به نام خدا

درس سیستم‌های عامل

نیم‌سال دوم ۰۱-۰۰

دانشکدهٔ مهندسی کامپیوتر

دانشگاه صنعتی شریف

مدرس **مهدی خرازی**

تمرین گروهی **دو**

موضوع **زمان‌بندی**

موعد تحویل مستند طراحی **ساعت ۲۳:۵۹ دوشنبه ۲۹ فروردین ۱۴۰۱**

موعد تحویل کد و گزارش نهایی **ساعت ۲۳:۵۹ شنبه ۱۰ اردیبهشت ۱۴۰۱**

با سپاس از دستیاران آموزشی **حسین ذاکری‌نیا، علی احتشامی، حسین احمدزاده، فرزام زهدی نسب، یاشار ظروفچی و صبا هاشمی**

اقتباس شده از **CS162 در بهار ۲۰۲۰ در دانشگاه کالیفرنیا، برکلی**

فهرست مطالب

۰ سرآغاز

شما در این تمرین سامانه‌ی مدیریت ریسه‌های Pintos [۱] را بهبود خواهید داد. در این سند، این ویژگی‌ها به طور مختصر توضیح داده شده‌اند و برای توضیحات بیش‌تر می‌توانید به قسمت منابع در این مستند [۲] مراجعه نمایید.

۱ ماموریت‌های شما

در تمرین گروهی یک هر ریسه‌ای که با آن کار نمودید (به جز init و idle) یک پردازه بود و فضای آدرس خودش را داشت و می‌توانست در فضای کاربر [۳] اجرا شود و داده‌هایش توسط یک پرونده‌ی اجرایی [۴] پشتیبانی می‌شد.
برای این تمرین کار را ساده می‌نماییم و تنها با ریسه‌های هسته [۵] کار خواهیم کرد. در نتیجه ریسه‌ها تنها در حالت هسته [۶] اجرا می‌شوند و اجزای فضای کاربر را ندارند. به طور خاص، ماکروهای USERPROG و FILESYS تعریف نخواهند شد. شما می‌توانید کدتان را بر روی کد نهایی تمرین گروهی اول هم پیاده‌سازی نمایید اما اجباری نیست و می‌توانید از کد اولیه و خام Pintos نیز استفاده نمایید.

- اگر شروع تازه را انتخاب می‌کنید، از وجود داشتن آخرین نسخه‌ی کدتان برای تمرین اول، در یک شاخه‌ی [۷] دیگر اطمینان حاصل نمایید. برای تمرین گروهی سوم به این کد نیاز خواهید داشت.
- اگر می‌خواهید کد پروژه اول‌تان را ادامه دهید، توجه کنید که ماکروهای USERPROG و FILESYS تعریف نشده‌اند. در نتیجه باید سعی نمایید تغییراتی را که در struct thread یا پرونده‌ی thread.c ایجاد کردید را داخل قطعات

```
#ifdef USERPROG
...
#endif
```

قرار دهید. در نتیجه تغییرات‌تان با این تمرین تداخل پیدا نخواهد کرد.

۱.۱ ماموریت ۱: ساعت زنگ‌دار بهینه

در Pintos ، ریسه‌ها می‌توانند با صدا زدن تابع زیر خودشان را در حالتِ خواب قرار دهند:

```
/**
 * This function suspends execution of the calling thread until time has
 * advanced by at least x timer ticks. Unless the system is otherwise idle, the
 * thread need not wake up after exactly x ticks. Just put it on the ready queue
 * after they have waited for the right number of ticks. The argument to
 * timer_sleep() is expressed in timer ticks, not in milliseconds or any another
 * unit. There are TIMER_FREQ timer ticks per second, where TIMER_FREQ is a
 * constant defined in devices/timer.h (spoiler: it's 100 ticks per second).
 */
void timer_sleep (int64_t ticks);
```

تابع timer_sleep() برای ریسه‌هایی که به صورت بی‌درنگ [۸] کار می‌کنند، سودمند است (به عنوان مثال چشمک زدن اشاره‌گر در هر ثانیه). پیاده‌سازی کنونی این تابع بهینه نیست؛ چون تابع thread_yield() در یک حلقه دائما صدا زده می‌شود تا زمان کافی گذشته باشد و این عمل باعث استفاده شدن چرخه [۹] های پردازنده در حالی که ریسه در حال انتظار است، می‌شود. ماموریت شما

[1] Threading System
[2] https://inst.eecs.berkeley.edu/~cs162/sp20/static/projects/proj2.pdf
[3] Userspace
[4] Executable File
[5] Kernel Threads
[6] Kernel Mode
[7] Branch
[8] Real Time
[9] Cycle

پیاده‌سازی مجدد `timer_sleep()` است تا این تابع به صورت بهینه اجرا شود و دیگر Busy Waiting نداشته باشیم.

۲.۱ ماموریت ۲: زمان‌بند اولویت‌دار

در Pintos هر ریسه یک اولویت[10] از ۰ (`PRI_MIN`) تا ۶۳ (`PRI_MAX`) دارد. اما زمان‌بندِ[11] کنونی به اولویت‌ها اهمیت نمی‌دهد. شما باید زمان‌بند را به نحوی تغییر دهید که ریسه‌های با اولویت بالاتر همیشه قبل از ریسه‌های با اولویت پایین‌تر اجرا شوند (زمان‌بندیِ مطلقا بر مبنای اولویت).
همچنین باید هر سه سازوکار به‌هنگام‌سازی[12] (قفل[13] ، سمافور[14] و متغیر شرطی[15]) در Pintos را به شکلی تغییر دهید که این منابع مشترک، ریسه‌های با اولویت بالاتر را بر ریسه‌های با اولویت پایین‌تر ترجیح دهند.
افزون بر این، اهدای اولویت[16] نیز باید برای قفل‌های Pintos پیاده‌سازی شود. زمانی که ریسه‌ی (آ) با اولویتی بالا، منتظر قفلی است که توسط ریسه‌ی (ب) با اولویتی پایین‌تر نگه داشته شده است، به طور موقت اولویت ریسه‌ی (ب) را به اولویت ریسه‌ی (آ) افزایش می‌دهیم. زمان‌بندی که اهدای اولویت نمی‌کند در برابر مشکل وارونگی اولویت[17] آسیب‌پذیر است. این مشکل سبب می‌شود در حالی که ریسه‌ی با اولویت بالای (آ) برای یک منبع که قفل آن در اختیار ریسه‌ی با اولویت پایین (ب) است، منتظر مانده، ریسه‌ای با اولویتی متوسط -مثلا (ج)- اجرا شود. یک زمان‌بند که اهدای اولویت انجام می‌دهد ابتدا به ریسه‌ی (ب) اجازه‌ی اجرا می‌دهد تا کارش را به اتمام برساند و قفل را آزاد کند و پس از آن قفل در اختیار ریسه‌ی (آ) قرار می‌گیرد تا بتواند اجرا شود.
پیاده‌سازی اهدای اولویت شما باید بتواند از موارد زیر پشتیبانی نماید:

۱. امکان اهدای اولویت از منابع مختلف

۲. بازگرداندن اولویت‌ها به مقدار اولیه هرگاه که قفل آزاد شود

۳. امکان اهداهای بازگشتی و تو در تو

هر ریسه می‌تواند با فراخواندن تابع `thread_set_priority(int new_priority)` اولویت خودش را تنظیم نماید. همچنین با فراخواندن تابع `thread_get_priority()` اولویت خودش را دریافت نماید.
اگر یک ریسه، دیگر بیشترین اولویت موثر[18] را نداشت (تابع `thread_set_priority` را فراخوانده است یا یک قفل را آزاد کرده است)، باید پردازنده را فوراً به ریسه‌ای با بیشترین اولویت واگذار کند.

۳.۱ ماموریت ۳: آزمایشگاه زمان‌بندی

برخلاف ماموریت قبلی که نیاز است زمان‌بند Pintos اصلاح شود، در این قسمت شما زمان‌بندهای مختلف را تجربه می‌نمایید. به طوری که جزییات سطح پایین پیاده‌سازی از آنان حذف شده است. به عنوان قسمتی از این تمرین، شما دو زمان‌بند را در یک محیط شبیه‌سازی شده داخل یک دفترچه‌ی IPython پیاده‌سازی، تحلیل و شبیه‌سازی می‌نمایید تا بفهمید که آنان در برابر حجم کارهای مختلف چگونه عمل می‌نمایند. سوالاتی که باید پاسخ دهید در یک سند جداگانه منتشر می‌شوند. کد شما برای «ماموریت ۳: آزمایشگاه شبیه‌سازی» در مدت انجامش نمره‌دهی نمی‌شود و تنها در انتها نمره آن محاسبه می‌شود.
شما می‌توانید دفترچه‌ی IPython را در داخل ماشین مجازی‌تان به صورت زیر اجرا نمایید: ابتدا jupyter را با اجرای دستور

```
sudo apt update && sudo apt install jupyter
```

نصب نمایید. سپس در پوشه‌ای که دفترچه‌ی شما قرار دارد، دستور

[10] Priority
[11] Scheduler
[12] Synchronization
[13] Lock
[14] Semaphore
[15] Condition Variable
[16] Priority Donation
[17] Priority Inversion
[18] Effective Priority

```
jupyter notebook --ip=192.168.162.162
```

را اجرا نمایید. حال در ماشین میزبان باید بتوانید با رفتن به آدرس 192.168.162.162:8888 در مرورگر وب‌تان دفترچه را مشاهده نمایید.

۴.۱ ماموریت اختیاری: زمان‌بند چند سطحه با صف بازخوردی

علاوه بر زمان‌بند اولویت‌دار، می‌توانید زمان‌بند چند سطحه با صف بازخوردی[19] را پیاده‌سازی نمایید. این زمان‌بند با جزییات در قسمت منابع توضیح داده شده است. Pintos با توجه به مقدار متغیر `bool thread_mlfqs` (در `thread.c`) یا از زمان‌بند چند سطحه با صف بازخوردی و یا از زمان‌بند اولویت‌دار استفاده می‌نماید. مقدار این متغیر در صورت وجود `mlfqs--` در دستور اجرای Pintos به `true` تغییر می‌نماید.

توجه کنید این قسمت از تمرین اختیاری است و نمره‌ای ندارد.

۲ تحویل‌دادنی‌ها

نمره‌ی شما بر اساس معیارهای زیر تعیین می‌گردد:

- ۱۵ درصد سند طراحی و جلسه بررسی طراحی
- ۶۰ درصد پیاده‌سازی و کد (لازم به ذکر است این نمره براساس نمره نمره‌دهنده خودکار و همچنین نظر TA محاسبه خواهد شد.)
- ۱۵ درصد آزمایشگاه زمان‌بندی
- ۱۰درصد گزارش نهایی و کیفیت کد

۱.۲ سند طراحی و جلسه بررسی طراحی

قبل از این که شروع به کد زدن کنید، بایستی برای پیاده‌سازی خود یک نقشه‌ی راه داشته باشید و بدانید که قصد دارید هر ویژگی را چطور پیاده‌سازی کنید و همچنین باید بتوانید خودتان را قانع کنید که طراحی را به درستی انجام داده‌اید و اشکالی در آن نیست. برای این تمرین گروهی، بایستی که یک مستند طراحی تحویل بدهید و در جلسه‌ی مرور طراحی، شرکت کنید. در این جلسه، دستیاران آموزشی با شما در مورد طراحی مد نظر شما مشورت خواهند کرد و از شما سوالاتی خواهند پرسید و بایستی بتوانید از طراحی خود دفاع کنید.

۱.۱.۲ بررسی اجمالی طراحی

قالب مستند طراحی این پروژه در آدرس `design/project2-design.md` قرار دارد. شما باید این مستند را کامل نمایید و در همان آدرس قرار دهید. مستند طراحی در فرمت Markdown است. برای مشاهده آن می‌توانید در طرشت به آدرس پرونده بروید و آن را در قالب نهایی مشاهده نمایید. برای هریک از ۲ بخش پروژه شما باید طراحی خود را از چهار جنبه (که در ادامه آورده می‌شود) توضیح دهید.

۱. **داده ساختارها و توابع:** هر داده ساختار، متغیرهای global و یا static و typedef ها و یا enum هایی را که به کد اضافه کردید یا تغییر دادید، به صورت کد C (نه شبه‌کد) بیان کنید. همراه کدها توضیحی حداقلی در مورد هدف این تکه‌کد دهید (توضیحات مفصل‌تر در بخش‌های بعدی مطلوب است).

۲. **الگوریتم‌ها:** در این بخش توضیح می‌دهید که چرا کد شما کار می‌کند! توضیحات شما باید مفصل‌تر از توضیحاتی باشد که در سند تمرین آمده است (سند تمرین موجود است و تکرار آن بی‌مورد است).از طرفی در نظر داشته باشید که توضیحات این بخش باید از سطح کد بالاتر باشد و لازم نیست که خط‌به‌خط کد توضیح داده شود. صرفا باید ما را قانع کنید که کد شما نیازمندی مطرح

[19] Multi-Level Feedback Queue Scheduler (MLFQS)

شده را برطرف کرده است. لازم به ذکر است که در این جا باید شرایطی که استثنا و حالت خاص به حساب می‌آیند توضیح داده شوند.
حجم توضیح شما در این بخش بسته به پیچیدگی کار و پیچیدگی طراحی شما متغیر است. توضیحات ساده و خلاصه مطلوب هستند اما در نظر داشته باشید که اگر توضیحات شما مبهم باشد و جزئیات کافی را بیان نکرده باشد باعث کسر نمره خواهد شد.
در ادامه راهنمایی‌هایی برای نوشتن سند مناسب آورده می‌شود:

- برای کارهای پیچیده مانند زمان‌بند اولویت‌دار پیشنهاد می‌کنیم که آن را به بخش‌های کوچک‌تر تقسیم کنید و هر بخش را جداگانه توضیح دهید.از ساده‌ترین شروع کنید و طراحی خود را کامل کنید. به عنوان مثال زمان‌بند اولویت‌دار را می‌توان به بخش‌های زیر تقسیم کرد:
  - انتخاب کردن ریسه‌ی بعدی برای اجرا کردن
  - گرفتن یک قفل
  - آزاد کردن یک قفل
  - محاسبه کردن مقدار اولویت موثر
  - زمان‌بندی اولویت‌دار برای قفل‌ها و سمافورها
  - زمان‌بندی اولویت‌دار برای متغیرهای شرطی
  - تغییر دادن اولویت یک ریسه
- نام متغیر ها و توابع را در `backtick` قرار دهید و با استفاده از **bold** و *italic* و سایر ویژگی‌های Markdown سند خود را گویاتر کنید.
- استفاده کردن از لیست‌ها باعث می‌شود نوشته شما خواناتر شود. اگر پاراگراف شما انسجام کافی ندارد می‌توانید با استفاده از لیست آن را بهبود ببخشید.
- تخمین از حجم توضیحات مورد نیاز برای بخش ساده‌تر (ساعت زنگ‌دار) یک پاراگراف و برای بخش پیچیده‌تر (زمان‌بند اولویت‌دار) دو پاراگراف است.
- انتظار داریم که قبل از نوشتن سند طراحی مقدار خوبی از کد Pintos را خوانده باشید. بدون مطالعه‌ی کد نمی‌توانید الگوریتم‌های خود را به درستی توضیح دهید.

۳. **به‌هنگام‌سازی:** استراتژی خود برای جلوگیری کردن از شرایط مسابقه[۲۰] را توضیح دهید و نشان دهید چرا درست کار می‌کند. موارد زیر را در طراحی در نظر داشته باشید:

- این بخش باید به صورت لیستی از تمامی حالاتی که منجر به دسترسی هم‌زمان به یک منبع می‌شود بیان شود و نشان دهید در هیچ حالتی مشکلی پیش نمی‌آید.
- هسته‌ی یک سیستم‌عامل برنامه‌ای پیچیده و چندریسه‌ای[۲۱] است از این رو باعث می‌شود که زمان‌بندی کارها به خودیِ خود امری پیچیده باشد. الگوریتم زمان‌بندی مناسب، الگوریتمی است که تا حد امکان ساده باشد تا بررسی صحت کارایی آن ممکن باشد.
- الگوریتم شما همچنین باید از نظر زمانی و حافظه نیز بهینه باشد و این جنبه از الگوریتم خود را نیز تحلیل کنید.
- علت به وجود آمدن مشکلات به‌هنگام‌سازی، داده‌های مشترک است. یک رویکرد مناسب برای نشان‌دادنِ درستیِ به‌هنگام-سازی این است که در گام اول، همه‌ی داده‌هایی را که توسط عواملی مستقل (خواه آن عوامل، ریسه‌ها باشند یا روند[۲۲] های مدیریت‌کننده‌های وقفه[۲۳]) مورد استفاده قرار می‌گیرند، شناسایی کنید. سپس در گام دوم، نشان دهید که این داده‌ها همیشه و با هر سناریویی ، مقادیری صحیح و سازگار خواهند داشت.
- لیست‌ها یکی از دلایل متداول مشکلات به‌هنگام‌سازی هستند. توجه کنید که لیست ها در Pintos ، thread-safe نیستند.

[20] Race Condition
[21] Multi-thread
[22] Routine
[23] Interrupt Handler

- فرآیند آزادسازی حافظه نیز می‌تواند منجر به مشکلات به‌هنگام‌سازی شود. اگر شما از اشاره‌گری به `struct thread` استفاده می‌کنید، باید نشان دهید که هنگامی که شما از آن استفاده می‌کنید آن ریسه نمی‌تواند exit کند و حافظه خود را آزاد کند.
- اگر توابع جدیدی می‌نویسید در نظر داشته باشید که آیا این توابع در دو ریسه به صورت هم‌زمان می‌توانند صدا زده شوند یا خیر (در واقع اگر صدا زده شوند مشکلی پیش نیاید) و همچنین اگر توابع‌تان به متغیرهای global یا static دسترسی دارند نشان دهید که مشکلی پیش نمی‌آید.
- مدیریت‌کننده‌های وقفه نمی‌توانند قفل را بگیرند. اگر نیاز به دسترسی به یک متغیرِ به‌هنگام‌شده[۲۴] در فرآیند مدیریت کردن آن داشتید راه‌کار خاموش کردن وقفه‌ها را در نظر داشته باشید.
- اگر ریسه‌ای قفلی را در اختیار دارد، به این معنی نیست که توسط زمان‌بند از اجرا بازداشته[۲۵] نخواهد شد. قفل‌ها فقط تضمین می‌دهند که در هر لحظه، تنها یک ریسه می‌تواند در ناحیه‌ی بحرانی[۲۶] باشد.

۴. **منطق:** توضیح دهید چرا طراحی شما از دیگر روش‌هایی که بررسی کردید بهتر است و کاستی‌های آن را شرح دهید. مثلاً، به این نکات توجه داشته باشید: چقدر طراحی قابل درک است؟ تا چه اندازه برنامه‌نویسی آن زمان‌بر است؟ پیچیدگی الگوریتم‌های شما از نظر زمانی و حافظه چقدر است؟ آیا می‌توان با هدف افزودن ویژگی‌های بیشتر به این طراحی، به راحتی این طراحی را تغییر داد؟

۲.۱.۲ سوال‌های افزون بر طراحی

شما باید به این سوالات در مستند طراحی خود پاسخ دهید:

۱. در کلاس، سه مشخصه‌ی مهم ریسه‌ها را که سیستم عامل هنگامی که ریسه در حال اجرا نیست ذخیره می‌کند، بررسی کردیم: program counter، stack pointer و registers. بررسی کنید که این سه در کجا و چگونه در Pintos ذخیره می‌شوند؟ مطالعه switch.S و تابع `schedule` در فایل `thread.c` می‌تواند مفید باشد.

۲. وقتی یک ریسه‌ی هسته در Pintos تابع `thread_exit` را صدا می‌زند، کجا و به چه ترتیبی صفحه‌ی شامل پشته و TCB یا `struct thread` آزاد می‌شود؟ چرا این حافظه را نمی‌توانیم به کمک صدا زدن تابع `palloc_free_page` داخل تابع `thread_exit` آزاد کنیم؟

۳. زمانی که تابع `thread_tick` توسط timer interrupt handler صدا زده می‌شود، در کدام پشته اجرا می‌شود؟

۴. یک پیاده‌سازی کاملا کاربردی و درست این پروژه را در نظر بگیرید که فقط یک مشکل درون تابع `sema_up()` دارد. با توجه به نیازمندی‌های پروژه سمافورها (و سایر متغیرهای به‌هنگام‌سازی) باید ریسه‌های با اولویت بالاتر را بر ریسه‌های با اولویت پایین‌تر ترجیح دهند. با این حال، پیاده‌سازی ریسه‌های با اولویت بالاتر را بر اساس اولویت مبنا[۲۷] به جای اولویت موثر[۲۸] انتخاب می‌کند. اساساً اهدای اولویت زمانی که سمافور تصمیم می‌گیرد که کدام ریسه رفع مسدودیت شود، تاثیر داده نمی‌شود. تستی طراحی کنید که وجود این باگ را اثبات کند. تست‌های Pintos شامل کد معمولی در سطح هسته (مانند متغیرها، فراخوانی توابع، جملات شرطی و ...) هستند و می‌توانند متن چاپ کنند و می‌توانیم متن چاپ شده را با خروجی مورد انتظار مقایسه کنیم و اگر متفاوت بودند، وجود مشکل در پیاده‌سازی اثبات می‌شود. شما باید توضیحی درباره این که تست چگونه کار می‌کند، خروجی مورد انتظار و خروجی واقعی آن فراهم کنید.

۳.۱.۲ بازخورد طراحی

شما در یک جلسه ی ۲۵-۳۰ دقیقه‌ای، طراحی خود را به دستیاران آموزشی پروژه ارائه می‌دهید. در آن جلسه باید آماده باشید تا به سوالات دستیار آموزشی در مورد طراحی خود پاسخ دهید و از طراحی خود دفاع کنید.

[24] Synchronized Variable
[25] Preempted
[26] Critical Section
[27] Base Priority
[28] Effective Priority

۴.۱.۲ نمره‌دهی

مستند طراحی و بازخورد طراحی با هم نمره‌دهی می‌شوند. این بخش ۱۵ نمره دارد که بر اساس توضیحات شما از طراحی در مستند طراحی و پاسخ‌دهی به سوالات در جلسه‌ی بازخورد طراحی نمره‌دهی می‌شود. باید حتما در جلسه بازخورد طراحی حضور داشته باشید تا نمره‌ای به شما تعلق گیرد.

۲.۲ پیاده‌سازی

نمره‌ی پیاده‌سازی شما توسط نمره‌دهنده‌ی خودکار داده می‌شود. Pintos یک مجموعه تست دارد که می‌توانید خودتان آن را اجرا کنید. دقیقا همین تست‌ها برای نمره‌دهی شما استفاده می‌گردد. لازم به ذکر است با تغییر دادن تست‌ها تغییری در تست‌هایی که سامانه‌ی داوری اجرا می‌کند ایجاد نمی‌شود و نمره‌ای که از آن به‌دست می‌آید، ملاک است.

۳.۲ گزارش نهایی

نمره‌دهی گزارش شما بر مبنای دو چیز است: **اول**، بایستی برای هر commit، پیام دقیقی نوشته باشید. بدین منظور پس از مشخص کردن پرونده‌هایی که قصد دارید آنها را commit کنید، فرمان زیر را اجرا کنید.

```
git commit
```

بعد از این فرمان، برای شما ویرایشگری باز خواهد شد که در آن پیام خود را بنویسید. پیام شما باید به گونه‌ای شفاف باشد که هم‌گروهی شما با خواندن فقط همین پیام، متوجه وضعیت کنونی پروژه شود. تلاش کنید طوری این پیام‌ها را بنویسید که حتی بدون نیاز به دیدار حضوری با یکدیگر، کار گروهی خود را انجام دهید و هماهنگ بمانید.

برای نمونه، می‌توانید اسلوب نوشتن چنین پیام‌هایی را در changelog های هسته‌ی سیستم عامل Linux ببینید. بدیهی است که انتظار نوشتن پیام‌هایی به این تفصیل وجود ندارد اما پیشنهاد می‌شود پیام شما حداقل اطلاعات زیر را داشته باشد:

```
Add some feature/Fix some bugs (some should be explained)

Test 27 passed but test 28 and 31 that related to that feature has some issues.
In line ... of file ... this pointer has invalid value that caused that problem(that
    should be explained)
```

به طور خاص، بایستی دقیق بودن پیام‌های خود را هنگام تلفیق کردن انشعاب‌های غیر اصلی در انشعاب `master` رعایت کنید.

دوم ،بعد از اتمام کد پروژه باید یک گزارش از پیاده‌سازی خود آماده کنید. گزارش خود را در مسیر `reports/project2.md` قرار دهید. موارد زیر در گزارش شما مطلوب است:

- تغییراتی که نسبت به سند طراحی اولیه داشتید و دلایلی را که به خاطر آن‌ها، این تغییرات را اعمال کردید، بیان کنید (در صورت لزوم آوردن بحث‌های خود با دستیار آموزشی مانعی ندارد).
- بیان کنید که هر فرد گروه دقیقا چه بخشی را انجام داد؟ آیا این کار را به صورت مناسب انجام دادید و چه کارهایی برای بهبود عملکردتان می‌توانید انجام دهید.

کد شما بر اساس کیفیت کد نیز نمره‌دهی خواهد شد. موارد بررسی از این دست می‌باشند:

- آیا کد شما مشکل بزرگ امنیتی در بخش حافظه دارد (به صورت خاص رشته‌ها در زبان C)؟ memory leak و نحوه مدیریت ضعیف خطاها نیز بررسی خواهد شد.
- آیا از یک Code Style واحد استفاده کردید؟ آیا style مورد استفاده توسط شما با Pintos هم‌خوانی دارد؟ (از نظر فرورفتگی و نحوه نام‌گذاری)
- آیا کد شما ساده و قابل درک است؟

- آیا کد پیچیده‌ای در بخشی از کدهای خود دارید؟ در صورت وجود آیا با قرار دادن توضیحات مناسب آن را قابل فهم کردید؟
- آیا کد Comment شده‌ای در کد نهایی خود دارید؟
- آیا کدی دارید که کپی کرده باشید؟
- آیا الگوریتم‌های linked list را خودتان پیاده‌سازی کردید یا از پیاده‌سازی موجود استفاده کردید؟
- آیا طول خط کدهای شما بیش از حد زیاد است؟ (۱۰۰ کاراکتر)
- آیا در مخزن Git شما، پرونده‌های دودویی حضور دارند؟ (پرونده‌های دودویی و پرونده‌های log را commit نکنید مگر این که واقعا لازم باشند.)