به نام خدا

سیستم‌های عامل (بهار ۱۴۰۱)

فاز دوم پروژه

استاد درس:

دکتر جوادی

مهلت نهایی ارسال پاسخ:

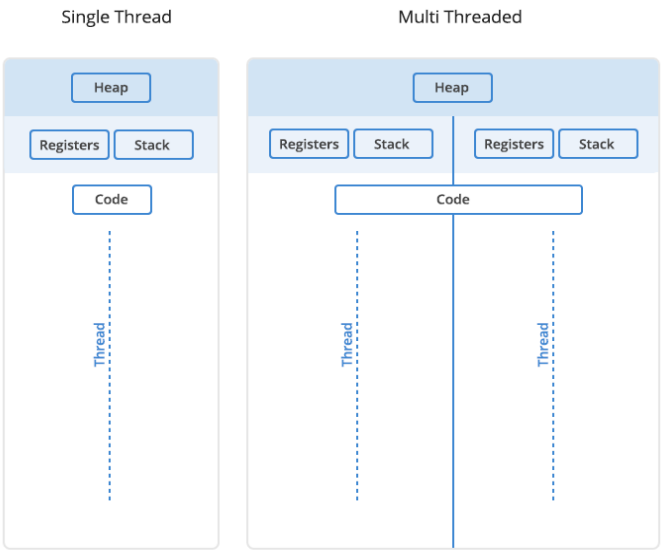
روز ۹ اردیبهشت ۱۴۰۱

ساعت ۲۳:۵۹

نکته مهم: دقت کنید که تمدید نخواهیم داشت و تحویل اسکایپی خواهید داشت و تنها دانشجویانی که فاز دوم را به موقع انجام داده‌اند، خواهند توانست وارد فاز سوم شوند.

مقدمه

در درس برنامه‌نویسی پیشرفته با مفهومی به نام ریسمان (thread) آشنا شدید. همان‌طور که می‌دانید، ریسمان‌ها به ما کمک می‌کنند تا بتوانیم عملیات‌های یک پردازه را به صورت موازی پیش ببریم. یک پردازه می‌تواند شامل یک ریسمان یا بیش از یک ریسمان باشد. در شکل زیر می‌توانید یک دید کلی از چندریسمانی و تفاوت آن با تک ریسمانی را مشاهده کنید.



در فاز دوم پروژه درس سیستم‌های عامل، ما از شما می‌خواهیم تا مکانیزم ایجاد و استفاده از ریسمان‌ها را در XV6 پیاده‌سازی کنید. و بعد از آن با استفاده از این مکانیزم یک مسئله را حل کنید. با انجام این فعالیت یک فهم عمیق از مفهوم ریسمان خواهید داشت و یک مبنای قوی برای برنامه‌نویسی‌های چند نخی شما در آینده خواهد بود.

فاز دوم (پیاده‌سازی Thread)

در ابتدا شما باید دو فراخوان سیستمی مهم را به منظور ساخت یک ریسمان پیاده‌سازی کنید.

```
- int thread_create(void *stack)
- int thread_id()
- int thread_join(int id)
```

فراخوانی سیستمی `thread_create`

فراخوانی سیستمی `thread_create` یک پردازه جدید را ایجاد می‌کند؛ اما پردازه ایجاد شده یک تفاوت خیلی مهم با پردازه ایجاد شده توسط دستور `fork` دارد. پردازه ایجاد شده توسط `thread_create` از همان فضای آدرسی که متعلق به پردازه والد است استفاده می‌کند و به اصطلاح، به هنگام ایجاد پردازه جدید، فضای آدرس کپی نمی‌شود، فقط وضعیت پردازه جدید به شکلی تغییر می‌کند که پردازه والد و فرزند از یک فضای آدرسی مشترک استفاده می‌کنند.

در شکلی که در مقدمه پروژه آورده شد، مشاهده کردید که هریک از فرآیندهای والد و فرزند نیاز به یک پشته جدا دارند. برای اینکار نیاز است تا با استفاده از دستور `malloc`، برای هر ریسمان ایجاد شده، یک پشته جدا تولید کنید.

توجه: برای راحتی کار، سعی کنید کار را به بخش‌های کوچک‌تر بشکنید و مرحله به مرحله پیش ببرید و فرآیند اشکال‌یابی را همزمان با توسعه انجام دهید. این نکته فرآیند اشکال‌یابی را به مراتب آسان‌تر خواهد کرد. برای راحتی استفاده فراخوان سیستمی `thread_create`، شما باید یک تابع سطح کاربر به صورت زیر تعریف کنید:

```
- int thread_creator(void (*fn) (void *), void *arg)
```

در مورد این تابع سطح کاربر به نکات زیر توجه کنید:

- این تابع دو ورودی می‌گیرد: (۱) ورودی اشاره‌گر به یک تابع و (۲) آرگومان‌های ریسمان ایجاد شده.
- این تابع در ابتدا نیاز دارد تا یک فضای page-aligned برای ریسمان جدید ایجاد کند و به پشته اختصاص دهد.
- پس از آن، باید فراخوان سیستمی `thread_create` را صدا بزند و ID ریسمان جدید را بازگرداند.
- در ریسمان ایجاد شده، باید اشاره‌گر به تابع ارسالی صدا زده شود، همچنین آرگومان‌های داده شده نیز به عنوان پارامترهای ورودی این تابع در نظر گرفته شود.
- به هنگام پایان یافتن تابع ارسالی، باید پشته خالی شود و exit صدا زده شود.

توجه: در حالت عادی ریسمان‌های متعلق به یک پردازه می‌بایست شناسه‌های (IDs) متفاوتی داشته باشند. با این وجود در این پروژه اشکالی ندارد که ریسمان‌های یک پردازه id مشابه داشته باشند. در حقیقت می‌توانید شناسه پردازه (pid) را thread id درنظر بگیرید تا با حداقل تغییرات پیاده‌سازی را انجام دهید.

فراخوانی سیستمی `thread_id`

فراخوان سیستمی `thread_id` باید شناسه ریسمان ایجاد شده را به عنوان خروجی باز گرداند.

فراخوان سیستمی `thread_join`

فراخوان سیستمی thread_join همانند فراخوان سیستمی wait می‌باشد. این فراخوان شناسه یک ریسمان ایجاد شده را به عنوان ورودی گرفته و پردازه والد را منتظر نگه می‌دارد تا عملیات ریسمان ایجاد شده به پایان برسد. این تابع پس از به پایان رسیدن عملیات توقف، اگر ریسمان ایجاد شده با موفقیت و بدون مشکل، فعالیت خودش را انجام داد، باید خروجی 0 و در صورت وقوع مشکل باید خروجی 1- بدهد.

دقت شود که `thread_join` فقط برای ریسمان ها اعمال می‌شود (یعنی منتظر می‌ماند تا اجرای ریسمان فرزند تمام شود) و فراخوانی سیستمی `wait` فقط برای پردازه‌ها اعمال می‌شود و منتظر می‌ماند تا پردازه فرزند تمام شود. برای اینکار می‌توانید یک متغیر جدید داخل struct proc در فایل proc.h تعریف کنید که این متغیر نشان دهنده‌ی ریسمان یا پردازه بودن است.

توجه: این نکته را مدنظر داشته باشید که هنگام اجرای exit توسط یک ریسمان، درصورت وجود یک ریسمان دیگر برای همان پردازه، پیاده‌سازی شما page table مشترک را نباید آزاد کند. طبیعی است که برای فهم بهتر این بخش نیاز به مطالعه بیشتری خواهید داشت و تیم تدریس‌یاری آماده پاسخگویی به سوالات شما است.

حل مسئله

در این بخش می‌خواهیم با مکانیزمی که ایجاد کردیم، یک مسئله بسیار ساده را حل کنیم. در فایل تست خود دو متغیر عددی را به عنوان ورودی تعریف کنید؛ Limit و Base. از شما می‌خواهیم تا با استفاده از ریسمان‌ها مقدار Base را به قدری افزایش دهید تا به مقدار Limit برسد. برای اینکار، هر ریسمان، مقدار فعلی Base را یکی افزایش می‌دهد و اگر مقدار آن برابر با Limit شد exit را صدا می‌زند، در غیر این صورت باید یک ریسمان جدید ایجاد کند و منتظر بماند تا کار ریسمان ایجاد شده به پایان برسد. بعد از آن با استفاده از دستور thread_id باید در خروجی نشان بدهد که وضعیت ریسمان فرزند آن به چه صورت بوده است (موفق یا ناموفق).

خروجی مورد انتظار باید به این شکل باشد:

```
Base = 0, Limit = 2
[ID] 1 => [Success] 0
[ID] 2 => [Failed] -1
```

از شما درخواست داریم که یک **private repository** در گیت هاب درست کنید و تغییرات کد خود را مرحله به مرحله **commit** کنید و در صورت تمایل می توانید هر یک از تدریس یاران را به پروژه‌ی خود اضافه کنید. دقت کنید که شما نبایستی برنامه‌های خود را با دیگر دانشجویان به اشتراک بگذارید.

آنچه که باید ارسال کنید:

یک فایل زیپ با نام groupId_sid1_sid1.zip (که groupId را با شماره گروه خود و sid را با شماره دانشجویی خود جایگزین کنید) که شامل دو مورد زیر است:

- گزارشی مختصر از آنچه که انجام داده‌اید تا ریسمان‌ها را به XV6 اضافه کنید. ارسال گزارش الزامی است.
- پوشه‌ای که در آن کدهای شما وجود دارد. دقت کنید که تنها و تنها فایل‌هایی را که تغییر داده‌اید یا اضافه کرده‌اید را برای ما بفرستید.

موفق باشید

تیم درس سیستم‌های عامل