به نام خدا

# پروژه‌های درس سیستم‌های عامل

نیم سال اول ۰۱ – ۰۲

دانشکده‌ی مهندسی کامپیوتر

دانشگاه صنعتی شریف

---

دکتر حسین اسدی

با سپاس از تیم دستیاران آموزشی

## نکات مهم:

- پرسش و پاسخ برای تمامی پروژه‌ها **صرفاً** از طریق صفحه‌ی مربوط به پروژه از طریق CW انجام می‌پذیرد.
- گزارش پروژه بایستی در فرمت لاتک و در سامانه لاتک آنلاین دانشگاه نوشته شود. لذا یکی از نفرات پروژه بایستی قالب گزارش پروژه‌ی درس را از لینک ذیل انتخاب کرده و در این قالب گزارش تیم پروژه را ایجاد نماید. پروژه بایستی بین اعضای گروه به اشتراک گذاشته شود و تمامی اعضای گروه بایستی در نوشتار مشارکت نمایند. دقت شود تاریخچه مشارکت اعضای گروه در سامانه قابل رویت توسط دستیار آموزشی خواهد بود. گزارش پروژه درس بایستی با سرگروه مربوطه (آقای معینی جم یا آقای فراهانی) و خانم ادیبی در سامانه به اشتراک گذاشته شود:

e.adibi@sharif.edu, me.moeinijam@sharif.edu, ali.farahani@sharif.edu

آدرس ورود به سامانه لاتک دانشگاه:

https://latex.sharif.edu/login

قالب گزارش پروژه درس:

https://latex.sharif.edu/templates/63a98879ab56c300a159ea1e

- تعداد نفرات پروژه ۲ الی ۳ نفر می‌تواند باشد.
- هر پروژه نهایتاً توسط سه گروه قابل انجام است. لذا اولویت تخصیص با تیمی است که زودتر پروژه‌ی خود را انتخاب و توضیحات تک صفحه‌ی مربوطه را در صفحه‌ی درس بارگذاری نماید.
- هر گروه بایستی حداکثر تا تاریخ ۲۲ دی‌ماه تیم پروژه و توصیف مختصر پروژه را در قالب یک فایل pdf تک صفحه‌ای در صفحه درس بارگذاری نماید.
- بارگذاری تمامی مستندات بایستی توسط نماینده تیم پروژه انجام شود.
- مهلت انجام پروژه تا ۱۰ بهمن می‌باشد.
- تمامی پروژه‌ها تحویل حضوری دارند.
- توصیه می‌شود شروع پروژه را به روزهای آخر موکول نفرمایید و در اسرع وقت کارهای اولیه پروژه را شروع نمایید.
- **تذکر بسیار مهم:**
  - استفاده از کدهایی که ممکن است در اینترنت بیابید، مجاز نیست و شباهت کدهای شما با کدهای آماده یا کدهای سایر دانشجوهای درس، به منزله‌ی تقلب و ثبت نمره‌ی **صفر** خواهد بود.

**موفق و سربلند باشید**

## (پروژه‌های مربوط به گروه آقای معینی جم)

## پروژه‌ی اول

### عنوان:

پیاده سازی ویژگی‌های پیشرفته برای یک فایل سیستم

### مقدمه:

در این درس با مفاهیم اولیه فایل سیستم‌ها آشنا شدید. هدف این پروژه اضافه کردن مجموعه‌ای از ویژگی‌های پیشرفته‌تر به یک فایل سیستم ساده است که با زبان C پیاده سازی شده است. در انجام این پروژه شما با فایل سیستم، نحوه‌ی نوشتن و خواندن فایل‌ها از حافظه، و مفاهیم paging بیشتر آشنا می‌شوید.

### شرح پروژه:

کد فایل سیستم ابتدایی در این لینک آورده شده است. این فایل سیستم یک پیاده‌سازی ساده شده از فایل سیستم FAT32 است. شما باید در ابتدا کد داده شده را ران کنید و بخش‌های مختلف آن را متوجه شوید و سپس مزایا و معایب آن را نسبت به پیاده سازی اصلی بیان کنید.

فایل سیستم پیاده‌سازی شده در این لینک شامل چهار بخش زیر است.

1. Mounting/Un-mounting
2. File Creation/Deletion
3. File Descriptor Operations
4. File Reading/Writing

که بخش‌های ضروری یک فایل سیستم هستند و پیاده‌سازی فایل سیستم به این هفت تابع ساده‌سازی شده است.

### مراحل انجام پروژه:

- شما مختارید که هر تابع و ساختار دلخواهی به فایل fs.c اضافه کنید تا ویژگی‌های لیست شده در زیر را به کد اضافه کنید.
- می‌خواهیم فایل سیستم این قابلیت را داشته باشد که به کاربر اجازه ندهد فایل‌های مهم را تغییر دهند و یا آنها را پاک کنند. برای این کار می‌خواهیم هر زمان کاربر درخواست تغییر فایل‌هایی را داد که اسم آنها با کلمه 'lock' شروع می شوند را دهد، یک خطا رخ دهد.
- در قسمت writing این برنامه هنگام نوشتن یک فایل برای سادگی از سیستم اشتباهی برای محاسبه‌ی offset استفاده می‌شود. نحوه محاسبه‌ی offset کنونی در کد را توضیح دهید و آن را به حالت استاندارد اصلاح کنید.
- در این فایل سیستم حالتی که چندکاربر به صورت همزمان بخواهند تغییری در سیستم ایجاد کنند که ممکن است سبب ایجاد ناهمانگی در سیستم و یا تغییرات اشتباه در اطلاعات ما شود، بررسی نشده است. به عنوان مثال یک فایل را تغییر دهند و یا فایل‌هایی با یک نام تولید کنند. به منظور رفع این مشکل باید با استفاده از سمافور مکان‌های مورد نیاز برای لاک کردن را تشخیص دهید و آنها را پیاده سازی کنید.
همچنین در ادامه لازم است با ساختن یک فایل تست و اجرا کردن موارد آن روی چند ترد موارد ذکر شده را امتحان کنید و صحت کد خود را ارزیابی کنید.

- یکی دیگر از ساده‌سازی‌هایی که در این پیاده‌سازی اعمال شده این است که محتویات فایل‌ها به محض ساخته شدن در دیسک مجازی نوشته نمی‌شوند و تنها زمانی که داریم فایل را می‌نویسیم محتویات داخل دیسک آورده می‌شوند. در این قسمت شما باید کد را طوری تغییر دهید که در هنگام ساخت یک فایل محتویات آن درون دیسک مجازی ذخیره شود و در هنگام پاک کردن فایل این اطلاعات از داخل دیسک حذف شود.
- زمانی که قصد داریم فایلی را unmount کنیم تمام دستورات خواندن و نوشتنی که منتظر اجرا هستند باید خاتمه بیابند در نتیجه کد را طوری تغییر دهید که در زمان اجرای unmount، اطلاعات نوشته شده در file descriptor مربوط به فایل ذکر شده به حالت اولیه بازگردانده شود.
- در این قسمت باید حافظه نهان را در فایل سیستم خود پیاده‌سازی کنید. بهترین مکان برای قرار دادن کد این قسمت پس از رابط read/write همگام است. توجه داشته باشید که سایر بخش‌های فایل سیستم باید کمترین میزان تغییر را داشته باشند یا اصلا تغییر نکنند.

شما باید عملیات خواندن و نوشتن را برای حافظه نهان به نحوی پیاده‌سازی کنید که هر کدام یک سوم بلوک‌های فایل سیستم را در بر بگیرند. عملیات خواندن از حافظه نهان باید به این صورت باشد که ابتدا بلوک مد نظر در حافظه نهان جستجو شود و در صورت یافت نشدن آن، از دیسک خوانده شود. برای جایگزینی بلوک‌ها نیز سیاست دلخواهی (برای مثال LRU) را پیاده‌سازی کنید. در نوشتن بر روی دیسک نیز بلوک مورد نظر باید در صورت وجود فضا در حافظه نهان، آن بلوک را در حافظه نهان بنویسد و در غیر این صورت عملیات نوشتن همگام را اجرا کند. در اینجا نیازی به سیاستی برای جایگزینی بلوک‌ها نیست، اما باید نوشتن dirty block ها در دیسک صورت بگیرد. همچنین پیاده‌سازی یک تابع برای فلاش کردن حافظه نهان نیاز است، به این صورت که همه برنامه‌ها باید پیش از خارج شدن این تابع را فراخوانی کنند.

همچنین، توجه داشته باشید که باید برنامه را طوری بنویسید که هم بتوان آن را با استفاده از حافظه‌ی نهان و هم بدون آن اجرا کرد. در انتها باید یک فایل تست برای این پیاده‌سازی ایجاد کنید که دستورات خواندن و نوشتن را یکبار با استفده از این حافظه و یکبار بدون آن اجرا می‌کند. زمان اجرای این دستورات در دو نوع گفته شده را با هم مقایسه کنید.

### لینک های مفید:

از آنجایی که فایل سیستم پیاده‌سازی شده یک نسخه ساده شده از فایل سیستم مبنی بر File Allocation Table است بهتر است قبل از شروع این لینک را مطالعه کرده و با نحوه‌ی پیاده‌سازی این فایل سیستم آشنا شوید.

همچنین، در این لینک توضیحی کلی از نحوه‌ی نوشته شدن این پروژه و ساختار فایل سیستم پیاده سازی شده آورده شده است. بهتر است برای شروع ابتدا به لینک ذکر شده مراجعه کنید تا با بخش‌های مختلف فایل سیستم و عملکردهای آن بیشتر آشنا شوید.

## پروژه‌ی دوم

### عنوان:

پیاده‌سازی یک مفسر خط فرمان مشابه با Unix Shell

### مقدمه:

هدف از این پروژه پیاده سازی یک مفسر خط فرمان یا Shell است. برنامه‌ای که قرار است پیاده سازی کنید، به شکل پایه به این صورت کار می‌کند که شما یک دستور را تایپ کرده (یا در جواب به خواسته Shell عبارتی را تایپ می‌کنید) و Shell با ساختن یک پروسه فرزند دستوری که تایپ شده را اجرا کرده و بعد از اجرای آن منتظر دستور بعدی می‌ماند. به بیان دیگر باید نسخه‌ای مشابه ولی بسیار ساده‌تر از Shell ای که در سیستم‌عامل‌های لینوکسی وجود دارد را پیاده‌سازی کنید.

### شرح پروژه:

برنامه شما باید به دو شکل تعاملی (Interactive) یا دسته‌جمعی (Batch) قابل اجرا باشد. در حالت تعاملی، از کاربر ورودی خواسته شده و متناسب با دستور وارد شده کاری که باید انجام می‌شود. در حالت دسته جمعی یک فایل batch در هنگام اجرای برنامه Shell مشخص می‌شود که لیستی از دستوراتی که باید اجرا بشوند در آن نوشته شده و با دریافت این فایل، Shell بدون نمایش بخشی برای ورود دستور از سمت کاربر به اجرای این دستوارت می‌پردازد. در این حالت باید پیش از اجرای هر دستور، دستوری که در حال اجرا است به کاربر نمایش داده شود. برنامه Shell با رسیدن به دستور quit یا انتهای ورودی (انتهای فایل یا وارد شدن Ctrl+D توسط کاربر) پایان می‌پذیرد.

هر خط می‌تواند شامل چندین دستور باشد که با ; از یکدیگر جدا شده‌اند. دستوراتی که در یک خط با ; از هم جدا شده‌اند باید به صورت همروند اجرا بشوند. توجه کنید که این رفتار با رفتار رایج Shell های لینوکسی که دستورات را یک به یک اجرا می‌کنند متفاوت است. هنگام اجرای این دستورات همروند Shell نباید دستور جدیدی از کاربر یا فایل دریافت کند باید تا اتمام تمامی دستوراتی که در یک خط بوده‌اند صبر کند (سیستم‌کال‌های wait یا waitpid می‌تواند در این بخش مفید باشد).

به عنوان مثال

```
prompt> ls -l ; cat file ; grep foo file2
```

در این حالت هر سه دستور ls و cat و grep با هم به صورت هم‌روند اجرا می‌شوند و مخلوط شدن خروجی نهایی آن‌ها در هنگام نمایش به کاربر هم امری طبیعی است و نیازی به جلوگیری از آن ندارید.

نکته دیگر این است که دستور quit یک دستور درونی shell شما خواهد بود و قرار نیست مانند باقی دستورات به صورت یک برنامه مجزا در یک پروسه فرزند اجرا شود. بلکه مشاهده آن باید به اجرای shell خاتمه بدهد.

دستوراتی که در خود shell وارد می‌شوند برنامه‌های رایج لینوکسی هستند که از پیش وجود دارند و نیازی به پیاده سازی آن‌ها ندارید و صرفا باید آن‌ها اجرا بشوند.

### نحوه‌ی اجرای برنامه:

برنامه شما باید با این دستور اجرا بشود:

```
shell [batchFile]
```

که در آن batchFile که آرگومان اختیاری است. در صورت وجود آن، فایل مشخص شده به صور batch اجرا می‌شود و در غیر این صورت shell به صورت تعاملی اجرا می‌شود.

### نکات ضروری:

این نوع برنامه‌های سیستمی اساسی باید سعی کنند تا حد امکان به هیچ وجه دچار اشکال و خرابی نشوند. در نتیجه باید طوری برنامه خود را بنویسید که تحت هیچ شرایطی مشکلاتی نظیر Core Dump یا گیر کردن در Loop بی‌نهایت و مواردی نظیر آن رخ ندهد. خطاهای مختلف هم باید به شکلی معقول رسیدگی بشوند. بخشی از خطاهایی که ممکن است با آن‌ها رو به رو بشوید به شرح زیر هستند:

- تعداد آرگومان‌هایی که به در هنگام اجرا داده شده دست نباشند. مثلا در هنگام اجرا دو فایل به shell داده بشوند.
- batchFile مشخص شده وجود نداشته باشد.
- دستوری که در shell وارد شده وجود نداشته باشد.
- اندازه کل دستور وارد شده بیش از ۵۱۲ کاراکتر باشد.

در دو حالت اول باید برنامه شما با نمایش پیام مناسب در stderr متوقف شود. در حالت سوم و چهارم باید پیام خطای مناسبی چاپ شده و ادامه برنامه انجام بشود.

همچنین توجه کنید حالت‌هایی نظیر خط خالی یا Space به تعداد بالا در بین آرگومان‌های ورودی دستورات ممکن است رخ بدهند که خطا نیستند و نیازی به چاپ پیام خطا برای آنان نیست ولی باید به درستی هندل شده و خللی در اجرای برنامه ایجاد نکنند. مثلا خط زیر معتبر است و باید به درستی اجرا بشود.

```
prompt> ;;;; cat   file ; ;;;; grep    foo     file2
```

توجه کنید که به هر حال تحت هیچ شرایطی خطاها نباید منجر به متوقف شدن Shell شما بشوند.

### لینک‌های مفید:

مطالعه لینک زیر برای آشنایی با نحوه پیاده‌سازی یک Shell می‌تواند مفید باشد:

https://brennan.io/2015/01/16/write-a-shell-in-c/

همچنین مطالعه این لینک برای اطلاع در مورد پیشینه و تحول Shell در لینوکس توصیه می‌شود:

https://developer.ibm.com/tutorials/l-linux-shells/

## پروژه‌ی سوم

### عنوان:

پیاده‌سازی یک تخصیص‌گر حافظه‌ی Heap مشابه با malloc

### مقدمه:

در این پروژه شما باید یک تخصیص‌گر حافظه Heap برای پردازه‌های سطح کاربر بنویسید. توابعی که پیاده‌سازی می‌کنید مشابه malloc و free خواهند بود.

تخصیص‌گرهای حافظه دو وظیفه مهم دارند. اول این که آن‌ها از سیستم‌عامل می‌خواهند که قسمت heap فضای آدرسی پروسه را گسترش بدهد. این کار از طریق دستورات sbrk یا mmap انجام می‌شود. علاوه بر این، وظیفه تخصیص حافظه و پیدا کردن قسمت بهم‌پیوسته‌ای از حافظه تخصیص یافته که به اندازه کافی برای درخواست برنامه بزرگ باشد هم برعهده تخصیص‌گرهای حافظه است. همچنین بعد از آزاد شدن هم مدیریت این که این فضاهای آزاد در لیستی قرار داشته باشند برعهده تخصیص‌گرهای حافظه است.

این نوع تخصیص‌گرهای حافظه معمولا در قالب Standard Library ارائه شده و جزوی از خود سیستم‌عامل نیستند. به طور دقیق‌تر، تخصیص‌گرهای حافظه در فضای آدرسی یک برنامه فعالیت می‌کنند و اطلاعی از صفحات فیزیکی یا نگاشت بین آدرس‌های منطقی و فیزیکی ندارند.

برای پیاده سازی این قابلیت‌ها در پروژه خود، بهتر است این روش کار را مدنظر داشته باشید. ابتدا، هنگام درخواست حافظه از سیستم عامل، باید از mmap استفاده کنید (کار با آن از sbrk راحت‌تر است). دوما، هر چند یک تخصیص‌گر حافظه واقعی هنگامی که نتواند درخواست کاربر را برآورده کند، مجددا رخواست حافظه می‌کند، چیزی که شما پیاده می‌کنید تنها یک بار باید این کار را انجام دهد و نیازی به تکرار عمل نیست. سوما، می‌توانید از هر داده‌ساختاری که می‌خواهید برای مدیریت قسمت‌های آزاد حافظه استفاده کنید. با این حال باید برنامه‌شما از نظر پرفرمنس رفتار معقولی داشته باشد.

حال ابتدا به رفتار malloc و free عادی توجه کنید.

- تابع void* malloc(size_t size) : این تابع به اندازه size بایت فضا اختصاص داده و اشاره‌گری به ابتدای فضای داده شده بر می‌گرداند. مقادیری که از قبل در آن قسمت حافظه به تصادف یا از قبل باقی‌مانده‌اند، تغییری نمی‌کنند.
- تابع void free(void* ptr) : این تابع فضایی که با ptr به آن شاره می شود را آزاد می‌کند تا بعدا بتواند مورد استفاده قرار بگیرد. باید توجه کرد که ptr باید پوینتری باشد که پیش‌تر توسط یکی از توابع malloc یا calloc یا realloc بازگردانده شده باشد.

توابعی که شما باید پیاده‌سازی کنید، Mem_Alloc(int size) و Mem_Free(void* ptr) نام دارند و تقریبا مشابه رفتار عادی آن‌ها هستند. با این تفاوت که ptr ای که به Mem_Free داده می‌شود، لزوما نیازی نیست که خروجی Mem_Alloc باشد و می‌تواند اشاره‌گری به هر بخش معتبری از حافظه که به برنامه داده شده است باشد. به عنوان مثل تکه کد زیر در برنامه شما باید به درستی کار کند ولی با توابع عادی، این تکه کد رفتار درستی نخواهد داشت.

```
int *ptr;
// The returned memory object is between ptr and ptr+100
if ((ptr = (int *)Mem_Alloc(100 * sizeof(int))) == NULL) exit(1);
// Could replace 30 with any value from 0 to 99..
Mem_Free(ptr+30);
```

در نتیجه، در پیاده سازی شما باید داده ساختاری پیچیده‌تر از حالت معمولت malloc دا شته با شید که ناحیه‌هایی از حافظه که تو سط Mem_Alloc ارائه شده‌اند (و نه لزوما فقط ابتدای بازه) را در نظر بگیرید. به طور خاص، این داده ساختار به شما این امکان را خواهد داد که به طور موثر، هر آدرسی را به memory object نظیر آن که کل بازه را در بر گرفته است نگاشت کنید. نحوه پیاده‌سازی داده‌ساختار با شما خواهد بود. به طور کلی باید دوو تابع Mem_GetSize(void* ptr) و Mem_IsValid(void *ptr) را هم پیاده سازی کنید که در ادامه توضیح داده خواهند شد و ptr در آن‌ها می‌تواند به هر قسمتیت از memory object اشاره کند.

این پروژه به طور خلاصه، اهداف زیر را دنبال می‌کند:

۱. درک نکات ریز پیاده‌سازی تخصیص‌گر حافظه

۲. درک نکات مربوط به تنظیم پرفرمنس برای Workload های مختلف

۳. تقویت مهارت‌های برنامه‌نویسی سیستمی به کمک mmap و mprotect

۴. ایجاد یک Shared Library

## شرح پروژه:

برای این پروژه باید چندین تابع مختلف را که بخشی از یک shared library هستند پیاده‌سازی کنید. توجه کنید که قرار نیست که تابع main را در این shared library بنویسید، بلکه باید توابعی که در زیر آمده را پیاده سازی کرده و بعدا برای تست در یک فایل دیگر که تابع main دارد، آن‌ها را تست کنید. Prototype تمامی توابع در فایل mem.h همراه این پروژه داده شده است. این فایل را به هیچ وجه ویرایش نکنید.

توابعی که باید پیاده‌سازی کنید بدین شرح هستند:

- تابع int Mem_Init(int sizeOfRegion) : این تابع تنها یک بار توسط پردازه اجرا می‌شود. sizeOfRegion تعداد بایت‌هایی است که باید از طریق دستور mmap از OS درخواست کنید. توجه کنید که احتمالا باید این مقدار را به بالا گرد کنید تا مقداری که درخواست می‌کنید مضربی از اندازه صفحات باشد (برای این موضوع به تابع getpagesize() مراجعه کنید). توجه کنید که از این فضا برای داده‌ساختارهایی که برای مدیریت حافظه نیاز دارید استفاده خواهد شد.

- تابع void* Mem_Alloc(int size) : این تابع مشابه malloc است. ورودی آن سایز تعداد بایت‌هایی است که قرار است تخصیص داده شوند و در خروجی آدرس بایت اول object تخصیص داده شده را بر می‌گرداند. اگر فضای کافی در sizeOfRegion بایتی که توسط Mem_Init تخصیص داده شده وجود نداشته باشد، NULL بازگردانده می‌شود.

- تابع int Mem_Free (void* ptr) : این تابع memory object ای که ptr درون آخر قرار دارد را آزاد می‌کند. به بیان دیگر مثلا اگر یک فضای آدرسی ۱۰۰ بایتی توسط Mem_Alloc اختصاص داده شده باشد که ابتدای آن ptr باشد، در صورتی که ptr+10 به این تابع داده شود هم باید تمام این ۱۰۰ بایت آزاد بشود. در صورتی که ورودی آن NULL باشد هیچ عملیاتی اتفاق نمی‌افتد خروجی این تابع ۰ در صورت موفقیت و -۱ در صورتی است که آدرس داده شده در هیچ یک از فضاهای تخصیص یافته قرار نداشته باشد. توجه کنید که اگر آدرس قبلا آزاد شده باشد هم در عمل جزو فضاهای تخصیص نیافته است و در صورتی که به این تابع داده شود خروجی باید -۱ باشد.

- تابع int Mem_IsValid (void* ptr) : این تابع در صورتی که ptr در یکی از object های تخصیص یافته قرار داشته باشد ۱ و در غیر این صورت ۰ بر می‌گرداند.
- تابع int Mem_GetSize (void* ptr) : در صورتی که ptr در بازه‌ای که یک object تخصیص یافته قرار دارد، قرار داشته باشد این تابع مقدار اندازه آن آبجکت به بایت را بر می‌گرداند. در غیر این صورت -۱ بازگردانده می‌شود.

شما باید این توابع را پیاده‌سازی کرده و در یک shared library با نام libmem.so در اختیار قرار بدهید تا سایر برنامه‌نویسان بتوانند با لینک کردن کد خودشان با این فایل، از آن استفاده کنند.

با فرض این که کد شما در فایل mem.c باشد، برای ایجاد یک shared library به نام libmem.so باید دستورات زیر را اجرا کنید.

```
gcc -c -fpic mem.c
gcc -shared -o libmem.so mem.o
```

برای لینک کردن این کتابخانه با یک کد که قرار ا ست از آن ا ستفاده کند، باید نام آن را به شکل "-lmem" و محل پیدا کردن آن را به شکل "-L." مشخص کنید. عبارت جلوی L محل پیدا کردن فایل است و . یعنی در همان پوشه‌ای که قرار داریم به دنبال آن بگردد.

```
gcc mymain.c -lmem -L. -o myprogram
```

پیش از اجرا کردن برنامه myprogram باید یک environment variable به نام LD_LIBRARY_PATH هم تعریف کنید که مشخص کند کتابخانه مورد نظر باید از کجا پیدا شود. با فرض این که بخواهید برنامه را در همان پوشه اجرا کنید، باید این دستور را اجرا کنید:

```
gcc mymain.c -lmem -L. -o myprogram
```

**نمره‌دهی:**

برای این پروژه ۶۰ درصد نمره به پیاده‌سازی درست موارد خواسته شده اختصاص دارد. ۳۰ درصد به کیفیت کد و مواردی نظیر کارایی و توان اجرای تست‌ها و ۱۰ درصد هم به مستندات تعلق می‌گیرد.

برای تست برنامه سه پارامتر اندازه، ترتیب و وضعیت استفاده از اشاره‌گر مورد بررسی خواهند بود.

برای مقوله اندازه سه حالت داریم:

- تنها تعدادی شی با اندازه‌های کوچک بین ۸ تا ۲۵۶ بایت
- تعدادی آبجکت کوچک با سایز حدودا ۶۴ بایت و تعدادی آبجکت بزرگ با سایز حدودا ۶۴ کیلوبایت
- تعدادی آبجکت که اندازه همگی آن‌ها توان ۲ باشد.

برای مقوله ترتیب دو حالت داریم:

- کاربر N شی را ایجاد کرده و سپس همه آن‌ها را آزاد می‌کند. سپس N شی دیگر را ایجاد کرده و مجدد همه آن‌ها را آزاد می‌کند.
- کاربر N شی ایجاد کرده. سپس N/2 آن‌ها را آزاد کرده و از بقیه آن‌ها استفاده کرده و در انتها N/2 باقی‌مانده را آزاد می‌کند.

- برای مقوله وضعیت استفاده از اشاره‌گر دو حالت داریم:
- کاربر تنها از Mem_Free برای آزاد کردن همان پوینتری که توسط Mem_Alloc داده شده استفاده می‌کند و از بازه‌های درونی تخصیص داده شده استفاده نمی‌کند. همچنین از توابع Mem_IsValid و Mem_GetSize هم استفاده نمی‌کند.

- کاربر از Mem_Free برای آزاد کردن فضای تخصیص داده شده به هر شکلی استفاده کرده و لزوما دقیقا همان پوینتر خروجی Mem_Alloc را استفاده نمی‌کند. همچنین امکان استفاده از توابع Mem_IsValid و Mem_GetSize هم وجود دارد.

### مستندات:

علاوه بر کد، شما باید در یک فایل README منطق طراحی مورد ا ستفاده خود، علی‌الخ صوص داده ساختاری که ا ستفاده کرده اید را توضیح دهید. همچنین سیاست مورد استفاده برای تخصیص فضا (first-fit، best-fit و...) که مورد استفاده بوده باید توضیح داده شود. در این مستند باید هر نوع ابهامی که در فهم پروژه وجود داشته ذکر شده و فرضی که برای حل آن در نظر گرفته شده هم ذکر شد. همچنین اگر باگ یا مشکلی در کد شما وجود دارد که از وجود آن آگاه هستید، باید در این مستند ذکر بشود.

### لینک‌های مفید:

فایل mem.h در لینک زیر قابل دریافت است:

https://drive.google.com/file/d/1NCmWOpg1J6xVvEqFpFb3IopocUlvCPiV/view?usp=share_link

همچنین استفاده از لینک‌های زیر می‌تواند در انجام بهتر پروژه به شما کمک کند:

https://medium.com/@andrestc/implementing-malloc-and-free-ba7e7704a473

https://stackoverflow.com/questions/3479330/how-is-malloc-implemented-internally

https://www.cprogramming.com/tutorial/shared-libraries-linux-gcc.html

## پروژه‌ی چهارم

### عنوان:

پیاده‌سازی IO Cache برمبنای سیاست LFU

### مقدمه:

امروزه از حافظه‌های بر پایه فلش، مانند SSD، به عنوان cache برای دیسک‌های سخت که کندتر و با سایز بزرگ‌تر هستند، استفاده می شود. به این کار IO caching گفته می شود. بزرگترین مزیت IO caching آن ا ست که بدون ایجاد تغییر در زیر ساخت‌های سخت افزاری و تنها با نصب و سپس تعریف یک حافظه سریع‌تر به عنوان cache می‌توان سرعت و کارایی سیستم را افزایش داد.

توجه مهم: مطالعه و مرور مباحث cache از درس معماری کامپیوتر به شما در اجرای این پروژه کمک شایانی خواهد کرد، مطالبی مانند direct mapped cache, associative caches, …

### شرح پروژه:

در سیستم عامل لینوکس، ابزارهای متن باز زیادی برای ایجاد (تعریف) کش وجود دارد. یکی از شناخته شده‌ترین این ابزارها، EnhanceIO می‌باشد که کد آن از طریق بخش لینک‌های مفید در دسترس است. کار کردن با این ابزار بسیار ساده است. در ادامه توضیحات لازم به منظور کار کردن شما با این ابزار خواهد آمد.

### مراحل انجام پروژه:

- ساخت ما شین مجازی ubuntu 12 به عنوان محیط کار پروژه. توجه دا شته با شید که در صورت ا ستفاده از نسخه‌های بالاتر، ممکن است با مشکلاتی روبرو شوید.
- این ماشین مجازی باید شامل دو حافظه باشد. یک حافظه کند و یک حافظه سریع که به عنوان کش از آن استفاده خواهد شد. نسبت سایز این دو حافظه ۱۰ به ۱ خواهد بود (سایز حافظه کند ۱۰ برابر حافظه سریع). بهترین راه برای ایجاد این دو حافظه، استفاده از دیسک سخت و SSD، به ترتیب است. در صورتی که سیستم شما دارای SSD نمی‌باشد، می‌توانید از یک USB 2 به عنوان حافظه کند و از هارد دیسک خود به عنوان حافظه سریع استفاده کنید.

توجه خیلی مهم: تنظیمات auto caching را برای هر دو حافظه کند و سریع تعریف شده در ما شین مجازی خود، disable نمایید. در غیر این صورت تاثیرات تعریف کش و استفاده از آن، غیر قابل بررسی خواهد شد.

- ابزار EnhanceIO را کامپایل و نصب نمایید. برای این کار می‌توانید از این لینک کمک بگیرید.
- ابزار fio را نصب نمایید.

این ابزار یک برنامه کاربردی است که از آن به منظور ارزیابی عملکرد حافظه‌ها در هنگام خواندن/ نوشتن از/روی آن‌ها مورد استفاده قرار می‌گیرد. نحوه نصب این ابزار با استفاده از دستور ساده زیر انجام می‌شود.

```
sudo apt-get install fio
```

نحوه استفاده از دستور fio برای ارزیابی سرعت خواندن و نوشتن در حافظه:

این ابزار یک فایل ورودی با پسوند ini می‌گیرد که در آن مشخصات تسک خواسته شده از آن حافظه، آمده است. دستور اجرای fio به شکل زیر است:

fio job.ini

محتویات یک فایل ساده job.ini در زیر قابل مشاهده است:

```
[test]
direct=1
size=50M
blocksize=4096
ioengine=libaio
rw=rw
numjobs=1
iodepth=8
filename=/dev/sdb
```

[test] نام section ا ست. لازم به ذکر ا ست که یک فایل ini شامل section های مختلف می‌تواند با شد که هر کدام شامل تعدادی متغیر با مقادیر اختصاص داده شده به آن‌ها است. ما در این پروژه تنها از یک section استفاده می‌کنیم.

Direct یک متغیر bool ا ست که یک بودن آن، کار کردن م ستقیم با device مورد نظر ما که نامش در filename، در پایین فایل، آمده اســت را مشــخص می‌کند و از دخالت ســیســتم عامل در فرآیند خواندن و نوشــتن جلوگیری می‌کند. این کار باعث می‌شود عملکرد خود device به صورت خالص مورد ارزیابی قرار گیرد. ما در این پروژه همواره این را برابر ۱ در نظر می‌گیریم.

size ، اندازه کل محتوایی که قرار است روی device مان نوشته و خوانده شود، را مشخص می‌کند.

Blocksize اندازه چانک‌هایی است که در یک بار عمل خواندن و یا نوشتن روی device مورد استفاده قرار می‌گیرد.

Ioengine را همان libaio قرار دهید.

Rw را هم همان rw بگذارید. این به معنای توالی و پشت سر هم بودن عملیات خواندن و نوشتن است.

numjobs را ۱ در نظر بگیرید چون تنها یک device را تست می‌کنیم.

Iodepth را نیز بدون تغییر ۸ در نظر بگیرید.

Filename نام دیوایسی است که می‌خواهید تست را روی آن انجام دهید. برای اینکه لیست storage های سیستم خود را به دست آورید از دستور lsblk استفاده کنید و حافظه مد نظر خود را با توجه به ظرفیت آن از بین آن‌ها تشخیص دهید.

- حال با اســتفاده از EnhanceIO یک کش با replacement policy = LRU و cache mode های WT و WB بســازید و با اســتفاده از دســتور fio عملکرد دیوایس را در هر کدام از دو حالت به دســت آورید و با حالت بدون کش مقایســه کنید. نحوه ساخت کش با استفاده از enhanceIO به دستور زیر است:

```
eio_cli create -d <SLOW_DEVICE_ADD> -s <FAST_DEVICE_ADD> -p <POLICY_NAME> -m <CACHE_MODE> -c <CACHE_NAME>
```

برای DEVICE_ADD ها باز هم باید از lsblk نام حافظه‌های ســـریع و کند خود را به دســـت آورید که چیزی شـــبیه نام /dev/sdb می‌تواند باشد.

دستور حذف کردن یک کش بسیار ساده و به صورت زیر است:

Eio_cli delete -c <CACHE_NAME>

همچنین در هر لحظه می‌توانید کش‌های ساخته شده توسط EnhanceIO را با دستور eio_cli info به دست آورید.

به منظور دیباگ کردن کدتان می‌توانید محتویات فایل

/proc/enhanceio/<cache_name>/stats

را مشاهده کنید. این فایل شامل اطلاعات مفیدی درباره کش شما است مانند تعداد hit های خواندن و نوشتن، تعداد missها، تعداد block های dirty و ...

- در ادامه شـــما باید کدهای ابزار enhanceIO را گســـترش دهید و replacement policy جدیدی به نام LFU به آن اضـــافه کنید. همانطور که میدانید، در سیاست LFU، از بین بلاک‌های با index یکسان، بلاکی به عنوان بلاک قربانی انتخاب می‌شود که به کمترین تعداد استفاده شده است.

  برای این کار شما باید کدهای پوشه Driver/enhance_io را مطالعه کنید و با نحوه پیاده سازی یک سیاست جایگذاری آشنا شوید.

  پس از پیاده سازی این سیاست و کامپایل مجدد enhanceIO شما باید مجددا، یک کش با سیاست جدید پیاده سازی شده ایجاد کنید و عملکرد دیوایس کند خود را مجدد با ابزار fio ارزیابی نمایید.

### خروجی‌های قابل تحویل:

- گزارش مکتوب از تنظیمات ماشین مجازی، دستورات اجرا شده و نتایج تست‌ها
- کدهای شما
- نیازی به تحویل فایل ماشین مجازی نیست ولی در زمان تحویل حضوری پروژه باید همان ماشین مجازی‌ای که کدها روی آن اجرا شده‌اند را داشته باشید و مراحل خواسته شده را به درخواست TA انجام دهید.

### لینک‌های مفید:

ابزار EnhanceIO از لینک زیر قابل دسترس است:

https://github.com/lanconnected/EnhanceIO

کاربرد عملی این ابزار در پژوهش‌های مربوط به حافظه‌ی سیستم‌ها:

https://ieeexplore.ieee.org/document/9380565

ابزار Flashcache که EnhanceIO برمبنای آن توسعه یافته و بررسی آن دید بهتری نسبت به این ابزارها به دست می‌دهد:

https://github.com/facebookarchive/flashcache

## پروژه‌ی پنجم

### عنوان:

پیاده‌سازی IO Cache برمبنای سیاست LIFO

### مقدمه:

امروزه از حافظه‌های بر پایه فلش، مانند SSD، به عنوان cache برای دیسک‌های سخت که کندتر و با سایز بزرگ‌تر هستند، استفاده می شود. به این کار IO caching گفته می شود. بزرگترین مزیت IO caching آن ا ست که بدون ایجاد تغییر در زیر ساخت‌های سخت افزاری و تنها با نصب و سپس تعریف یک حافظه سریع‌تر به عنوان cache می‌توان سرعت و کارایی سیستم را افزایش داد.

توجه مهم: مطالعه و مرور مباحث cache از درس معماری کامپیوتر به شما در اجرای این پروژه کمک شایانی خواهد کرد، مطالبی مانند direct mapped cache, associative caches, …

### شرح پروژه:

در سیستم عامل لینوکس، ابزارهای متن باز زیادی برای ایجاد (تعریف) کش وجود دارد. یکی از شناخته شده‌ترین این ابزارها، EnhanceIO می‌باشد که کد آن از طریق بخش لینک‌های مفید در دسترس است. کار کردن با این ابزار بسیار ساده است. در ادامه توضیحات لازم به منظور کار کردن شما با این ابزار خواهد آمد.

### مراحل انجام پروژه:

- ساخت ما شین مجازی ubuntu 12 به عنوان محیط کار پروژه. توجه دا شته با شید که در صورت ا ستفاده از نسخه‌های بالاتر، ممکن است با مشکلاتی روبرو شوید.
- این ماشین مجازی باید شامل دو حافظه باشد. یک حافظه کند و یک حافظه سریع که به عنوان کش از آن استفاده خواهد شد. نسبت سایز این دو حافظه ۱۰ به ۱ خواهد بود (سایز حافظه کند ۱۰ برابر حافظه سریع). بهترین راه برای ایجاد این دو حافظه، استفاده از دیسک سخت و SSD، به ترتیب است. در صورتی که سیستم شما دارای SSD نمی‌باشد، می‌توانید از یک USB 2 به عنوان حافظه کند و از هارد دیسک خود به عنوان حافظه سریع استفاده کنید.

توجه خیلی مهم: تنظیمات auto caching را برای هر دو حافظه کند و سریع تعریف شده در ما شین مجازی خود، disable نمایید. در غیر این صورت تاثیرات تعریف کش و استفاده از آن، غیر قابل بررسی خواهد شد.

- ابزار EnhanceIO را کامپایل و نصب نمایید. برای این کار می‌توانید از این لینک کمک بگیرید.
- ابزار fio را نصب نمایید.

این ابزار یک برنامه کاربردی است که از آن به منظور ارزیابی عملکرد حافظه‌ها در هنگام خواندن/ نوشتن از/روی آن‌ها مورد استفاده قرار می‌گیرد. نحوه نصب این ابزار با استفاده از دستور ساده زیر انجام می‌شود.

```
sudo apt-get install fio
```

نحوه استفاده از دستور fio برای ارزیابی سرعت خواندن و نوشتن در حافظه:

این ابزار یک فایل ورودی با پسوند ini می‌گیرد که در آن مشخصات تسک خواسته شده از آن حافظه، آمده است. دستور اجرای fio به شکل زیر است:

fio job.ini

محتویات یک فایل ساده job.ini در زیر قابل مشاهده است:

```
[test]
direct=1
size=50M
blocksize=4096
ioengine=libaio
rw=rw
numjobs=1
iodepth=8
filename=/dev/sdb
```

[test] نام section ا ست. لازم به ذکر ا ست که یک فایل ini شامل section های مختلف می‌تواند با شد که هر کدام شامل تعدادی متغیر با مقادیر اختصاص داده شده به آن‌ها است. ما در این پروژه تنها از یک section استفاده می‌کنیم.

Direct یک متغیر bool ا ست که یک بودن آن، کار کردن م ستقیم با device مورد نظر ما که نامش در filename، در پایین فایل، آمده اســت را مشــخص می‌کند و از دخالت ســیســتم عامل در فرآیند خواندن و نوشــتن جلوگیری می‌کند. این کار باعث می‌شود عملکرد خود device به صورت خالص مورد ارزیابی قرار گیرد. ما در این پروژه همواره این را برابر ۱ در نظر می‌گیریم.

size ، اندازه کل محتوایی که قرار است روی device مان نوشته و خوانده شود، را مشخص می‌کند.

Blocksize اندازه چانک‌هایی است که در یک بار عمل خواندن و یا نوشتن روی device مورد استفاده قرار می‌گیرد.

Ioengine را همان libaio قرار دهید.

Rw را هم همان rw بگذارید. این به معنای توالی و پشت سر هم بودن عملیات خواندن و نوشتن است.

numjobs را ۱ در نظر بگیرید چون تنها یک device را تست می‌کنیم.

Iodepth را نیز بدون تغییر ۸ در نظر بگیرید.

Filename نام دیوایسی است که می‌خواهید تست را روی آن انجام دهید. برای اینکه لیست storage های سیستم خود را به دست آورید از دستور lsblk استفاده کنید و حافظه مد نظر خود را با توجه به ظرفیت آن از بین آن‌ها تشخیص دهید.

- حال با اســتفاده از EnhanceIO یک کش با replacement policy = LRU و cache mode های WT و WB بســازید و با اســتفاده از دســتور fio عملکرد دیوایس را در هر کدام از دو حالت به دســت آورید و با حالت بدون کش مقایســه کنید. نحوه ساخت کش با استفاده از enhanceIO به دستور زیر است:

```
eio_cli create -d <SLOW_DEVICE_ADD> -s <FAST_DEVICE_ADD> -p <POLICY_NAME> -m <CACHE_MODE> -c <CACHE_NAME>
```

برای DEVICE_ADD ها باز هم باید از lsblk نام حافظه‌های ســـریع و کند خود را به دســـت آورید که چیزی شـــبیه نام /dev/sdb می‌تواند باشد.

دستور حذف کردن یک کش بسیار ساده و به صورت زیر است:

Eio_cli delete -c <CACHE_NAME>

همچنین در هر لحظه می‌توانید کش‌های ساخته شده توسط EnhanceIO را با دستور eio_cli info به دست آورید.

به منظور دیباگ کردن کدتان می‌توانید محتویات فایل

/proc/enhanceio/<cache_name>/stats

را مشاهده کنید. این فایل شامل اطلاعات مفیدی درباره کش شما است مانند تعداد hit های خواندن و نوشتن، تعداد missها، تعداد block های dirty و ...

- در ادامه شـما باید کدهای ابزار enhanceIO را گسـترش دهید و replacement policy جدیدی به نام LIFO به آن اضـافه کنید. همانطور که میدانید، در سیاست LIFO، از بین کل بلاک‌های با index یکسان، بلاکی به عنوان قربانی انتخاب می شود که آخر از همه به کش اضافه شده است.

برای این کار شما باید کدهای پوشه Driver/enhance_io را مطالعه کنید و با نحوه پیاده سازی یک سیاست جایگذاری آشنا شوید.

پس از پیاده سازی این سیاست و کامپایل مجدد enhanceIO شما باید مجددا، یک کش با سیاست جدید پیاده سازی شده ایجاد کنید و عملکرد دیوایس کند خود را مجدد با ابزار fio ارزیابی نمایید.

## خروجی‌های قابل تحویل:

- گزارش مکتوب از تنظیمات ماشین مجازی، دستورات اجرا شده و نتایج تست‌ها
- کدهای شما
- نیازی به تحویل فایل ماشین مجازی نیست ولی در زمان تحویل حضوری پروژه باید همان ماشین مجازی‌ای که کدها روی آن اجرا شده‌اند را داشته باشید و مراحل خواسته شده را به درخواست دستیار آموزشی انجام دهید.

## لینک‌های مفید:

ابزار EnhanceIO از لینک زیر قابل دسترس است:

https://github.com/lanconnected/EnhanceIO

کاربرد عملی این ابزار در پژوهش‌های مربوط به حافظه‌ی سیستم‌ها:

https://ieeexplore.ieee.org/document/9380565

ابزار Flashcache که EnhanceIO برمبنای آن توسعه یافته و بررسی آن دید بهتری نسبت به این ابزارها به دست می‌دهد:

https://github.com/facebookarchive/flashcache

## (پروژه‌های مربوط به گروه آقای فراهانی)

### پروژه‌ی ششم:

### عنوان پروژه:

سامانه برخط ویژگی‌شناسی بارهای کاری یادگیری ماشین/یادگیری عمیق از دیدگاه I/O

### مقدمه:

امروزه سامانه هایی همچون Tensorboard جهت بررسی و تحلیل نتایج بدست آمده از اجرای برنامه های یادگیری ماشین/یادگیری عمیق وجود دارند. این سامانه ها نتایج بسیار مفید و در مواردی، بسیار دقیق، همچون میزان دقت مدل یادگیری، میزان خطا در تشخیص تصاویر، میزان منابع پردازشی استفاده شده در طول اجرا، مقایسه اجرای برنامه از نظر پارامترهای اجرا، به کاربر ارائه میدهند. اما مشکل عمده این سامانه ها، عدم ارائه نتایج تاثیرات اجرای برنامه در لایه ذخیره سازی و راهکارهایی جهت بهبود عملکرد این لایه از سامانه پردازشی میباشد. همچنین، تعداد زیادی از این وب اپلیکیشن ها تنها از یک یا چند فریم ورک مشخص پشتیبانی میکنند.

با توجه به موارد فوق، نیاز به طراحی سامانه ویژگی شناسی بارهای کاری برخط جهت تحلیل اجرای برنامه های حوزه یادگیری ماشین/یادگیری عمیق بصورت جامع و در تمام ابعاد سامانه اعم از پردازشی و ذخیره سازی احساس میشود.

### شرح پروژه:

هدف از انجام این پروژه، پیاده سازی سامانه ای برخط، جهت تحلیل و نمایش نتایج بدست آمده از اجرای برنامه های کاربردی پردازش سریع در حوزه یادگیری ما شین/یادگیری عمیق و بطور خاص، در لایه ذخیره سازی سامانه میبا شد. بدین صورت که ابتدا کاربر کد یا برنامه اجرایی خود را به همراه منابع مورد نیاز در سامانه بارگزاری میکند. سپس سامانه بطور خودکار محیط اجرای برنامه را آماده سازی کرده، اجرای برنامه را شروع میکند و در همین حین، بطور همزمان ابزار تحلیل لایه ذخیره سازی شامل blktrace و iostat را نیز اجرا میکند. در نهایت، نتایج بد ست آمده از اجرای ابزار I/O Trace، تحلیل و ویژگی شنا سی شده و ب صورت برخط در قالب نمودار، در وب اپلیکیشن طراحی شده با رابط کاربری مناسب به کاربر نمایش داده میشود.

### مراحل انجام پروژه:

- در ابتدا باید یک وب اپلیکیشن ساده پیاده سازی نمایید که در فاز اول، شامل محیطی است که کاربر میتواند فریم ورک یا کتابخانه های مدنظر خود را (شامل TensorFlow، PyTorch، OpenCV و scikit-learn) انتخاب کرده و سپس، کد و دستورات خود را در آن بارگذاری کند.
- در مرحله بعد، کاربر نوع ، حجم و نام دیتاست را مشخص کرده و میتواند اجرای فاز Training، Inference و یا هر دو را انتخاب کند.
- سپس، وب اپ طراحی شده با توجه به موارد مشخص شده از سمت کاربر، محیط اجرای برنامه را آماده سازی کرده و اجرا بر روی ماشین مجازی یا سرور واقعی شروع میشود.
- در همین حین، بطور همزمان و با توجه به راهنمایی که در اختیارتان قرار میگیرد، بررسی I/O Trace را با ابزار گفته شده انجام می‌گیرد.

- در نهایت، باید کدی را جهت ا ستخراج نتایج اولیه بد ست آمده از ابزارها پیاده سازی کرده و عمل ویژگی شنا سی را در سطح نموداری و متنی (با توجه به موارد خواســـته شـــده در راهنمای۲ که در اختیارتان قرار میگیرد) انجام داده و آنها را با طراحی مناسب در صفحه ای در وب اپلیکیشن به کاربر نمایش دهید.

## نکات مهم:

- منظور از برنامه کاربردی، یک پروژه متن باز یا کد آماده‌ای ا ست که می‌توانید در وب سایت هایی همچون Github و Kaggle بیابید.
- برای دمو و صحت سنجی پروژه، می بای ست یک برنامه از فریم ورک TensorFlow، یک برنامه از فریم ورک PyTorch، یک برنامه با کتابخانه OpenCV و یک برنامه با کتابخانه scikit-learn انتخاب نموده و مراحل انجام کار را (به عنوان کاربر سیستم) شبیه سازی کنید. برنامه های انتخاب شده باید به تایید تیم دستیاری برسد.
- برنامه‌های انتخابی باید شامل مباحثی همچون تشخیص تصاویر انسان، تشخیص اجسام در عکس، پردازش ویدئو و پردازش متن یا زبان طبیعی باشد.
- انتخاب دیتاست توسط کاربر، از لیست مشخص و از پیش تعیین شده ای است که بصورت API به شما داده میشود.
- پیاده سازی وب اپلیکیشن میتواند با هر زبان یا فریم ورکی باشد و انتخاب آن به عهده افراد تیم است.
- توصیه و یا اجباری در چگونگی پیاده سازی رابط کاربری (UI) وجود ندارد، اما طبیعتاً بخشی از نمره دریافتی به کیفیت تجربه کاربری (UX) اختصاص دارد.
- محیط اجرا می‌تواند یک ماشین مجازی و یا سیستم‌عامل لینوکسی باشد که به عنوان سیستم اصلی از آن استفاده می‌کنید.
- تمام مراحل کار باید بصورت خودکار انجام شود. (مثلا اتصال وب اپلیکیشن به ماشین اجرا با API صورت گیرد).
- همانطور که گفته شد، سامانه طراحی شده بایستی بصورت برخط کار کند. بدین معنا که در طول زمان اجرای برنامه، تمامی موارد خواسته شده در فایل راهنمای۲، باید بطور لحظه ای در قالب یک داشبورد به کاربر نمایش داده شده و بروزرسانی گردد (به جهت آشنایی بیشتر یک نمونه داشبورد در راهنمای۲ موجود است).
- راهنمای۱ جهت نصب و استفاده از ابزار blktrace و iostat در اختیار شما قرار خواهد گرفت.
- جهت درک صحیح از نمودارها و نتایج خواسته شده حاصل از ویژگی‌شناسی، راهنمای۲ در اختیارتان قرار خواهد گرفت.

## لینک‌های مفید:

- https://kaggle.com/code
- https://paperswithcode.com/datasets
- https://strugglers.net/~andy/blog/category/linux/blktrace
- https://www.tensorflow.org/tensorboard
- https://neptune.ai
- https://wandb.ai/site

## پروژه‌ی هفتم

### عنوان پروژه:

آشنایی با ویژگی‌شناسی بارهای کاری I/O در برنامه های کاربردی پردازش سریع

### مقدمه:

در سال‌های اخیر، رشد برنامه‌های کاربردی داده-محور موجب شده‌است که لایه‌ی ذخیره سازی به یکی از گلوگاه‌های کارایی سامانه‌های پردازش سریع تبدیل شود. یکی از روش‌های بهبود کارایی، بهره‌گیری از معماری شخصی سازی شده در این سامانه‌ها است که مستلزم شناخت صحیح و دیدگاه جامع نسبت به ویژگی‌ها و رفتار برنامه‌های کاربردی می‌باشد. ویژگی‌شناسی بارهای کاری از دیدگاه ورودی/خروجی، این امکان را می‌دهد تا بوسیله ارزیابی الگوی رفتاری برنامه‌های کاربردی در لایه بلوک ورودی/خروجی سیستم عامل، ابعاد گوناگون مسیر داده در لایه سیستم عامل و لایه ذخیره‌سازی بررسی شده و به بهبود عملکرد بخش ذخیره‌سازی سامانه منجر شود.

### شرح پروژه:

در این پروژه قصد داریم تا با استفاده از ابزار غنی سیستم عامل لینوکس در حوزه ویژگی شناسی بارهای کاری I/O، تعدادی از برنامه‌های داده-محور در حوزه یادگیری ماشین/یادگیری عمیق که در سامانه‌های پردازش سریع کاربرد زیادی دارند را اجرا کرده و همزمان عملیات بررسی و ویژگی‌شناسی را در سطح لایه بلاک ورودی/خروجی سیستم عامل انجام دهیم.

ابزار مورد استفاده در این پروژه شامل blktrace و iostat می‌باشد. این دو ابزار، بطور همزمان و در حین اجرای برنامه‌های کاربردی اجرا شده و مشخصاتی بسیار مفید و دقیق از لایه بلاک ورودی/خروجی سیستم عامل به کاربر بر می گردانند. این مشخصات شامل نوع عملیات ورودی/خروجی، زمان انجام عملیات ورودی/خروجی، آدرس‌های مورد دسترسی توسط برنامه ارسال کننده درخواست، میزان خواندن/نوشتن در ثانیه، بهره وری دیسک و ... می‌باشد.

شما باید تعدادی برنامه کاربردی در حوزه یادگیری ماشین/یادگیری عمیق که توسط فریم ورک‌هایی نظیر TensorFlow و PyTorch و کتابخانه‌هایی همچون OpenCV و scikit-learn را اجرا کرده و در حین اجرا، ابزار گفته شده را به جهت بررسی و ویژگی شناسی لایه ذخیره سازی سیستم اجرا نمایید. سپس، عملیات ویژگی شناسی را بر روی خروجی دو ابزار blktrace و iostat انجام داده و در نهایت، نتایج متنی/تصویری شامل نمودارها و فایل‌های متنی خواسته شده را نمایش دهید.

### مراحل انجام پروژه:

- در ابتدا باید ۲ برنامه کاربردی از فریم ورک TensorFlow، ۲ برنامه از فریم ورک PyTorch، ۲ برنامه با کتابخانه OpenCV و یک برنامه با کتابخانه scikit-learn انتخاب نموده و در مورد کارکرد و کاربری آنها بطور مختصر گزارشی تحویل نمایید.
- پس از تایید برنامه‌ها توسط تیم دستیاری، باید محیط تست و پیاده سازی را آماده نمایید. این محیط باید شامل موارد زیر باشد (موارد گفته شده **حداقل کانفیگ** می‌باشد):
  - سیستم عامل لینوکس (ترجیحا توزیع Ubuntu و نسخه کرنل ۴ و بالاتر)
  - ۴ هسته پردازشی (۸ ریسمان پردازشی)
  - ۱۰ گیگابایت حافظه اصلی
  - ۱۰۰ گیگابایت فضای ذخیره سازی (ترجیحا از نوع SSD)

- سپس به آماده‌سازی محیط اجرای برنامه‌ها بپردازید. در این بخش، اگر برنامه مدنظر به زبان پایتون می‌باشد، باید از محیط مجازی Anaconda استفاده نموده و پکیج‌ها و Dependency های مورد نیاز برنامه را نصب کنید.
- حال برنامه را اجرا کرده و همزمان، با توجه به راهنمایی که در اختیارتان قرار میگیرد، برر سی I/O Trace را با ابزار گفته شده انجام می‌دهید.
- در نهایت، باید کدی را جهت ا ستخراج نتایج اولیه بد ست آمده از ابزارها پیاده سازی کرده و عمل ویژگی شنا سی را در سطح نموداری و متنی (با توجه به موارد خواسته شده در راهنمایی که در اختیارتان قرار میگیرد) انجام دهید.
- این مراحل به ازای تمام برنامه‌های انتخابی انجام میگردد.

## نکات مهم:

- منظور از برنامه کاربردی، یک پروژه متن باز یا کد آماده‌ای ا ست که می‌توانید در وب سایت هایی همچون Github و Kaggle بیابید.
- دیتا ست‌های انتخابی جهت اجرای مرحله Training، باید حداقل ۲۰ گیگابایت (بدون احت ساب Label و annotation) با شد. همچنین، می‌توانید از لینک زیر دیتاست‌های پرکاربرد حوزه یادگیری ماشین را مشاهده نموده و طبق راهنما دانلود نمایید:

  *https://hpc.sharif.edu/data-services*

- برنامه‌های انتخابی باید شامل مباحثی همچون ت شخیص ت صاویر ان سان، ت شخیص اج سام در عکس، پردازش ویدئو و پردازش متن یا زبان طبیعی باشد.
- نمودارهای ترسیمی حاصل از نتایج بدست آمده باید با کتابخانه های رسم نمودار در زبانهای برنامه نویسی (مثل کتابخانه matplotlib در زبان پایتون) باشد. رسم نمودار با Excel مورد تایید نمیباشد.
- محیط اجرا می‌تواند یک ماشین مجازی و یا سیستم‌عامل لینوکسی باشد که به عنوان سیستم اصلی از آن استفاده می‌کنید.
- راهنمای۱ نصب و استفاده از ابزار blktrace و iostat در اختیار شما قرار خواهد گرفت.
- جهت درک صحیح از نمودارها و نتایج خواسته شده حاصل از ویژگی‌شناسی، راهنمای۲ در اختیارتان قرار خواهد گرفت.

## لینک‌های مفید:

- https://kaggle.com/code
- https://paperswithcode.com/datasets
- https://strugglers.net/~andy/blog/category/linux/blktrace

## پروژه‌ی هشتم:

### عنوان:

پیاده‌سازی وب سرور چندریسه‌ای

### مقدمه:

شما در حال توسعه یک وب سرور خواهید بود. این وب سرور تنها با یک ریسه در ابتدا باید کار کند. سپس وب سرور را چند ریسه‌ای کنید تا کارآمدتر باشد.

### شرح و مراحل پروژه:

ابتدا یک وب‌سرور تک‌ریسه‌ای توسعه دهید. دقت کنید این وب‌سرور باید تمام درخواست‌ها را در داخل یک صف ذخیره کند و به ترتیب به آن‌ها رسیدگی کند. می‌توانید فرض کنید وظیفه وب‌سرور محاسبه یک عبارت ریاضی است و مثل ماشین‌حساب جمع دو عدد ورودی را پاسخ می‌دهد.(usecase وب سرور در این سوال مدنظر نیست)

در این وب‌سرور سیاست‌های مختلفی برای هندل کردن نوبت درخواست بعدی می‌توانید در نظر بگیرید مثل FCFS. مهم نیست این الگوریتم چه باشد ولی حتما توجه داشته باشید که پالیسی انتخاب از لیست را به درستی پیاده سازی کنید که در ادامه برای انتخاب از ترد پول به آن نیاز خواهید داشت.

در گام بعدی یک ترد پول با تعداد ترد n بسازید و ریکوئست‌ها را بر اساس الگوریتم FCFS(یا الگوریتم دلخواه دیگر در بخش پیشین پیاده کردید) به این ترد ها تخصیص دهید. راه های متفاوتی برای این کار وجود دارد ولی پیشنهاد میشود یک ترد مستر داشته باشید که مسئولیت ساختن ترد پول اولیه را به عهده داشته باشد و در ادامه تخصیص ریکوئست ها را مدیریت کند. همینطور اگر جایی برای جلوگیری از race condition لازم بود کاری انجام دهید به هیچ عنوان busy waiting نکنید و ترجیحا با conditional variables آن ها را مدیریت کنید.

در پایان چند آماره زیر را پیاده سازی کنید تا بتوانیم از صحت عملکرد سرور اطمینان حاصل کنیم.

ورود: زمان ورود برای اولین بار توسط سیستم مشاهده شد

ارسال: مدت زمان بین زمان رسیدن و زمانی که درخواست توسط یک ریسه به عهده گرفته شود.

اتمام: مدت زمان بین زمان رسیدن و زمانی که thread شروع به نوشتن پاسخ در سوکت می کند.

### نکات انجام پروژه:

- این پروژه باید به C زده شود.
- ایجاد نشت حافظه در C بسیار آسان است. بنابراین، استفاده از valgrind برای بررسی آنها به شدت توصیه می شود.
- برای اطمینان از صحت پیاده سازی حتما به آماره ها توجه داشته باشید.
- ایجاد ریسه های بیش از حد باعث هدر رفتن منابع می شود و برای ایجاد ریسه های استفاده نشده زمان صرف می شود.

**لینک های مفید:**

- https://www.geeksforgeeks.org/handling-multiple-clients-on-server-with-multithreading-using-socket-programming-in-c-cpp/

## پروژه‌ی نهم

## عنوان پروژه:

پیاده سازی یک فایل سیستم ساده بر روی دیسک مجازی

## مقدمه:

در این پروژه، ما یک کتابخانه در سطح کاربر به نام libFS خواهیم ساخت که بخشی از یک فایل سیستم را پیاده سازی می کند. فایل سیستم شما در داخل کتابخانه ای ساخته می شود که برنامه ها می توانند برای دسترسی به فایل‌ها و پوشه‌ها با آن پیوند برقرار کنند. کتابخانه شما خود با لایه link میشود که یک "دیسک" را پیاده سازی می کند. ما این کتابخانه، یعنی LibDisk را ارائه می دهیم که باید از آن استفاده کنید.

## شرح پروژه و مراحل انجام:

### مشخصات LibFS

ما با توصیف از رابط واسط کاربری LibFs به فایل سیستم شروع می کنیم. سه بخش برای API وجود دارد: دو فراخوانی عمومی فایل سیستم، مجموعه‌ای ازفراخوانی‌ها که با دسترسی به فایل سروکار دارند، و مجموعه‌ای از تماس ها که با پوشه ها و برنامه‌ها (به عنوان مثال، برنامه‌های آزمایشی خودتان، و مطمئناً برنامه‌های آزمایشی ما) به منظور آزمایش فایل سیستم شما، با LibFS مرتبط می‌شوند.
نحوه عملکرد کتابخانه شما و همچنین نحوه رسیدگی آن به خطاها آزمایش خواهد شد. هنگامی که خطایی رخ می دهد (هر خطای احتمالی در زیر در تعریف API مشخص شده است)، کتابخانه شما باید متغیر سراسری osErrno را برابر خطای توضیح داده شده در تعریف API زیر قرار دهد و کد خطای مناسب را برگرداند. به این ترتیب، برنامه‌هایی که به کتابخانه شما link میشوند، راهی برای دیدن اتفاقی که هنگام بروز خطا رخ داده است، دارند.

### API عمومی فایل سیستم

```
int FS_Boot (char *path)
```

FS_Boot() باید دقیقاً یک بار قبل از فراخوانی سایر توابع LibFS فراخوانی شود. path که یا به یک فایل واقعی اشاره می کند که در آن "تصویر دیسک" شما ذخیره شده یا به فایلی اشاره می‌کند که هنوز وجود ندارد و باید برای نگهداری یک تصویر دیسک جدید ساخته شود. پس از موفقیت، 0 را برگردانید. در صورت شکست، 1- را برگردانید و osErrno را برابر E_GENERAL قرار دهید.

```
int FS_Boot (char *path)
```

FS_Sync() اطمینان حاصل می کند که محتویات فایل سیستم به طور پایدار روی دیسک ذخیره شده. جزئیات بیشتر در مورد نحوه انجام این کار با استفاده از libDisk در زیر موجود است. پس از موفقیت، 0 را برگردانید. در صورت شکست، 1- را برگردانید و osErrno را برابر E_GENERAL قرار دهید.

### API دسترسی به فایل

توجه داشته باشید که تعدادی از این عملیات‌ها با مسیرنام‌ها (pathnames) سروکار دارند. از این رو، ما باید چند فرض در مورد مسیرنام‌ها داشته باشیم. همه مسیرها مطلق هستند. یعنی هر زمان که به یک فایل اشاره شد، مسیر کاملی که از ریشه شروع می شود انتظار منظور است. همچنین، فرض کنید که حداکثر طول نام یک نام فایل 16 بایت (15 کاراکتر به string null terminator) است. در نهایت، فرض کنید که حداکثر طول یک مسیر 256 کاراکتر است.

```
int File_Create (char *file)
```

File_Create() یک فایل جدید با نامی که فایل به آن ا شاره می کند ایجاد می کند. اگر فایل از قبل وجود دا شت، باید 1- را برگردانید و osErrno را روی E_CREATE تنظیم کنید. توجه: پس از این فراخوانی، فایل نباید "باز" باشـــد. در عوض، File_Create() باید به سادگی یک فایل جدید روی دیسک با اندازه 0 ایجاد کند. پس از موفقیت، 0 را
برگردانید. در صورت شکست، 1- را برگردانید و E_CREATE را برگردانید.

```
int File_Open(char *file)
```

File_Open() یک فایل را باز می کند (که نام آن توسط آرگومان file مشخص شده) و یک توصیف‌گر فایل (عددی بزرگتر یا مساوی 0) را برمی گرداند که می تواند برای خواندن یا نوشتن داده ها در آن فایل استفاده شود. اگر فایل وجود نداشت، 1- را برگردانید و osErrno را برابر E_NO_SUCH_FILE قرار دهید. اگر از قبل حداکثر تعداد فایل باز شـــده اســـت، 1- را برگردانید و osErrno را روی E_TOO_MANY_OPEN_FILES تنظیم کنید.

```
int File_Read (int fd، void *buffer، int size)
```

File_Read() باید به اندازه size بایت از فایلی که توسط توصیف‌گر فایل fd مشخص شده است. داده ها باید در بافری که buffer به آن اشاره می کند خوانده شود. همه خواندن‌ها باید از محل فعلی نشان‌گر فایل شروع شوند و نشان‌گر فایل باید پس از خواندن به مکان جدید جابه‌جا شـــود. اگر فایل باز نبود، 1- را برگردانید و osErrno را برابر E_BAD_FD قرار دهید. اگر فایل باز بود، تعداد بایت‌های خوانده شده باید برگردانده شود که می‌تواند کمتر یا مساوی اندازه با شد. (تعداد ممکن ا ست کمتر از بایت‌های درخوا ستی با شد زیرا می‌توان به انتهای فایل رسید). اگر نشانگر فایل از قبل در انتهای فایل باشد، باید صفر برگردانده شود، حتی در فراخوانی‌های دوباره.

```
int File_Write (int fd، void *buffer، int size)
```

File_Write() باید به اندازه size بایت را از بافر بر روی فایلی که توسط توصیف‌گر fd مشخص شده بنویسد. همه نوشتن ها باید از محل فعلی نشان‌گر فایل شروع شود و نشانگر فایل باید پس از نوشتن به مکان فعلی جابه‌جا شود. توجه داشته باشید که نوشتن تنها راه افزایش اندازه یک فایل ا ست. اگر فایل باز نیست، 1- را برگردانید و osErrno را برابر E_BAD_FD قرار دهید. پس از موفقیت در نو شتن، تمام داده‌ها باید روی دی سک نو شته شوند و مقدار size باید برگردانده شود. اگر نو شتن نمی تواند کامل شود (به دلیل کمبود ف ضا)، 1- را برگردانید و osErrno را برابر E_NO_SPACE قرار دهید. در نهایت، اگر فایل از حداکثر اندازه فایل بیشـــتر شـــد، باید 1- را برگردانید و osErrno را برابر E_FILE_TOO_BIG قرار دهید.

```
int File_Seek (int fd, int offset)
```

File_Seek() باید مکان فعلی نشانگر فایل را به روز کند. مکان جدید به صورت offset از ابتدای فایل درج شده است. اگر افست بزرگتر از اندازه فایل یا منفی است، 1- را برگردانید و osErrno را برابر E_SEEK_OUT_OF_BOUNDS قرار دهید. اگر فایل در حال حاضر باز نیست، 1- را برگردانید و osErrno را برابر E_BAD_FD قرار دهید. پس از موفقیت، مکان جدید نشانگر فایل را برگردانید.

```
int File_Close(int fd)
```

File_Close() فایلی را که توسط توصیف‌گر فایل fd مشخص شده است را می‌بندد. اگر فایل در حال حاضر باز نیست، 1- را برگردانید و osErrno را روی E_BAD_FD تنظیم کنید. پس از موفقیت، 0 را برگردانید.

```
int File_Unlink (char *file)
```

File_Unlink() باید فایل file را حذف کند، از جمله حذف آن فایل از پوشه‌ای که در آن قرار دارد و آزاد کردن بلوک های داده و inode هایی که فایل ا ستفاده می‌کرد. اگر فایل وجود ندا شت، 1- را برگردانید و osErrno را برابر E_NO_SUCH_FILE قرار دهید. اگر فایل در حال حا ضر باز ا ست، 1- را برگردانید و osErrno را برابر E_FILE_IN_USE قرار دهید (و فایل را حذف نکنید). پس از موفقیت، 0 را برگردانید.

**API کار با پوشه‌ها**

```
int Dir_Create(char * path)
```

Dir_Create() یک پو شه جدید در م سیر path می سازد. در فرایند ساخت یک پو شه جدید، چندین مرحله را انجام می دهد: ابتدا باید یک فایل جدید (از نوع پوشه) اختصاص دهید و سپس باید یک ورودی پوشه جدید را در والد پوشه فعلی اضافه کنید. در صورت هر گونه شکست، 1- را برگردانید و osErrno را برابر E_CREATE قرار دهید. پس از موفقیت، 0 را برگردانید. توجه داشته باشید که Dir_Create() بازگشتی است. یعنی اگر فقط "/" وجود داشته باشد، و شما بخواهید پوشه "/a/b/" را بسازید، هردوی "a/" و "a/b/" باید ساخته شود.

```
int Dir_Size (char * path)
```

Dir_Size() تعداد بایت های پو شه‌ای که تو سط آرگومان path مشخص شده ا ست را برمی گرداند. این تابع باید برای یافتن اندازه پو شه قبل از فراخوانی Dir_Read() (توضیح داده شده در زیر) برای یافتن محتویات پوشه استفاده شود.

```
int Dir_Read (char * path, void *buffer, int size)
```

از Dir_Read() می‌توان برای خواندن محتویات یک پو شه ا ستفاده کرد. این تابع باید مجموعه‌ای از پو شه‌ها را در buffer برگرداند. اندازه هر واحد خروجی 20 بایت است و شامل نام 16 بایتی پوشه‌ها و فایل‌های داخل پوشه‌ای است که با path مشخص شده به همراه یک عدد ۴ بایتی که شماره inode را نشان میدهد. اگر size به اندازه کافی بزرگ نیست که همه خروجی‌ها را شامل شود، 1- را برگردانید و osErrno را برابر E_BUFFER_TOO_SMALL قرار دهید. در غیر این صورت، داده‌ها را در buffer قرار دهید و تعداد ورودی‌های پوشه موجود در فهرست را برگردانید (به عنوان مثال، اگر دو ورودی در پوشه وجود دارد، 2 را برگردانید).

```
int Dir_Unlink (char * path)
```

Dir_Unlink() پوشه اشاره شده path را پاک می‌کند، بلوک‌های inode و داده آن را آزاد می‌کند و ورودی آن را از پوشه پدر پاک می‌کند. پس از موفقیت، 0 را برگردانید. توجه: Dir_Unlink () تنها زمانی با موفقیت پایان می‌یابد که هیچ فایلی در پو شه وجود ندا شته باشد. اگر هنوز فایل‌هایی در پوشه وجود دارد، 1- را برگردانید و osErrno را برابر E_DIR_NOT_EMPTY قرار دهید. اگر کسی سعی

کرد پوشه ریشه ("/") را حذف کند، به او اجازه ندهید این کار را انجام دهد! 1- را برگردانید و osErrno را روی E_ROOT_DIR تنظیم کنید.

### چند نکته مهم

هنگام خواندن یا نوشتن یک فایل، باید مفهوم نشان‌گر فایل را پیاده سازی کنید. پس از باز کردن یک فایل، نشانگر فایل بر روی آغاز فایل (بایت 0) جای میگیرد. اگر کاربر پس از آن N بایت را از فایل بخواند، نشان‌گر فایل باید به N به‌روز شود. خواندن M بایت دیگر، بایتهای N تا N+M را برمی گرداند. بنابراین، با فراخوانی مکرر read (یا نوشتن)، یک برنامه می تواند کل فایل را بخواند (یا بنویسد). البته، File_Seek() برای تغییر صریح مکان نشان‌گر فایل عمل میکند.

توجه داشته باشید که لازم نیست نگران پیاده‌سازی هیچ رویه‌ای مربوط به مسیرنام‌های نسبی نیستید. به عبارت دیگر، همه مسیرنام‌ها، مسیرهای مطلق خواهند بود. بنابراین، همه مسیرنام‌هایی که به هر یک از APIهای فایل و پوشه شما داده می شود، مسیرهای کاملی خواهند بود که از ریشه فایل سیستم آغاز می شوند، یعنی /a/b/c/foo.c. بنابراین، فایل سیستم شما نیازی به نگه‌داری هیچ مفهومی از "پوشه کاری فعلی" را ندارد.

## نکات اجرایی:

### انتزاع دیسک

یک فایل سیستم واقعی همه داده‌ها را بر روی دیسک ذخیره می‌کند، اما از آنجایی که ما تماما در سطح کاربر (usermode) کار میکنیم، داده‌ها را در یک دیسک «جعلی» که بدون هیچ هزینه‌ای در اختیار شما قرار می‌گیرد، ذخیره می‌کنیم. در LibDisk.c و LibDisk.h "دیسک" را پیدا خواهید کرد که برای این تکلیف باید از آن استفاده کنید.
"دیسکی" که ما ارائه می کنیم NUM_SECTORS بخش را به شما ارائه می دهد که هر کدام به اندازه SECTOR_SIZE است (اینها به عنوان contanst در LibDisk.h تعریف شده اند). بنابراین، شما باید از این مقادیر در ساختارهای فایل سیستم خود استفاده کنید. مدل دیسک بسیار ساده است: به طور کلی، فایل سیستم خواندن یا نوشتن را بر روی سکتوری از دیسک را انجام می دهد. در واقعیت، دیسک خواندن و نوشتن را توسط یک آرایه در حافظه انجام میدهد. توابع دیگر API دیسک به شما امکان می دهد محتویات فایل سیستم خود را در یک فایل لینوکس معمولی ذخیره کنید و بعداً فایل سیستم را از آن فایل بازیابی کنید.

### API دیسک

```
int Disk_Init()
```

Disk_Init() باید دقیقاً یک بار توسط سیستم عامل شما قبل از انجام هر گونه عملیات دیسک دیگر فراخوانی شود.

```
int Disk_Init()
```

Disk_Load() برای واکشی محتویات یک فایل سیستم در فایل file در حافظه فراخوانی می شود. این تابع (و Disk_Init() قبل از آن) احتمالاً یک بار توسط کتابخانه شما هنگام "فرایند بوت"، یعنی در طول FS_Boot() اجرا می شود.

```
int Disk_Init()
```

Disk_Save() نمای درون حافظه فعلی دیسک را در فایلی به نام فایل ذخیره می کند. این روال برای ذخیره محتویات "دیسک" شما در یک فایل واقعی استفاده می شود تا بتوانید بعداً دوباره آن را "بوت کنید". این روال احتمالاً توسط FS_Sync () فراخوانی می شود.

```
int Disk_Write (int sector، char* buffer)
```

Disk_Write() داده‌های buffer را در سکتور مشخص شده توسط sector می نویسد. فرض می شود که اندازه بافر دقیقا برابر اندازه sector باشد.

```
int Disk_Read (int sector، char* buffer)
```

Disk_Read() یک سکتور sector را به بافر مشخص شده واکشی میکند. همانند Disk_Write()، فرض می شود که اندازه بافر دقیقا برابر اندازه sector است.
برای همه APIهای دیسک: همه این عملیات‌ها در صورت موفقیت 0 و در صورت شکست 1- برمی گردانند. اگر خرابی وجود داشته باشد، diskErrno را برابر مقداری مناسب قرار دهید. برای جزئیات، کد LibDisk.c را بررسی کنید.

## ساختمان داده‌های دیسک

بخش بزرگی از درک یک فایل سیستم، درک ساختمان‌داده‌های آن است. البته احتمالات زیادی وجود دارد. در زیر یک رویکرد ساده وجود دارد که ممکن است نقطه شروع خوبی باشد.

ابتدا، در جایی روی دیسک باید برخی از اطلاعات عمومی در مورد فایل سیستم را در بلوکی به نام superblock ضبط کنید. این باید در یک موقعیت شناخته شده روی دیسک باشد. در این مورد، آن را به اولین بلوک تبدیل کنید. برای این تکلیف، نیازی نیست چیز زیادی در آنجا ضبط کنید. در واقع، شما باید دقیقاً یک چیز را در superblock ثبت کنید. یک عدد جادویی. هر عددی را که دوست دارید انتخاب کنید، و هنگامی که یک فایل سیستم جدید را راه اندازی می کنید، همانطور که در بخش راه اندازی زیر توضیح داده شده است، عدد جادویی را در superblock بنویسید. سپس، وقتی دوباره با همین فایل سیستم بوت می شوید، مطمئن شوید که وقتی آن superblock را می‌خوانید، عدد جادویی وجود دارد. اگر آنجا نیست، فرض کنید این یک فایل سیستم خراب است (و نمی توانید از آن استفاده کنید).

دقت کنید که یکی از چالش‌های این پروژه، ذخیره و بازیابی فایل‌های بزرگ است. به طور خاص، محاسبه کنید که فایل سیستم شما از حداکثر چه اندازه‌ای برای یک فایل پشتیبانی میکند.

## جدول فایل‌های باز

هنگامی که یک پردازه یک فایل را باز می کند، فایل سیستم نخست یک جستجو برای یافت آن فایل انجام میدهد. با این حال، در پایان جستجو، برای اینکه بتوانید فایل را به طور کارا بخوانید و بنویسید (بدون انجام چندباره جستجوی مسیر) باید برخی از اطلاعات را نگه دارید. این اطلاعات باید در یک جدول به نام جدول فایل‌های باز برای هر فایل نگهداری شود. هنگامی که یک پردازه یک فایل را باز می کند، باید آن را به عنوان اولین مدخل در این جدول قرار دهید. بنابراین، اولین فایل باز شده باید جایگاه را در جدول بگیرد و عدد توصیفگر فایل 0 برگردانده شود. دومین فایل باز شده (اگر اولی باشد هنوز باز است) باید توصیفگر 1 را دریافت کند و .... هر جایگاه جدول هر آنچه که در مورد فایل متناظر آن برای خواندن یا نوشتن کارآمد در آن بدانید را باید نگه دارد. این موضوع را در نظر داشته باشید و جدول خود را بر اساس آن طراحی کنید. اشکالی ندارد که اندازه جدول خود را به اندازه‌ای ثابت محدود کنید.

## پایداری در دیسک

انتزاع دیسکی که در بالا به شما ارائه شد، داده ها را تا زمانی که Disk_Save() فراخوانی شود، در حافظه نگه می دارد. بنابراین، شما باید Disk_Save() را فراخوانی کنید تا تصویر فایل سیستم پایدار بماند. یک سیستم‌عامل واقعی داده‌ها را به‌طور مکرر روی دیسک قرار می‌دهد تا ت ضمین کند که داده‌ها از بین نمی‌روند. با این حال، در این پروژه، شما فقط باید این کار را زمانی انجام دهید که FS_Sync() توسط برنامه‌ای که به LibFS شما link شده فراخوانی شود. برای هر File_Write،File_Read و سایر عملیات FS که با دیسک تعامل دارند، باید Disk_Read و Disk_Write را فراخوانی کنید.

## راه اندازی (booting)

هنگام "بوت کردن" سیستم عامل خود ( راه اندازی آن)، باید فایل سیستم را نیز واکشی کنید.دقت کنید که در بالا حرفی از فایلی زده شد که محتویات دیسک شبیه سازی شده شما را در خود نگه می دارد. اگر چنین فایلی وجود دا شت، میتوانید آن را واکشی کنید (از طریق Disk_Load()، و سپس بررسی کنید و مطمئن شوید که دیسک سالم است. برای مثال، اندازه فایل باید معادل NUM_SECTORS در SECTOR_SIZE باشد، و superblock باید اطلاعاتی را داشته باشد که شما انتظار دارید در آن وجود داشته باشد (همانطور که در بالا توضیح داده شد). اگر هر یک از آن اطلاعات نادرست است، باید خطا را گزارش کرده و خارج شوید.

با این حال، یک وضعیت دیگر وجود دارد: اگر فایل دیسک وجود نداشته باشد، به این معنی است که باید یک دیسک جدید ایجاد کنید و superblock آن را مقداردهی اولیه کنید و یک پوشه ریشه خالی در فایل سیستم ایجاد کنید. بنابراین، در این مورد، باید از Disk_Init() و به دنبال آن چند عملیات Disk_Write() برای مقداردهی اولیه دیسک، و سپس Disk_Save() برای انجام این تغییرات در دیسک استفاده کنید.

## نکات دیگر

* **کش کردن**: فایل سیستم شما نباید هیچ گونه کش کردن انجام دهد. یعنی تمام عملیات‌ها باید API دیسک را فراخوانی کنند.
* **پو شه‌ها**: یک پو شه را به عنوان یک نوع "ویژه" از فایل در نظر بگیرید که اتفاقاً حاوی اطلاعات پو شه ا ست (مانند لینوکس). بنابراین، شما باید بیتی در inode خود داشته باشید که به شما بگوید که آیا این فایل یک فایل معمولی است یا یک پوشه. قالب پوشه خود را ساده نگه دارید: یک فیلد ثابت 16 بایتی برای نام، و یک ورودی 4 بایتی به عنوان شماره inode.
* **حداکثر اندازه فایل**: حداکثر اندازه فایل در این پروژه برابر ۲ گیگابایت میباشد. اگر برنامه ای سعی کند یک فایل (یا پوشه) را بیش از این اندازه بزرگ کند، باید شکست بخورد.
* **حداکثر طول عنصر در نام مسیر**: 16 کاراکتر. شما لازم نیست نگران پشتیبانی از مسیرنامه‌های طولانی باشید. بنابراین، آن را ساده نگه دارید و 16 بایت برای هر ورودی نام در یک فهرست ذخیره کنید.

* اگر File_Write() فقط تا حدی موفق شود (یعنی بخشی از فایل نوشته شود اما فضای دیسک تمام شود)، میتوانید 1- را برگردانید و osErrno را به مقدار مناسب قرار دهید.
* شما نباید اجازه تداخل پوشه و نام فایل را در یک پوشه بدهید (یعنی یک فایل و یک پوشه با همان نام در یک پوشه).
* با فرض اینکه حداکثر طول نام یک فایل 16 بایت باشد به این معنی است که 15 کاراکتر به اضافه یک برای جداکننده انتهای رشته.
* حداکثر طول یک مسیر 256 کاراکتر است.
* حداکثر تعداد فایل های باز 256 می باشد.
* کاراکترهای قانونی نام فایل شامل حروف (حساس به حروف بزرگ)، اعداد، نقطه (".")، خط تیره ("-") و زیرخط ("_") است.

## موارد تحویل پروژه:

فایل های زیر در اینجا برای شما ارائه شده است:
فایل منبع برای انتزاع دیسک:

LibDisk.c

فایل هدر برای انتزاع دیسک:
LibDisk.h

یک نمونه Makefile برای LibDisk:
Make.LibDisk

هدر LibFS:
LibFS.h

فایل منبع LibFS:
LibFS.c

یک نمونه Makefile برای LibFS:
Make.LibFS

یک مثال main.c:
main.c

یک نمونه Makefile:

Make.main

❋**توجه۱**: شما نباید یک خط کد را در LibDisk تغییر دهید.
❋**توجه۲**: شما نباید چیزی را در مورد رابط LibFS (همانطور که در LibFS.h تعریف شده است) تغییر دهید.
❋**توجه۳**: برای استفاده makefile بدون نام "makefile" یا "Makefile"، فقط "make -f Make.main" را تایپ کنید (برای مثال).

## پروژه‌ی دهم:

## عنوان:

پیاده‌سازی شل سفارشی‌شده

## مقدمه:

در این تمرین یک شل مشابه بش در پیاده‌سازی خواهید کرد. هدف یک شل این است که به کاربر یک واسط جهت ارتباط با سرویس‌های سیستم عامل، که شامل مدیریت پرونده‌ها و پردازه‌ها می شود، ارائه دهد.

## مراحل انجام پروژه:

### فرمان‌های ابتدایی:

ساختار کد شل شما یک توزیع‌کننده (job distributer) برای فرمان‌ها دارد. در واقع هر شل یک مجموعه از فرمان‌های درونی دارد که کارکردهای مربوط به خود شل هستند و نه برنامه‌های خارجی. مثلا فرمان quit باعث می‌شود که شل از اجرا خارج شود.

در گام اول دو دستور quit و help را پیادهسازی کنید. با اجرای دستور help می‌بایست دستوراتی که شل شما پشتیبانی می‌کند را به همراه توضیحی مختصر نمایش دهد.

در گام بعدی دو دستور cwd و cd را پیاده سازی کنید. با دستور cwd باید مسیر پوشه فعلی را در خروجی استاندارد چاپ کنید. دستور cd یک ورودی مشابه /path1/path2/path3 می‌گیرد (در این بخش مسیرها کامل هستند و دربخش‌های بعد باید از مسیرهای نسبی پشتیبانی کنید) و مسیر کاری فعلی را به آن تغییر می‌دهد.

### اجرای برنامه:

اگر تلاش کنید چیزی در شل تایپ کنید که از فرمان های داخلی نباشد، شل شما باید یک پیام مبنی بر اینکه نمی داند چگونه باید برنامه را اجرا کند چاپ کند. طوری شل خود را تغییر دهید که هر گاه فرمان اجرای برنامه ای را به آن بدهید، بتواند آن را اجرا کند. اولین کلمهٔ فرمان، نام برنامه و مابقی کلمات ورودی های برنامه خواهند بود. در حال حاضر می‌توانید فرض کنید که مسیر کامل پرونده اجرایی به عنوان ورودی به شما داده می‌شود، مثلا:

```
$ /usr/bin/echo 123
  123
$ /usr/bin/wc somthing.txt
  77 262 1843 shell.c
```

در بخش‌های بعدی تلاش خواهید کرد که به جای پشتیبانی از آدرس کامل برنامه، پشتیبانی از نام ساده آن (wc) را پیاده سازی کنید. وقتی شل بخواهد یک برنامه را اجرا کند، باید با فراخوانی یکی از توابع خانوادهٔ exec یک پردازهٔ فرزند فورک کند. همچنین پردازهٔ والد باید تا اتمام کار پردازهٔ فرزند صبر کرده و سپس منتظر فرمان های بعدی باشد.

### تاریخچه:

شل شما باید با دستور history بتواند تمام دستوراتی که تا به حال اجرا شده است را در خروجی نمایش دهد. هر دستور در تاریخچه را در یک خط چاپ کنید.

**تفکیک پذیری مسیرها:**

هر برنامه‌ای و از جملهٔ آن‌ها برنامهٔ شل، به یک مجموعه از متغیرهای محلی (environment variable) دسترسی دارند که به صورت یک جدول هش از زوج مرتب های کلید و مقدار سازماندهی شده‌اند. یکی از این متغیرهای محلی متغیر PATH است. شما می‌توانید این متغیر را بر روی سیستم‌عامل خود چاپ کنید (نه در شلی که پیاده‌سازی کردید!):

```
$ echo $PATH
  /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:
```

وقتی بش یا هر شل دیگری، بخواهد یک برنامه را اجرا کند، در تمام مسیرهای موجود در متغیر محلی PATH به دنبال برنامه ای با نام PATH می گردد و اولین برنامه‌ای که پیدا کند را اجرا می‌کند. هر پوشه در PATH با استفاده از علامت ':' از سایرین جدا می شود. حال باید شل دست ساز خود را چنان تغییر دهید که از متغیر محلی PATH استفاده کرده و برنامه را با نام سادهٔ آن نیز اجرا کند.

*توجه ۱: باید کماکان از نوشتن مسیر کامل برنامه نیز پشتیبانی شود.*

*توجه ۲: به هیچ وجه از execvp استفاده نکنید وگرنه نمره ای به شما تعلق نخواهد گرفت. در عوض می توانید از execv استفاده کرده و مسیرهای مورد نیاز را خودتان بسازید.*

دقت کنید که در پایان این مرحله می‌توانید از دستوراتی که در بش (شل اصلی سیستم‌عامل شما) تعریف شده است (مانند ls, mkdir و grep و ...)، صرفا با نوشتن اسم آنها (و بدون مشخص کردن آدرس کامل) اجرا کنید.

**اجرای موازی برنامه‌ها:**

در این قسمت می‌خواهیم که در یک خط چندین دستور را همزمان اجرا کنیم. برای این کار از نویسه '; ' استفاده می‌کنیم. در نتیجه چندین دستور را در می‌توان در یک خط نوشت و آنها را با ';' جدا می‌کنیم.

$ ls; ls -l; ls; echo 444; cat a.txt

برای سهولت، فرض کنید بعد از هر نویسه ';' یک نویسه فاصله نیزی وجود دارد. انتظار می‌رود که این دستورات به صورت موازی اجرا شوند. در نتیجه ممکن است که خروجی آنها با یکدیگر تداخل کنند و خروجی به ترتیب اجرای آنها نباشد.

**تعریف دستور دلخواه:**

یک پرونده بنام my_commands.txt بسازید. در این فایل شما می‌توانید دستورات دلخواه خود را تعریف کنید و با نام جدیدی که برای هر یک در نظر گرفتید، آنها را اجرا کنید. برای مثال فرض کنید که محتوی پرونده به شکل زیر است:

```
gst:: git status
lsl:: ls -al
```

در اینجا با دو دستور جدید *gst* و *lsl* را تعریف کردیم که با اجرای آنها در شل باید خروجی دستوری که در جلوی آنها تعریف شده است را بگیریم. هر تعریف باید در یک خط باشد. اسمی که برای دستور جدید در نظر می‌گیرید باید در ابتدای خط بیاید و سپس با " ::" جدا شود.

برای سادگی، فرض می‌شود دستوراتی که تعریف می‌کنید با دستورات داخلی شل و دستوراتی که در PATH پیدا می‌شوند متفاوت است.

**هدایت ورودی خروجی:**

گاهی می‌خواهیم یک برنامه به جای کار کردن با ورودی و خروجی استاندارد، ورودی خود را از یک پرونده بخواند یا خروجی خود را در یک پرونده بنویسد. دستور

```
$ [process] > [file]
```

به شل می‌گوید که خروجی استاندارد پردازه باید در یک پرونده نوشته شود. به طور مشابه دستور

```
$ [process] < [file]
```

به شل می‌گوید که محتوای پرونده را به عنوان ورودی استاندارد پردازه به کار ببرد. شما باید شل خود را به گونه ای تغییر دهید که از هدایت کردن ورودی و خروجی استاندارد به پرونده‌ها پشتیبانی کند. فرض کنید که همواره پیرامون دو نویسهٔ < و > فضای خالی وجود دارد.

**استفاده از لوله (Pipe):**

لوله یا پایپ یکی از روش‌های پرکاربرد برای هدایت ورودی و خروجی است در شل است. این قابلیت به شما اجازه می‌دهد تا خروجی یک دستور را به عنوان ورودی دستور بعدی بدهید. برای مثال دستور زیر را در نظر بگیرید:

```
$ cat a.txt | grep ‘start’
```

در اینجا خروجی دستور cat به عنوان ورودی به دستور grep داده می‌شود. در نتیجه دستور بالا معادل این است که در فایل a.txt تمامی کلمات start را بدست بیاوریم.

در این قسمت شل شما باید از کاراکتر '|' پشتیبانی کند، و خروجی دستورات سمت چپ '|' را به عنوان ورودی دستورات سمت راست '|' باید داده شود. برای سهولت فرض کنید که در هر یک دستور حداکثر یک بار از کاراکتر '|' استفاده می‌شود.

برای پیاده‌سازی این قسمت و قسمت قبل می‌توانید از دستورات pipe، dup و dup2 کمک بگیرید.

**پردازش پس زمینه:**

تاکنون شل به گونه‌ای بوده است که قبل از شروع برنامهٔ بعدی منتظر اتمام برنامه های قبلی می‌ماند. بسیاری از شل‌ها امکان اجرای یک دستور در پس‌زمینه را با قرار دادن علامت & در انتهای خط فرمان فراهم می‌سازند. پس از شروع برنامهٔ پس‌زمینه، شل به شما اجازه می دهد که پردازه های بیشتری را بدون انتظار جهت اتمام پردازهٔ پس زمینه، شروع کنید.

شل را به گونه ای تغییر دهید که فرمان‌هایی که با قالب مذکور وارد می شوند را در پس‌زمینه اجرا کند.

$ sleep 60 &

و بلافاصله پس از اجرای آن با استفاده از دستور ps پردازه‌ی در حال اجرا در پس زمینه را ببینید. می توانید فرض کنید که همواره پیرامون نویسهٔ & فاصله وجود دارد. همچنین فرض کنید که این نویسه آخرین نشان در آن خط فرمان است.

**خطاها:**

توجه کنید که شلی که پیاده‌سازی می‌کنید در هیچ یک از مراحل نباید با خطا به پایان برسد، مثلا core dump یا اتمام ناقص ( premature terminate). در صورتی که اجرای هر یک از دستورات با خطا مواجه شد باید یک پیغام خطای مناسب چاپ شود و شل شما به اجرا ادامه

دهد. برای مثال، اگر برای اجرای فرمانی که دو argument به عنوان ورودی می‌گیرد با تعداد کمتر یا بیشتری argument مواجه شود شل شما چنین پیامی چاپ کند:

*[command]:* Number of command line arguments ([*# arguments*]) are not compatible with your command.

به طرز مشابه برای باقی خطاها به دلخواه خود یک پیام بامعنی چاپ کنید.

**موارد تحویل:**

پیاده‌سازی شما باید به زبان C باشد. شما باید پرونده‌هایی که برای انجام این پروژه ایجاد کرده‌اید (شامل پرونده‌های c. و h.) را در یک پرونده با پسوند zip. قرار دهید. کدهای شما باید به درستی کامپایل شود. به این منظور یک makefile در کف پوشه اصلی ایجاد کنید به گونه‌ای که با اجرای دستور make در ترمینال کدهای شما کامپایل شود و خروجی قابل اجرای با نام shell ایجاد شود. در نتیجه با اجرای پرونده shell می‌بایست شل شما به اجرا دربیاید.

$ make

(starts compiling your codes)

$ ./shell.o

(shell starts)

> pwd

*[current directory]*

*** شل شما می‌بایست هر یک از ویژگی‌هایی که در قسمت‌های قبل توضیح داده شده است را اجرا کند. همچنین به نسبت به چاپ پیغام‌های خطای مناسب توجه کنید.

## پروژه‌ی یازدهم:

### عنوان:

پیاده سازی دستورات مدیریت حافظه

### مقدمه:

هدف این پروژه پیاده سازی د ستورات مدیریت حافظه در کتابخانه ی ا ستاندارد C ا ست. در انجام این پروژه شما با وا سط POSIX و ساختار حافظه مجازی پردازه ها آشنا می‌شوید.

راهنما: صفحات راهنمای رسمی malloc و sbrk مراجع خوبی برای انجام این تمرین هستند.

توجه: بدیهی اســت اســتفاده از دســتورهای اســتاندارد مدیریت حافظه در C مانند malloc، realloc free در این پروژه مجاز نیســت و با هدف آن در تناقض خواهد بود.

### شرح پروژه:

در این پروژه شما باید ۳ تابع my_realloc , my_free , my_malloc را پیاده سازی کنید و از آن ها برای تخصیص حافظه بهره بگیرید. طبیعی است که برای تخصیص حافظه در برنامه حق استفاده از دستورات استاندارد را ندارید. یک فایل تست جهت آشنایی شما به شما داده می‌شود.

### مراحل انجام پروژه:

- شما مختارید که هر تابع و ساختار دلخواهی به فایل my_alloc.h اضافه کنید.
- شما در نهایت مجاز به آپلود دو فایل با پسوند .h و دو فایل با پسوند .c هستید.
- در این پروژه شما باید قابلیت های اضافه ای نیز به تابع های نام برده اضافه کنید که به آنها اشاره خواهیم کرد.
- پیاده سازی الگوریتم **first-fit , buddy memory allocation** اجباری است.
- برای هندل کردن حافظه های تخصیص یافته از یک لینک لیست بهره بگیرید که استراکت مربوطه را نیز در همان هیپ اضافه کنید این بدان معناست که سایزی که به هر درخواست اختصاص می‌دهید برابر سایز درخواست شده و سایز meta است.
- تابع show_stats()
  - این تابع ابتدا تمام بلوک هایی که allocate شده اند به صورت زیر در یک خط چاپ میکند.
  - start_add end_add size
  - start_add end_add size
  - .
  - .
  - .
- سپس تمام بلوک هایی که وضعیت free دارند را چاپ می‌کند.
  - start_add end_add size
  - start_add end_add size
  - .
  - .
  - .
- و در نهایت تمام حجمی که allocate شــده و تمام حجمی که free اســت را چاپ می‌کند ســپس اختلاف پوینتر sbrk را به مقادیر allocate + free نشان می‌دهد.

- allocated size, free size, sbrk(0) - (allocated size + free size)

- تابع set algorithm این تابع تنها یکبار صدا زده می شود. و برای بار های بعدی هیچ کاری نمیکند. اگر پارامتر پاس داده شده صفر با شد یعنی از الگوریتم first-fit و اگر یک با شد از buddy memory allocation ا ستفاده شود. دقت کنید که این تابع پس از حتی یک بار صدا زده شدن هر کدام از دستورات تخصیص یا آزاد سازی حافظه دیگر قابل استفاده نیست و نباید کاری بکند.
- تابع set maximum allocation
  - این تابع یک مقدار را به عنوان ورودی میگیرد و پس از ســـت شـــدن آن دیگر درخواســـت هایی که مقدار حافظه درخواستی شان از آن عدد بیشتر با شد پاسخ داده نمی شوند. به صورت دیفالت مقدار ماکزیمم تخصیص حافظه را بدون محدودیت در نظر بگیرید اگر مقدار پاس داده شده منفی با شد نیز به معنای آن ا ست که مقدار تخ صیص را مجددا بدون محدودیت در نظر بگیرید.
- تابع set minimum allocation
  - مشابه تابع maximum منتها این بار درخواست هایی که مقدارشان از یک عددی کمتر با شد پاسخ داده نمی شوند. اگر مقدار پاس داده شده منفی باشد یا صفر باشد یعنی هیچ محدودیتی برای مینیمم مقدار تخصیص حافظه نداریم.
- تابع my_free
  - اگر اشاره کرد نال پاس داده شده باشد باید هیچ کاری نکنید.
  - یکی از عوارض آزاد سازی حافظه fragmentation است درباره این پدیده در ریپورت گزارش توضیح دهید و توضیح دهید دهید برای کم کردن عوارض آن از چه الگوریتم هایی اســـتفاده می‌کنید. (یکی از الگوریتم های ســـاده به هم پیوستن بلاک های خالی متوالی و در نظر گرفتن آنان به عنوان یک بلاک خالی پیوسته است.)
- تابع my_malloc
  - اگر بلاک خالی به اندازه خواسته شده وجود ندارد باید از sbrk استفاده شود.
  - اگر بلاک خالی موجود از اندازه خواسته شده بیشتر است آن را به دو بلاک تقسیم کنید.
  - اگر بلاک پیدا شده تنها کمتری بزرگ تر از اندازه خواسته شده است آن را قسمت نکنید.
  - اگر نمی‌توانید تخصیص حافظه را انجام دهید نال بازگردانید. اگر سایز خواسته شده صفر است نیز نال برگردانید.
  - یک پارامتر دیگر به نام fill ورودی می‌گیرد که بیانگر آن است که خانه های خواسته شده را با چه مقداری پر کنید. دقت کنید که مقدار آن برابر ۸ بیت است.
- تابع my_realloc
  - اگر نمی‌توانید سایز جدید را اختصاص دهید نال برگردانید.
  - اگر مقدار سایز صفر است معادل دستور free است.
  - اگر پوینتر داده شده نال و سایز عدد مثبت است آنرا معادل دستور malloc در نظر بگیرید.
  - اگر سایز خواسته شده کمتر از بلاک اصلی است بلاک اصلی را دو قسمت کنید.
  - یک پارامتر دیگر به نام fill ورودی می‌گیرد که بیانگر آن است که خانه های خواسته شده را با چه مقداری پر کنید. دقت کنید که مقدار آن برابر ۸ بیت است.
- شما باید برای هر کدام از بخش های بالای گفته شده حداقل یک تست نیز بنویسید.
  - تست کردن تخصیص حافظه و آزاد سازی و تابع show stats
  - تست کردن set maximum , minimum

### لینک های مفید:

برای آشـــنایی بیشـــتر با شـــیوه تخصـــیص حافظه و الگوریتم های پردازش بلاک ها در پروژه به این لینک (Additional Information مراجعه کنید. شما مختارید که از هر کدام از الگوریتم های توضیح داده شده در لینک استفاده کنید.

**پروژه‌ی دوازدهم:**

**عنوان:**

پیاده‌سازی الگوریتم SMQ در حافظه‌ی نهان OpenCAS

**مقدمه:**

## ذخیره‌سازی نهان

امروزه بسیاری از سیستم‌های کامپیوتری از دیسک‌های سخت[۱](HDD) به عنوان حافظه‌های مانا[۲]برای ذخیره‌سازی اطلاعات استفاده می‌کنند. با پیشرفت سریع تکنولوژی و تولید حافظه‌های مبتنی‌بر فلش مانند حافظه‌های حالت جامد[۳](SSD) که سرعت بالاتر و توان مصرفی کمتر در ازای قیمت بالاتر در مقابل دیسک‌های سخت دارند این سوال به وجود آمد که چگونه می‌توان از این حافظه‌ها برای افزایش کارایی استفاده کرد. یکی از روش‌ها برای بهره‌گیری از این حافظه‌ها، روش ذخیره‌سازی نهان[۴]است.

با ایده استفاده از حافظه نهان در پردازنده‌ها برای بهبود عملکرد آن‌ها، آشنا شده‌اید. ذخیره‌ساز نهان نیز از همین ایده استفاده می‌کند با این تفاوت که در این حالت، حافظه نهان برای بهبود عملکرد دسترسی به حافظه‌های مانا استفاده می‌شود به گونه‌ای که از یک حافظه‌ی با ظرفیت پایین اما با سرعت بالا مانند حافظه‌های SSD به عنوان حافظه نهان برای حافظه کم سرعت و با ظرفیت بالا مانند دیسک‌های HDD استفاده می‌شود. یکی از مزایای اصلی این روش این است که نیازی به تغییر در زیرساخت موجود نیست و با اضافه کردن این حافظه به عنوان حافظه نهان می‌توان عملکرد سامانه کامپیوتری را بهبود بخشید.

برای پیاده‌سازی ذخیره‌سازی نهان، ابزارهایی در سطح هسته سیستم‌عامل لینوکس وجود دارد که انجام این کار را راحت‌تر می‌کنند. این ابزارها، در لایه نگاشت دستگاه[۵]در هسته لینوکس پیاده‌سازی می‌شوند و مدیریت جایابی داده‌ها را برعهده می‌گیرند. به عبارت دیگر این ابزارها سازوکارهایی[۶]را در اختیار کاربر می‌گذارند که می‌توند از طریق سفارشی‌سازی[۷]ابزار متناسب با کاربرد خود، مدیریت جایابی داده را تنظیم کنند و عملکرد سامانه خود را بهبود دهند.

ابزارهای متنوعی برای بهره‌گیری از ذخیره سازی نهان وجود دارد که برخی از آن‌ها به صورت متن‌باز در اختیار توسعه‌دهندگان قرار گرفته است. بدین منظور دو مورد از بهترین ابزارها ذخیرسازی نهان یعنی OpenCAS و EnhanceIO انتخاب شده‌اند. این ابزارها از حافظه‌های SSD به عنوان حافظه نهان برای دیسک‌های HDD پشتیبانی می‌کنند. همچنین هرکدام از این ابزارها سازوکارهایی را برای نهان‌سازی[۸]و الگوریتم‌های برای ترفیع[۹](قرارگیری داده در حافظه سریع‌تر یعنی SSD) و تنزل[۱۰](خارج شدن داده از حافظه سریع‌تر و قرارگیری در دیسک‌های HDD) دارند.

---

[1] Hard Disk
[2] Non-Volatile
[3] Solid State Drive
[4] IO Cache
[5] Device Mapper
[6] Policy
[7] Customization
[8] Caching
[9] Promotion
[10] Demotion

هدف از این پروژه استفاده از ابزارهای معرفی شده جهت پیاده‌سازی، بررسی ذخیره‌سازی نهان، آشنایی با لایه نگاشت دستگاه و تغییرات جزئی در ابزار مورد نظر است.  به عبارت دیگر هر یک از گروه‌ها باید یکی از این ابزارها را کامپایل و آزمایش‌نمایند. سپس کد منبع این برنامه‌ها را مطالعه کنند و تغییرات جزئی در این ابزارها ایجاد نمایند.

## مراحل انجام پروژه (۱)

أ. یک ماشین مجازی لینوکسی نصب نمایید.

ب. این ماشین را پیکربندی کنید به گونه‌ای که یک حافظه سریع ۱ گیگابایتی و یک حافظه کند با ظرفیت ۱۰ گیگابایتی داشته باشد.

برای اینکار راه‌های متفاوتی وجود داد. مثلاً می‌توانید یک دیسک مجازی بر روی یکusb2.0 flash drive  به عنوان حافظه کند و یک دیسک مجازی بر روی دیسک سخت خود بعنوان حافظه سریع بسازید. می‌توانید حافظه کند را یک دیسک مجازی در هر جایی در نظر بگیرید و به جای حافظه سریع از یک ramdisk  استفاده کنید. اما اگر در سیستم خود هم SSD هم HDD دارید، بهترین حالت این است که روی آن‌ها به ترتیب حافظه‌های سریع و کند را بسازید یا پارتیشن‌هایی روی آن‌ها جدا کنید و اجازه کنترل آن‌ها را به ماشین مجازی دهید.

ت. ابزار مورد نظر را کامپایل و نصب کنید.

ث. ابزار fio را برای بررسی عملکرد ابزار ذخیره‌سازی نهان نصب نمایید.

**Rand Read Write 50 50**

```
fio --filename=<Device_Path> --readwrite=randrw --name=randrw --blocksize=4k --rwmixread=50 --rwmixwrite=50  --direct=  --size=30G --ioengine=libaio --iodepth=2 --numjobs=4 --refill_buffers --norandommap --randrepeat=0  --group_reporting
```

**Rand Read Write 30 70**

```
fio --filename=<Device_Path> --readwrite=randrw --name=randrw --blocksize=4k --rwmixread=30 --rwmixwrite=70  --direct=  --size=30G --ioengine=libaio --iodepth=2 --numjobs=4 --refill_buffers --norandommap --randrepeat=0  --group_reporting
```

**Rand Read Write 70 30**

```
fio --filename=<Device_Path> --readwrite=randrw --name=randrw --blocksize=4k --rwmixread=70 --rwmixwrite=30  --direct=  --size=30G --ioengine=libaio --iodepth=2 --numjobs=4 --refill_buffers --norandommap --randrepeat=0  --group_reporting
```

**Sequential Write**

```
fio --filename=<Device_Path> --readwrite=write --name=write --blocksize=128k --direct=1 --ioengine=libaio --size=3G --iodepth=2 --numjobs=4 --refill_buffers --norandommap --randrepeat=0 --group_reporting
```

**Sequential Read**

```
fio --filename=<Device_Path> --readwrite=read --name=read --blocksize=128k --direct=1 --ioengine=libaio --size=3G --iodepth=2 --numjobs=4 --refill_buffers --norandommap --randrepeat=0 --group_reporting
```

## مراحل انجام پروژه (۲)

**فاز اول:**

أ. نصب و کامپایل ابزار OpenCAS

ب. انجام آزمایش‌هایی با FIO برای تست عملکرد ابزار

**فاز دوم:**

توجه شود که برای بخش ترفیع، الگوریتم‌های مختلفی وجود دارد که برای هر یک از این الگوریتم‌ها باید موارد زیر بررسی گردد.

أ. بررسی کد به منظور درک بخش مربوط به ترفیع

ب. بررسی و رسم فلوچارت مربوط به بخش ترفیع

ت. بررسی و رسم معماری مربوط به بخش ترفیع

**فاز سوم:**

تغییر در الگوریتم ترفیع و اضافه کردن الگوریتم Stochastic Multi-Queue (SMQ) به آن

## لینک‌های مفید

- https://open-cas.github.io/
- https://github.com/Open-CAS/open-cas-linux
- https://git.backbone.ws/kolan/backbone-sources/commit/66a636356

## پروژه‌ی سیزدهم:

### عنوان:

پیاده‌سازی الگوریتم **ARC** در حافظه‌ی نهان **OpenCAS**

### مقدمه:

مقدمه‌ی این پروژه همانند پروژه‌ی دوازدهم می‌باشد.

### مراحل انجام پروژه:

**مراحل انجام پروژه (۱) در پروژه‌ی دوازدهم را انجام دهید.**

**فاز اول:**

أ. نصب و کامپایل ابزار OpenCAS

ب. انجام آزمایش‌هایی با FIO برای تست عملکرد ابزار

**فاز دوم:**

توجه شود که برای بخش تنزل، الگوریتم‌های مختلفی وجود دارد که برای هر یک از این الگوریتم‌ها باید موارد زیر بررسی گردد.

أ. بررسی کد به منظور درک بخش مربوط به تنزل

ب. بررسی و رسم فلوچارت مربوط به بخش تنزل

ت. بررسی و رسم معماری مربوط به بخش تنزل

**فاز سوم :**

تغییر در الگوریتم ترفیع و اضافه کردن الگوریتم adaptive replacement cache (ARC) به آن

### لینک‌های مفید

- https://open-cas.github.io/
- https://github.com/Open-CAS/open-cas-linux
- Megiddo, Nimrod, and Dharmendra S. Modha. "Outperforming LRU with an adaptive replacement cache algorithm." *Computer* 37.4 (2004): 58-65.

## پروژه‌ی چهاردهم:

### عنوان پروژه:

پیاده‌سازی الگوریتم FIFO در حافظه‌ی نهان EnhanceIO

### مقدمه:

مقدمه‌ی این پروژه همانند پروژه‌ی یازدهم است.

### مراحل انجام پروژه:

### مراحل انجام پروژه (۱) در پروژه‌ی دوازدهم را انجام دهید.

**فاز اول:**

أ. نصب و کامپایل ابزار EnhanceIO

ب. انجام آزمایش‌هایی با FIO برای تست عملکرد ابزار

**فاز دوم:**

از آنجایی که حجم کد مربوط به EnhanceIO کم است، انتظار می‌رود موارد زیر برای کد EnhanceIO گزارش گردد:

أ. بررسی و رسم فلوچارت مربوط به کد

ب. بررسی و رسم معماری مربوط به کد

**فاز سوم:**

تغییر در الگوریتم ترفیع و اضافه کردن الگوریتم FIFO[1] به آن

### لینک‌های مفید

- https://github.com/stec-inc/EnhanceIO
- https://lwn.net/Articles/538435/

[1] First in first out