به نام خداوند بخشنده مهربان

دانشگاه صنعتی شریف
مدرس: مهدی خرازی

تمرین گروهی اول *
سیستم‌های عامل

دانشکده‌ی مهندسی کامپیوتر
پاییز ۱۳۹۷

۱. مقدمه

در این تمرین شما باید ویژگی‌های جدیدی را به سیستم ریسه[۱] سیستم‌عامل Pintos اضافه کنید. در اینجا به صورت خلاصه این ویژگی‌ها توضیح داده می‌شوند و اطلاعات تکمیلی در بخش منابع این سیستم‌عامل موجود می‌باشد. صفحه‌ی مستند Pintos و مستند تکمیلی منابع تمرین مراجع خوبی برای این تمرین هستند. برای آن‌که دید خوبی نسبت به این تمرین داشته باشید، پیشنهاد می‌کنیم پیش از ادامه مطالعه این مستند، به مستند Pintos مراجعه کنید و توضیحات کلی بخش مقدمه آن را بخوانید. همچنین لازم است پیش از شروع کد زدن، بخش‌های Appendix A1-A5 ، Appendix B [4.4BSD Scheduler] ، Appendix E [Debugging Tools] و Appendix F [Development Tools] از مستند Pintos را بخوانید. دقت داشته باشید که این پیش‌زمینه در ادامه، کارتان را بسیار ساده‌تر خواهد کرد، بنابراین به هیچ وجه آن را به تاخیر نیندازید.

۲. بخش ۱: ساعت زنگ‌دار بهینه شده

در Pintos ریسه‌ها برای رفتن به حالت غیرفعال[۲] می‌توانند تابع زیر را فراخوانی کنند:

```
/**
* 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
```

* این تمرین برگرفته از تمرین ارائه شده در دانشگاه برکلی، مربوط به درس CS۱۶۲ می‌باشد.
با تشکر از تیم دستیاران آموزشی

[۱] Thread
[۲] Blocked

```
* 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);
```

این تابع برای ریسه‌هایی مناسب است که به صورت real-time فعالیت می‌کنند. (به عنوان مثال برای چشمک زدن نشانه‌گر). در پیاده‌سازی فعلی `timer_sleep()` تابع `thread_yield()` در یک حلقه فراخوانی می‌شود تا جایی که مدت زمان لازم سپری شده باشد و این پیاده‌سازی بهینه نیست. وظیفه‌ی شما پیاده‌سازی مجدد تابع `timer_sleep()` است، به گونه‌ای که بهینه و بدون «busy waiting» اجرا شود.

۳. بخش ۲: زمان‌بند اولویت‌دار

ریسه‌ها در Pintos اولویتی بین ۰(PRI_MIN) و ۶۳(PRI_MAX) دارند ولی در پیاده‌سازی فعلی زمان‌بند [۳]، از مقدار اولویت ریسه‌ها استفاده نمی‌شود. شما باید تغییراتی در این زمان‌بند ایجاد کنید تا همواره ریسه‌های با اولویت بالاتر، قبل از ریسه‌های با اولویت کمتر اجرا شوند. علاوه بر این شما باید ۳ ویژگی اولیه‌ی Pintos برای هماهنگ‌سازی[۴] (قفل، سمافور و متغیرهای شرطی) را به گونه‌ای تغییر دهید که این منابع نیز ریسه‌های با اولویت بالاتر را ارجحیت دهند.

همچنین شما باید Priority donation را برای قفل‌ها در Pintos پیاده‌سازی کنید. هنگامی که یک ریسه با اولویت بیشتر (A) باید منتظر گرفتن قفل بماند و ریسه با اولویت کمتر (B) از قبل این قفل را در اختیار گرفته است، به صورت موقت باید اولویت ریسه‌ی (B) به اولویت ریسه‌ی (A) افزایش یابد.

Priority Donation به این دلیل انجام می‌گیرد تا مشکل ایجاد شده به دلیل Priority Inversion را حل کند. به عنوان مثال فرض کنید که یک زمان‌بند priority donation را انجام ندهد. در این صورت ریسه‌هایی که اولویت بیشتر از B دارند برای اجرا شدن انتخاب می‌شوند ولی اولویت اجرا باید با ریسه‌ی B باشد تا قفلی که در اختیار دارد را زودتر آزاد کند تا ریسه‌ی A آن را در اختیار بگیرد. یک ریسه می‌تواند اولویت خود را با فراخوانی تابع thread_set_priority(its new_priority) تغییر دهد و می‌تواند اولویت فعلی خود را با فراخوانی تابع thread_get_priority() بدست آورد.

اگر یک ریسه‌ی دیگر بیشترین اولویت بین سایر ریسه‌ها را نداشته باشد (یا یک قفل را آزاد کرده باشد و یا تابع thread_set_priority() را با مقداری کمتر از مقدار فعلی اولویت خود صدا زده باشد)، باید بلافاصله پردازنده را به ریسه‌ای با بیشترین اولویت واگذار کند.

[۳] Scheduler
[۴] Synchronization

۴. بخش ۳: Multi-level Feedback Queue Scheduler (MLFQS)

علاوه بر زمان‌بند اولویت‌دار شما باید یک الگوریتم زمان‌بندی چند سطحه با صف بازخوردی نیز پیاده‌سازی کنید که با جزئیات در بخش منابع توضیح داده شده است. بسته به مقدار متغیر «bool thread_mlfqs» ، زمان‌بند Pintos باید از یکی از دو الگوریتم پیاده‌سازی شده توسط شما استفاده کند. این متغیر با دستور `--mlfqs` در خط فرمان تغییر وضعیت می‌دهد.

۵. تحویل دادنی‌ها

نمره‌ی تمرین شما شامل ۳ بخش است:

- ٪۲۰ مستند طراحی و بازخورد طراحی
- ٪۷۰ کد و پیاده‌سازی
- ٪۱۰ گزارش نهایی و کیفیت کد نوشته شده

۱.۵. مستند طراحی و بازخورد طراحی

قبل از شروع به پیاده‌سازی، شما باید مستندی شامل مراحل پیاده‌سازی برای هر یک از ویژگی‌های یاد شده را آماده کنید و مطمئن شوید که طراحی شما مناسب و صحیح است. برای این تمرین شما باید یک مستند طراحی را تحویل داده و از دستیار آموزشی اختصاص داده شده به خود، بازخورد مناسب بگیرید.

۲.۵. دستورالعمل مستند طراحی

مستند طراحی را در فایل doc/project1.md که قبلا در مخزن گیت گروه شما ساخته شده بنویسید. شما باید از مارکدان[۵] برای قالب مستند طراحی خود استفاده کنید. شما می‌توانید پیش‌نمایش مستند طراحی خود را روی واسط وب گیت با رفتن به آدرس زیر مشاهده کنید: (بخش group1 را با شماره گروه خود جای‌گذاری کنید.)

https://tarasht.ce.sharif.ir/ce424-971-groups/ce424-971-group1/src/master/doc

برای هر یک از بخش‌های این تمرین شما باید چهار جنبه‌ی زیر از تمرین را با توجه به مستند طراحی خودتان توضیح دهید. توصیه‌ی ما این است که یک بخش برای هر یک از ۳ بخش تمرین ایجاد کرده و سپس زیربخش‌هایی را برای جنبه‌های مختلف تمرین خود بسازید.

[۵] Markdown

۱.۲.۵. ساختار داده‌ها و توابع

معنای تمامی struct ها، متغیرهای سراسری و ثابت، typedefها یا enumهایی که اضافه می‌کنید و یا تغییر می‌دهید. تمامی موارد بالا باید به زبان C بوده و نه به صورت سودوکد. همچنین به صورت خلاصه درباره‌ی دلیل تغییر هر بخش توضیح دهید. توضیحات شما باید تا حد امکان خلاصه باشند. برای سایر بخش‌ها باید به صورت کاملا جزیی و دقیق توضیحات خود را بنویسید.

۲.۲.۵. الگوریتم‌ها

در این بخش شما باید توضیح دقیقی از نحوه‌ی کار کردن کد خود ارائه کنید. توضیحات شما باید در حد توضیحات ارائه شده برای مستند Pintos باشند. از بازنویسی توضیحات ارائه شده برای پروژه یا در مستند Pintos خودداری کنید. از طرف دیگر توضیحات شما باید گویاتر از کد نوشته شده توسط شما باشد ولی توضیح خط به خط در مورد کد نیز لازم نیست. در نوشته‌ی خود سعی کنید تا ما را قانع کنید که کد شما تمامی خواسته‌های صورت تمرین را برآورده می‌کند. (حتی حالت‌های خاص نامتعارف)

حجم نوشته‌های این بخش بستگی به پیچیدگی هر بخش و پیچیدگی طراحی شما دارد. توضیحات باید ساده باشند و توضیحات مبهم منجر به کسر نمره از شما خواهد شد. چند نکته:

- برای بخش‌های پیچیده مانند زمان‌بند اولویت‌دار توصیه می‌کنیم که این بخش را به چند زیربخش تقسیم کنید. الگوریتم خود را برای هر زیربخش به صورت مجزا توضیح دهید. با ساده‌ترین اجزا شروع کرده و کم کم طراحی خود را جلو ببرید. برای مثال بخش الگوریتم‌های شما برای زمان‌بند اولویت‌دار می‌تواند زیربخش‌های زیر را داشته باشد:
  - انتخاب ریسه‌ی بعدی برای اجرا
  - در اختیار گرفتن قفل‌ها
  - آزاد کردن قفل‌ها
  - محاسبه‌ی اولویت موثر[۶]
  - زمان‌بندی اولویت‌ها برای قفل‌ها و سمافورها
  - زمان‌بندی اولویت‌ها برای متغیرهای شرطی
  - تغییر دادن اولویت یک ریسه
- اسم متغیرها و توابع را در `backtick` قرار دهید. (مثلا تابع `thread_get_nice`) از **bold** و ***italic*** و بقیه مارکدان‌ها برای افزایش خوانایی مستند طراحی خود استفاده کنید.

[۶] effective priority

- اگر وابستگی متنی در پاراگراف شما کم است، بهتر است برای افزایش خوانایی توضیحات خود از لیست استفاده کنید.

- تمامی ویژگی‌های ضروری را در توضیحات خود بیاورید. مقدار مناسب برای بخش‌های ساده تر (مثلا Alarm Clock) یک پاراگراف و برای بخش‌های پیچیده تر دو صفحه اسکرین (حدود ۶۰ خط) است. (مثلا برای Priority Scheduler)

- برای نوشتن مستند طراحی لازم است که بخش‌های زیادی از کدهای Pintos را خوانده باشید. در صورتی که جزئیات ساختار Pintos را ندانید، نمی‌توانید توضیحات خوبی از الگوریتم خود بدهید.

۳.۲.۵. هماهنگ‌سازی

روش خود برای جلوگیری از شرایط مسابقه [۷] توضیح دهید و توجیه کنید چرا این راهکار در همه‌ی حالت‌ها کار می‌کند. برای نوشتن این بخش به این نکات توجه کنید:

- این بخش باید به صورت **لیستی از تمام حالت‌های ممکن از دسترسی‌های همزمان به منابع مشترک** باشد. در هر یک از این حالت‌ها باید نشان دهید که روش هماهنگ‌سازی شما، رفتار درست را انجام میدهد.

- چون هسته‌ی سیستم عامل یک پردازه‌ی چندـ‌ریسه ای پیچیده است، حالت‌های هماهنگ‌سازی ریسه‌های مختلف در آن می‌تواند پیچیده باشد. برای همین روش‌های ساده، که به راحتی بتوان درستی آن‌ها را در حالت‌های مختلف بررسی کرد، روش‌های بهتری هستند. اگر روش شما به راحتی قابل توضیح دادن نیست، بهتر است دنبال راه‌کاری برای ساده سازی آن باشید.

- همچنین سعی کنید که روش شما، از نظر زمانی و حافظه، بهینه باشد.

- مشکلات هماهنگ‌سازی به دلیل منابع مشترک اتفاق می‌افتد. یک راه خوب برای استدلال کردن در مورد هماهنگ‌سازی این است که مشخص کنید چه داده‌هایی بین بخش‌های مختلف (مثل ریسه‌ها یا رسیدگی‌کننده‌ی وقفه[۸]) مشترک است و بعد نشان دهید که این داده‌ها بدون مشکل هستند.

- لیست‌ها در Pintos به صورت thread-safe نیستند، و یکی از مشکلات رایج هماهنگ‌سازی به شمار می‌روند.

- فراموش نکنید که بازپس‌گیری حافظه [۹] می‌تواند باعث ایجاد مشکل هماهنگ‌سازی شود. اگر از اشاره‌گرها به `struct thread` استفاده شود باید نشان دهید که به پایان رسیدن آن ریسه باعث بازپس‌گرفته شدن آن `struct` نمی‌شود.

---

[۷] Race Condition
[۸] Interrupt Handler
[۹] memory deallocation

- اگر تابع جدید می‌سازید باید در نظر داشته باشید که تابع شما می‌تواند هم‌زمان از دو ریسه‌ی مختلف صدا زده شود. اگر این تابع از متغیری سراسری یا ثابت استفاده می‌کند نباید هیچ مشکل هماهنگ‌سازی در آن به وجود آید.

- رسیدگی‌کننده‌های وقفه نمی‌توانند قفل دراختیار بگیرند. اگر در رسیدگی به یک وقفه نیاز به استفاده از یک منبع مشترک دارید وقفه‌ها را غیرفعال کنید.

- قفل‌ها جلوی قبضه[۱۰] شدن ریسه‌ها را نمی‌گیرند. قفل‌ها فقط تضمین می‌کنند که در هر زمان فقط یکی از ریسه‌ها بتواند وارد بخش بحرانی شود.

۴.۲.۵. بنیاد و پایه

توضیح دهید چرا طراحی شما از دیگر روش‌هایی که بررسی کردید بهتر است. و کاستی‌های آن را شرح دهید. به نکته‌هایی مثل اینکه چقدر طراحی قابل درک است، چقدر برنامه‌نویسی آن زمان‌بر است و پیچیدگی الگوریتم‌های شما از نظر زمانی و حافظه چقدر است و آیا می‌توان به راحتی این طراحی را تغییر داد تا ویژگی‌های بیشتری را بتوان در آن قرار داد توجه داشته باشید.

۳.۵. سوالات افزوده برای سند طراحی

.علاوه به بر بخش‌های قبل، در مستند طراحی خود باید پاسخ این سوالات را نیز بدهید:

۱. یک پیاده سازی از این پروژه فرض کنید که در آن همه چیز به درستی رعایت شده به جز مشکل در تابع `sema_up` . طبق خواسته‌های پروژه، سمافورها (و بقیه‌ی متغیرهای همگام‌سازی) باید به ریسه‌هایی با اولویت بیشتر ارجحیت شوند. اما مشکل پیاده‌سازی گفته شده این است که انتخاب ریسه‌ها بر اساس **اولویت پایه‌ی** آنهاست و نه **اولویت موثر** و در واقع هنگامی که سمافور تصمیم میگیرد کدام ریسه آزاد شود، priority donation ها در نظر گرفته نمی‌شوند. تست‌ـ کیسی طراحی کنید که می‌تواند وجود این مشکل را نشان دهد.

تست کیس‌های Pintos کدهای سطح هسته‌ای هستند، که صحتِ پیاده سازی ویژگی ای را بررسی می‌کند و بر اساس آن خروجی‌های مختلفی چاپ می‌کند. با اجرا کردن آن‌ها و مقایسه خروجی چاپ شده با خروجی مورد نظر می‌توان بررسی کرد که آن ویژگی به درستی پیاده‌سازی شده است یا خیر. **توضیح دهید که تستی که طراحی کردید به چه شکل کار می‌کند و خروجی حالت صحیح، و حالتی که در آن این مشکل وجود دارد را مشخص کنید.**

۲. سوال در مورد **MLFQS**: فرض کنید مقدار nice value ریسه‌های A ، B ، و C به ترتیب برابر ۰، ۱ و ۲ باشد. و مقدار `recent_cpu` همه‌ی آن‌ها ۰ است. جدول زیر را پر کنید. در این جدول R(A) و P(A) به

[۱۰] preempt

معنی `recent_cu` و `priority` ریسه‌ی A و در آخرین ردیف ریسه‌ی انتخاب شده برای اجرا در چند تیک ساعت بعدی است.

Timer ticks	R(A)	R(B)	R(C)	P(A)	P(B)	P(C)	thread to run
0							
4							
8							
12							
16							
20							
24							
28							
32							
36							

۳. آیا ابهامی در توصیفات پیاده‌سازی باعث غیرقطعی بودن مقدارهای جدول سوال قبل شد؟ اگر پاسخ مثبت است، توضیح دهید از چه روشی برای رفع کردن آن استفاده کردید؟

۴.۵ بازخورد طراحی

شما در یک جلسه‌ی ۲۰ ـ ۲۵ دقیقه ای، طراحی خود را به TA پروژه ارائه می‌دهید. در آن جلسه باید آماده باشید تا به سوالات TA درمورد طراحی خود پاسخ دهید و از طراحی خود دفاع کنید.

۵.۵ نمره دهی

مستند طراحی و بازخورد طراحی با هم نمره‌دهی می‌شوند. این بخش ۲۰ نمره دارد که بر اساس توضیحات شما از طراحی در مستند طراحی و پاسخ‌دهی شما به سوالات در جلسه ی بازخورد طراحی نمره دهی می‌شود. باید حتما در جلسه‌ی بازخورد طراحی حضور داشته باشید تا نمره‌ای به شما تعلق بگیرد.

۶. پیاده‌سازی

نمره‌ی پیاده سازی شما توسط نمره‌دهنده‌ی خودکار داده می‌شود. Pintos یک مجموعه تست دارد که می‌توانید خودتان آن را اجرا کنید. شما در هر زمان می‌توانید نمره‌ی کد زده شده‌ی خود را با وارد شدن در نمره‌دهنده‌ی خودکار بررسی کنید. این نمره در یک پرونده به نام grade.txt در پوشه‌ی مربوط به هر تمرین قرار می‌گیرد.

۷. گزارش نهایی و کیفیت کد

بعد از تمام شدن کد پروژه، باید گزارش نهایی خود را ارسال کنید. این گزارش را در ‘reports/project1.md’ در مخزن گیت مربوط به گروه خود بنویسید. گزارش نهایی شما باید شامل این بخش‌ها باشد:

- تغییراتی که از سند طراحی اولیه خود دادید، و توضیح اینکه چرا این تغییرات مورد نیاز بوده است. دوباره تکرار کردن مباحثی که در جلسه با دستیاران آموزشی بیان شدند اشکالی ندارد.
- تقسیم وظایف گروه، در مورد بخش‌های مختلف پروژه؛ چه بخش‌هایی از پروژه به خوبی پیش رفت، و کدام بخش‌های می‌توانست بهبود یابد.

نمره‌ی شما بسته به کیفیت کدی است که پیاده‌سازی کرده‌اید و این مساله به فاکتورهای مختلفی بستگی دارد:

- مدارکی که نشان‌دهند کد شما از مشکلات امنیتی حافظه رنج نمی‌برد (مخصوصا مشکلات مربوط به استرینگ C)، memory leaks، مدیریتِ ضعیف اورها یا شرایط مسابقه
- نوشتن کد به صورت پیوسته. همچنین کد شما باید با کد قبلی Pintos ترکیب شود. ایندنت‌ها، فاصله‌بندی و استانداردهای نام‌گذاری
- ساده و خوانا بودن کد
- کامنت‌گذاری در بخش‌های پیچیده‌ی کد
- نبود کد کامنت شده در ارسال نهایی
- نبود کد کپی و پیست شده و وجود توابع مناسب
- اینکه الگوریتم‌های مربوط به linked listها را دوباره پیاده‌سازی کرده‌اید یا اینکه از الگوریتم ارائه شده استفاده کردید.
- طولانی نبودن بیش از اندازه‌ی خط‌های کد (بیشتر از ۱۰۰ کاراکتر) کامیت‌های موجود در گیت شما. (آبجکت‌فایل‌ها و لاگ‌ها را کامیت نکنید مگر اینکه واقعا لازم باشند)

شما می‌توانید کد مربوط به Pintos را در پوشه‌ی pintos، بخش مربوط به مستند طراحی را در پوشه‌ی doc و بخش مربوط به گزارش نهایی پروژه را در پوشه‌ی reports در مخزن تمارین گروهی بیابید.