مدت زمانی را صرف انجام این تمرین کردید؟ نسبتا زیاد یا خیلی کم؟

آیا بعد از کار بر روی یک بخش خاص از این تمرین (هر بخشی)، این احساس در شما به وجود آمد که اکنون یک دید بهتر نسبت به برخی جنبه‌های سیستم عامل دارید؟

آیا نکته یا راهنمایی خاصی وجود دارد که بهتر است ما آنها را به توضیحات این تمرین اضافه کنیم تا به دانشجویان ترم های آتی در حل مسائل کمک کند؟

متقابلا، آیا راهنمایی نادرستی که منجر به گمراهی شما شود وجود داشته است؟

آیا پیشنهادی در مورد دستیاران آموزشی درس، برای همکاری موثرتر با دانشجویان دارید؟

این پیشنهادات میتوانند هم برای تمرین‌های گروهی بعدی همین ترم و هم برای ترم‌های آینده باشد.

آیا حرف دیگری دارید؟