به نام خدا

آزمایشگاه سیستم‌عامل

پروژه‌ی چهارم: هم‌گام‌سازی

طراحان:یاسمن جعفری-امیرحسین احمدی

مقدمه

در این پروژه با سازوکارهای هم‌گام‌سازی[1] سیستم‌عامل‌ها آشنا خواهید شد. با توجه به این که سیستم‌عامل xv6 از ریسه‌های[2] سطح کاربر پشتیبانی نمی‌کند هم‌گام‌سازی در سطح پردازه‌ها مطرح خواهد بود. هم‌چنین به علت عدم پشتیبانی از حافظه مشترک در این سیستم‌عامل، هم‌گام‌سازی در سطح هسته صورت خواهد گرفت. به همین سبب مختصری راجع به این قسم از هم‌گام‌سازی توضیح داده خواهد شد.

---

[1] Synchronization Mechanisms
[2] Threads

ضرورت هم‌گام‌سازی در هسته سیستم‌عامل‌ها

هسته سیستم‌عامل‌ها دارای مسیرهای کنترلی[۳] مختلفی می‌باشد. به طور کلی، دنباله دستورالعمل‌های اجرا شده توسط هسته جهت مدیریت فراخوانی سیستمی، وقفه یا استثنا این مسیرها را تشکیل می‌دهند. در این میان برخی از سیستم‌عامل‌ها دارای هسته با ورود مجدد[۴] می‌باشند. بدین معنی که مسیرهای کنترلی این هسته‌ها قابلیت اجرای همروند[۵] دارند. تمامی سیستم‌عامل‌های مدرن کنونی این قابلیت را دارند. مثلاً ممکن است برنامه سطح کاربر در میانه اجرای فراخوانی سیستمی در هسته باشد که وقفه‌ای رخ دهد. به این ترتیب در حین اجرای یک مسیر کنترلی در هسته (اجرای کد فراخوانی سیستمی)، مسیر کنترلی دیگری در هسته (اجرای کد مدیریت وقفه) شروع به اجرا نموده و به نوعی دوباره ورود به هسته صورت می‌پذیرد. وجود هم‌زمان چند مسیر کنترلی در هسته می‌تواند منجر به وجود شرایط مسابقه برای دسترسی به حالت مشترک هسته گردد. به این ترتیب، اجرای صحیح کد هسته مستلزم هم‌گام‌سازی مناسب است. در این هم‌گام‌سازی باید ماهیت‌های مختلف کدهای اجرایی هسته لحاظ گردد. به عنوان مثال از قفل‌گذاری، پلی را تصور کنید که دارای محدودیت وزنی بر روی خود می‌باشد. به طوری که در هر لحظه تنها یک خودرو می‌تواند از روی پل عبور کند و در غیر این صورت فرو می‌ریزد. قفل همانند یک نگهبان در ورودی پل مراقبت می‌کند که تنها زمانی به خودرو جدید اجازه ورود بدهد که هیچ خودرویی بر روی پل نباشد.

هر مسیر کنترلی هسته در یک متن خاص اجرا می‌گردد. اگر کد هسته به طور مستقیم یا غیرمستقیم توسط برنامه سطح کاربر اجرا گردد، در متن پردازه[۶] اجرا می‌گردد. در حالی که کدی که در نتیجه وقفه اجرا می‌گردد در متن وقفه[۷] است. به این ترتیب فراخوانی سیستمی و استثناها در متن پردازه

---

[3] Control Paths
[4] Reentrant Kernel
[5] Concurrent
[6] Process Context
[7] Interrupt Context

فراخواننده هستند. در حالی که وقفه در متن وقفه اجرا می‌گردد. به طور کلی در سیستم‌عامل‌ها کدهای وقفه قابل مسدود شدن نیستند. ماهیت این کدهای اجرایی به این صورت است که باید در اسرع وقت اجرا شده و لذا قابل زمان‌بندی توسط زمان‌بند نیز نیستند. به این ترتیب سازوکار هم‌گام‌سازی آن‌ها نباید منجر به مسدود شدن آن‌ها گردد. مثلاً از قفل‌های چرخشی[8] استفاده گردد یا در پردازنده‌های تک‌هسته‌ای وقفه غیرفعال گردد.

هم‌گام‌سازی در xv6

قفل‌گذاری در هسته xv6 توسط دو سری تابع صورت می‌گیرد. دسته اول شامل توابع acquire() (خط ۱۵۷۳) و release() (خط ۱۶۰۱) می‌شود که یک پیاده‌سازی ساده از قفل‌های چرخشی هستند. این قفل‌ها منجر به انتظار مشغول[9] شده و در حین اجرای ناحیه بحرانی وقفه را نیز غیرفعال می‌کنند.

۱) علت غیرفعال کردن وقفه چیست؟ توابع pushcli() و popcli() به چه منظور استفاده شده و چه تفاوتی با cli و sti دارند؟

دسته دوم شامل توابع acquiresleep() (خط ۴۶۲۱) و releasesleep() (خط ۴۶۳۳) بوده که مشکل انتظار مشغول را حل نموده و امکان تعامل میان پردازه‌ها را نیز فراهم می‌کنند. تفاوت اصلی توابع این دسته نسبت به دسته قبل این است که در صورت عدم امکان در اختیار گرفتن قفل، از تلاش دست کشیده و پردازنده را رها می‌کنند.

۲) حالات مختلف پردازه‌ها در xv6 را توضیح دهید. تابع sched() چه وظیفه‌ای دارد؟

یک مشکل در توابع دسته دوم عدم وجود نگه‌دارنده[10] قفل است. به این ترتیب حتی پردازه‌ای که قفل را در اختیار ندارد می‌تواند با فراخوانی تابع releasesleep() قفل را آزاد نماید.

---

[8] Spinlocks
[9] Busy Waiting
[10] Owner

۳) می‌توان با اعمال تغییری در توابع دسته دوم، امکان آزادسازی را تنها برای پردازه صاحب قفل مسیر نمود. قفل معادل در هسته لینوکس را به طور مختصر معرفی نمایید.

پیاده‌سازی سازوکارهای همگام‌سازی جدید

مانع

مانع‌ها[11] بسته به حوزه استفاده دارای انواع مختلفی بوده که در این بخش یکی از انواع آن‌ها پیاده‌سازی خواهد شد. هدف کلی مانع‌ها جلوگیری از اجرای خارج از ترتیب[12] است. یک دسته از مانع‌ها موسوم به مانع‌های بهینه‌سازی[13] [2,3] بوده که در سطح کامپایلر کاربرد دارند. به این ترتیب که وجود یک دستور مانع در یک نقطه از کد برنامه، اجازه انتقال دستورهای پیش از این نقطه به پس از آن و بالعکس را (حین بهینه‌سازی) به کامپایلر نمی‌دهد.

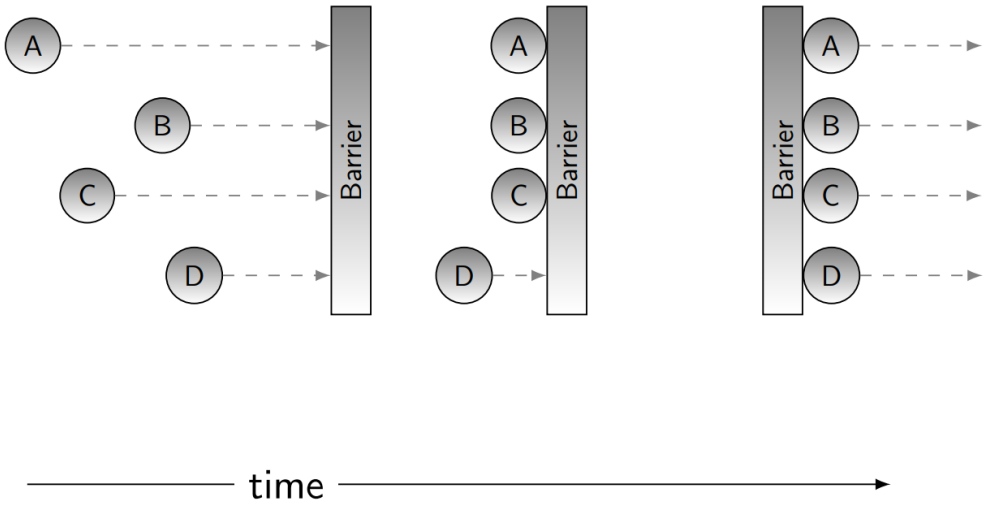
۴) پیاده‌سازی ماکروی barrier() در لینوکس برای معماری x86 را فقط بنویسید.

مانع بهینه‌سازی تنها از اجرای خارج از ترتیب در سطح کد می‌شود. جهت جلوگیری از تغییر ترتیب اجرا در سطح پردازنده از مانع‌های حافظه[14] [2,3] استفاده می‌شود. در این‌جا تمامی دستورالعمل‌های پیش از مانع حافظه باید پیش از هر دستورالعمل پس از آن اجرا شود.

۵) آیا یک دستور مانع حافظه باید مانع بهینه‌سازی هم باشد؟ نام ماکروی پیاده‌سازی سه نوع مانع حافظه در لینوکس در معماری x86 را به همراه دستورالعمل‌های ماشین پیاده‌سازی آن‌ها را ذکر کنید.

در نهایت نوع سومی از مانع‌ها وجو دارد که در حوزه پردازش موازی کاربرد داشته که در شکل زیر نشان داده شده است [1]:

11 Barriers
12 Out-of-Order Execution
13 Optimization Barriers
14 Memory Barriers



در این‌جا مانع برای یک دسته ریسه یا پردازه در سطح زبان برنامه‌سازی تعریف شده و هر ریسه/پردازه باید تا رسیدن تمامی ریسه‌ها/پردازه‌های دیگر در نقطه مانع، توقف کند.

۶) یک کاربرد از مانع در پردازش موازی ارائه دهید.

هدف از این پروژه، پیاده‌سازی همین نوع از مانع‌ها است. ابتدا نیاز است که یک برنامه سطح کاربر ایجاد شده و به تعداد مشخصی پردازه ایجاد گردد. هر یک از این پردازه‌ها باید عملیاتی زمان‌بر با زمان اجرای متفاوت انجام دهد (می‌توان از ایجاد تأخیر در کد استفاده کرد). حال می‌بایست پس از اتمام این عملیات، دستور مانع را قرار داده و صحت کد خود را بررسی نمایید. توجه کنید که کد شما باید شامل لاگ‌های کافی جهت ارائه صحت عملکرد کد باشد.

تمام قابلیت‌های مورد نیاز می‌بایست به صورت فراخوانی‌های سیستمی پیاده‌سازی شد. در پیاده‌سازی مانع، ممکن است نیاز به پیاده‌سازی سازوکارهایی همانند انتظار بر روی یک شرط خاص (چیزی شبیه متغیر وضعیت[۱۵]) و انتشار[۱۶] شوید.

[15] Condition Variable
[16] Broadcast

میوتکس با ورود مجدد

در این بخش از پروژه، پیاده‌سازی میوتکس با قابلیت ورود مجدد[۱۷] مدنظر است. همانطور که می‌دانید پس از در اختیار گرفتن میوتکس توسط یک پردازه، تا زمان آزادسازی این میوتکس توسط آن پردازه، امکان دریافت مجدد آن برای هیچ پردازه‌ای (اعم از خود پردازه مالک) وجود نخواهد داشت. اکنون حالتی را در نظر بگیرید که یک تابع به صورت بازگشتی خودش را صدا بزند و در بدنه‌ی این تابع بازگشتی، یک میوتکس را بگیرد. در این پروژه شما باید میوتکسی با قابلیت اخذ چندباره توسط پردازه مالک را پیاده‌سازی کنید.

سایر نکات

- تمیزی کد و مدیریت حافظه مناسب در پروژه از نکات مهم پیاده‌سازی است.
- از لاگ‌های مناسب در پیاده‌سازی استفاده نمایید تا تست و اشکال‌زدایی کد ساده‌تر شود. واضح است که استفاده بیش از حد از آن‌ها باعث سردرگمی خواهد شد.
- برای تحویل پروژه ابتدا یک مخزن خصوصی در سایت GitLab ایجاد نموده و سپس پروژه خود را در آن Push کنید. سپس اکانت UT_OS_TA را با دسترسی Maintainer به مخزن خود اضافه کنید. کافی است در محل بارگذاری در سایت درس، آدرس مخزن، شناسه آخرین Commit و گزارش پروژه را بارگذاری نمایید.
- پاسخ تمامی سؤالات را در کوتاه‌ترین اندازه ممکن در گزارش خود بیاورید.
- همه افراد باید به پروژه مسلط باشند و نمره تمامی اعضای گروه لزوماً یکسان نخواهد بود.
- در صورت تشخیص تقلب، نمره هر دو گروه صفر در نظر گرفته خواهد شد.
- فصل ۴ و انتهای فصل ۵ کتاب xv6 می‌تواند مفید باشد.

[17] Reentrant Mutex

- هر گونه سؤال در مورد پروژه را فقط از طریق فروم درس مطرح نمایید.

موفق باشید

مراجع

[1] Frédéric Haziza. 2009. Locks and Barriers. 30. Retrieved from https://www.it.uu.se/edu/course/homepage/os2/st09/handout-04.pdf

[2] Wolfgang Mauerer. 2008. *Professional Linux Kernel Architecture*. Wrox Press Ltd., Birmingham, UK, UK.

[3] Donald H. Pinkston. 2014. Caltech Operating Systems Slides.