به نام خدا

سیستم‌های عامل- گروه ۱ (نیم‌سال دوم ۹۹-۴۰۰)

پروژه پایانترم درس سیستم عامل

پیاده‌سازی ریسمان برای xv6

xv6 threads

آخرین تاریخ بارگذاری پاسخ در courses:

ساعت ۲۳:۵۹ روز ۲۸ تیر ۱۴۰۰ (این تاریخ حقیقتا قابل تمدید نیست)

مقدمه

معمولا یادگیری مبحث مهمی مانند سیستم عامل وقتی کامل می‌شود که برخی از مباحث ان را بر روی یک سیستم عامل آموزشی پیاده‌سازی کنید. در دانشگاه‌های مطرح دنیا از سیستم عامل xv6 به همین منظور استفاده می‌شود. شما نیز در این پروژه با سیستم عامل xv6 آشنا شده و سپس ریسمان یا thread را بر روی آن پیاده می‌کنید.

بخش مقدمه

این بخش برای این است که شما با xv6 آشنا شوید و یک فراخوانی سیستم مثالی (example system call) به آن اضافه کنید و در یک برنامه سطح کاربر، آن را فراخوانی کنید. دقت کنید که انجام حداقل این بخش بهتر از انجام ندادن کل پروژه است. ویدئوهایی در سایت درس قرار داده شده است که شما را برای دانلود کد xv6، کامپایل و اجرای ان راهنمائی می‌کند (کد xv6 در سایت درس نیز قرار داده شده است). توصیه اکید تیم درس ایجاد یک ماشین مجازی لینوکس و انجام کار بر روی آن است. بنابراین برای بخش اول، کافی است است با نحوه کامپایل و اجرای xv6 اشنا شوید و فراخوانی سیستمی hello world را به ان اضافه کنید و در یک برنامه سطح کاربر ان را فراخوانی کنید.

بخش اصلی

شما در این پروژه ریسمان سطح هسته یا kernel thread را برای xv6 پیاده‌سازی می‌کنید. به این منظور، بایستی که چهار فراخوانی سیستمی زیر پیاده‌سازی شوند:

- `int clone(void *stack)`
- `int lock(int *l)`
- `int unlock(int *l)`
- `int join()`

همچنین لازم است که یک تابع سطح کاربر با امضای زیر را پیاده‌سازی کنید که به عنوان تابع پوششی clone ایفای نقش می‌کند و هدف از آن آسان‌تر کردن استفاده از clone است.

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

توجه: کدنویسی خود را از کد پایه و ساده‌ترین حالت شروع کنید و کار خود را با پیاده‌سازی مدیریت فضای حافظه پیچیده، دشوارتر نکنید. دقت کنید که ما نتیجه کار شما را به شکل end-to-end ارزیابی می‌کنیم و بنابراین حتما بایستی که ریسمان سطح هسته پیاده شده باشد و به درستی کار کند تا بتوانید نمره کامل پروژه را بگیرید و سپس روی نمره اضافه آن تمرکز کنید.

همه توابعی که بیان شدند، در صورت رخداد خطائی در حین اجرای آنها، مقدار 1- را به عنوان خروجی برمی‌گردانند. جدول زیر مقادیر بازگشتی این توابع در صورت موفقیت اجرای آنها را نشان می‌دهد.

نام تابع	مقدار خروجی
`Lock and unlock`	0
`clone, join, and thread_create`	شناسه ریسمان یا thread ID

اولین و احتمالا سخت‌ترین گام پروژه نوشتن و عملیاتی کردن تابع clone است. این تابع بسیار شبیه به fork عمل می‌کند با این تفاوت که در این تابع به جای کپی کردن فضای ادرس به یک page directory جدید، تابع clone وضعیت پردازه جدید را به گونه‌ای مقداردهی اولیه می‌کند که پردازه فرزند و پردازه پدر فضای ادرس را به اشتراک بگذارند (به عبارت دیگر از یک page directory مشابه استفاده بکنند). اگر clone به این شکل نوشته شود، دو پردازه فضای حافظه را به اشتراک خواهند گذاشت و این دو عملا ریسمان هستند.

اگرچه دو ریسمان پدر و فرزند فضای ادرس را به اشتراک می‌گذارند، هر کدام به یک پشته (stack) جداگانه نیاز دارند. فضای پشته جدید احتمالا با استفاده از malloc تخصیص داده می‌شود. به عنوان مثال فرض کنید که ریسمان T1 یک کپی از خود به عنوان T2 را ایجاد می‌کند. T1 ابتدا حافظه‌ای به اندازه یک صفحه که page-aligned است را به پشته T2 اختصاص داده و اشاره‌گر به ان را به فراخوانی سیستمی clone ارسال می‌کند. از طرفی دیگر، فراخوانی سیستمی clone پشته ریسمان T1 را به این نقطه از حافظه کپی می‌کند و

ثبات‌های پشته و پایه T2 (stack and base registers) را تغییر می‌دهد تا از پشته جدید استفاده کند. به این نکته ریز و جالب دقت کنید که T2 باید از سابقه اجرای T1 مطلع باشد و این است که به آیتم‌های موجود در پشته T1 به عنوان مقدار اولیه پشته جدید خود نیاز دارد. پر واضح است که استفاده از فضای هیپ (heap) متعلق به T1 به عنوان پشته T2 کمی عجیب است، با این حال پیاده‌سازی این بخش با استفاده از روش‌های دیگر، وقت زیادتری از شما خواهد گرفت.

تابع thread_create وظیفه تسهیل استفاده از clone را برعهده دارد و اشاره‌گر به یک تابع و ارگومان‌های ریسمان را به عنوان ورودی می‌گیرد. این تابع بایستی که کارهای زیر را انجام بدهد:

- یک فضای page-aligned را به پشته جدید اختصاص دهد.
- فراخوانی سیستمی clone را صدا بزند که thread ID را به پدر بر می‌گرداند.
- در ریسمان فرزند، thread_create باید اشاره‌گر به تابع ارسالی را صدا زده و ارگومان‌های ارسالی به عنوان پارامترهای ورودی در اختیار این تابع قرار دهد.
- هنگامی که اجرای تابع ارسالی به اتمام رسید، thread_create بایستی که پشته را خالی کرده و exit را صدا بزند.

دقت کنید که تابع ارسالی به thread_create تنها بایستی که با دستور return به اجرای خود پایان دهد و از exit استفاده نکند. در غیر این صورت، thread_create فرصت خالی کردن فضای استفاده شده به عنوان پشته ریسمان فرزند را نخواهد یافت (در مرحله قبلی ریسمان پدر از فضای heap خود صفحه‌ای را برای پشته فرزند اختصاص داد).

join فراخوانی سیستمی دیگری است که شما پیاده‌سازی می‌کنید. این فراخوانی سیستمی بسیار شبیه به فراخوانی سیستمی wait در xv6 است. join منتظر می‌ماند تا اجرای *ریسمان* فرزند تمام شود و wait منتظر می‌ماند تا *پردازه* فرزند تمام شود.

فراخوانی‌های سیستمی lock و unlock باید رفتار نرمال خود را داشته باشند. هر کدی که از این فراخوانی‌ها استفاده می‌کند بایستی که ابتدا یک متغیر lock را به 0 مقداردهی کند. از طرف دیگری ریسمانی که منتظر به بدست آوردن یک lock است بایستی که به حالت sleep برود و استفاده از مکانیزم spin قابل قبول نیست. واضح است که در پیاده‌سازی unlock بایستی یکی از ریسمان‌های در حال sleep را از خواب بیدار کنید تا lock را در اختیار گیرد. دقت کنید شما در تئوری همه این مفاهیم را آموخته‌اید.

شما همچنین بایستی کدهایی را برای تست پیاده‌سازی خود بنویسید. تدریسیاران نیز تست‌هایی را خواهند نوشت و انتظار می‌رود که پیاده‌سازی شما همه این تست‌ها را با موفقیت پشت سر بگذارد.

بخش نمره اضافی

1) وقتی که یک ریسمان clone می‌شود، ریسمان جدید و ریسمان قدیمی باید توصیف‌گرهای فایل یکسان (same file descriptors) داشته باشند (این بخشی از پروژه اصلی شما است). با کار اضافی، شما

باید قابلیتی را فراهم کنید که وقتی که یک ریسمان یک توصیفگر فایل را باز می‌کند، این توصیفگر فایل جدید باید برای همه ریسمان‌های دیگر قابل رویت باشد.

2) برنامه init پردازه‌های زامبی را جمع‌آوری می‌کند. ***پردازه‌ای*** زامبی است که پردازه پدر بدون فراخوانی wait دستور exit را اجرا کرده است. شما بایستی که init.c را تغییر دهد تا ***ریسمان‌های زامبی*** را جمع‌آوری کند. ریسمانی زامبی است که پدرش بدون فراخوانی join دستور exit را اجرا کرده است.

راهنمائی

در حالت عادی ریسمان‌های متعلق به یک پردازه (process) بایستی که process id یکسان داشته باشند. با این وجود در این پروژه اشکالی ندارد که getpid برای ریسمان‌های متعلق به یک پردازه مقادیر متفاوتی را برگرداند. در واقع شما می‌توانید از pids به عنوان thread IDs استفاده کنید تا بتوانید با کمینه تغییرات، پیاده‌سازی خود را انجام دهید.

شما بایستی اطمینان حاصل کنید که در هنگام اجرای exit توسط یک ریسمان، پیاده‌سازی شما جدول صفحه (page table) این ریسمان را در صورت ***وجود*** دیگر ریسمان‌هایی که به این جدول صفحه اشاره می‌کنند، آزاد (free) ***نمی‌کند***. یک روش مناسب برای پیاده‌سازی این نیازمندی بسیار کلیدی، استفاده از مکانیزم شمارش ارجاعات یا reference counting است. با این حال شما می‌توانید از یک روش سریع دیگر نیز استفاده کنید. در روش دوم ابتدا جدول پردازه (process table) اسکن می‌شود تا تنها اگر ارجاع دیگری به جدول صفحه پیدا ***نشد***، جدول صفحه در نتیجه اجرای exit آزاد شود. طبیعی است که برای فهم کامل این بخش‌ها نیاز به مطالعه بیشتری خواهید داشت و تیم درس نیز آماده پاسخگویی به سوالات شما است.

وقتی که lock و unlock را پیاده‌سازی می‌کنید، فراخوانی‌های sleep و wakeup در proc.c می‌توانند به کمک شما بیایند.

موردی که بایستی خیلی به ان دقت کنید گسترش فضای ادرس توسط یک ریسمان متعلق به یک پردازه چند-ریسمانی است. کد مرتبط با این بخش را به دقت مطالعه و دنبال کنید و ببنید که در کجاها به قفل (lock) نیاز است و چه موارد دیگری بایستی که بروز شوند تا گسترش فضای ادرس برای یک پردازه چند-ریسمانی به درستی انجام شود.

تکمیل تابع clone به صورت افزایشی دشوار است. دو مورد مشکل که شما بایستی آن را به درستی انجام دهید عبارتند از: ۱) جلوگیری از حالت رقابت یا race condition در هنگام گسترش فضای آدرس و ۲) آزاد (free) نکردن جدول صفحه (page table) تا وقتی که تمامی ریسمان‌های متعلق به یک پردازه به اتمام برسند. توصیه ما این است که کار خود را از یک کد ساده (مانند لینک زیر) که این دو مشکل را با گسترش ندادن حافظه و exit نکردن دور می‌زند، شروع کنید.

http://pages.cs.wisc.edu/~cs537-3/Projects/threads.c

نحوه تحویل پروژه

- گزارش پروژه (شامل خروجی تمامی تست‌ها) وفایل‌های اضافه شده یا تغییر داده شده در xv6 را در قالب یک فایل zip با نام project_report_group_id_sid1_sid2.zip بارگذاری کنید.
- پروژه دارای تحویل به صورت انلاین است و توضیح کد و توضیح عملکرد سیستم عامل پرسیده می شود و ***بخش مهمی از نمره*** را در بر می گیرد. پس ضروری است که همه اعضای گروه به xv6 و پیاده‌سازی انجام شده تسلط داشته باشند.
- با توجه به انجام پروژه به شکل گروهی، تعداد commit های هر فرد باید متناسب باشد و خود گیت در حین تحویل چک می‌شود.
- لازم به ذکر است که تمامی پروژه‌ها پس از تحویل بررسی می‌شوند و هرگونه شباهت، به منزله نمره 0 برای تمام پروژه و تمرین‌ها خواهد بود.

موفق باشید

تیم تدریسیاری درس سیستم عامل