به نام خدا

آزمایشگاه سیستم‌عامل

پروژه چهارم: هم‌گام‌سازی

طراحان: علی خوش‌طینت - علیرضا زارع‌نژاد

مقدمه

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

ضرورت همگام‌سازی در هسته سیستم عامل ها

هسته سیستم‌عامل‌ها دارای مسیرهای کنترلی[2] مختلفی می‌باشد. به طور کلی، دنباله دستورالعمل‌های اجرا شده توسط هسته جهت مدیریت فراخوانی سیستمی، وقفه یا استثنا این مسیرها را تشکیل می‌دهند.

[1] Synchronization Mechanisms
[2] Control Paths

در این میان برخی از سیستم‌عامل‌ها دارای هسته با ورود مجدد[۳] می‌باشند. بدین معنی که مسیرهای کنترلی این هسته‌ها قابلیت اجرای همروند[۴] 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.

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[۵] اجرا می‌گردد. در حالی که کدی که در نتیجه وقفه اجرا می‌گردد در متن وقفه[۶] است. به این ترتیب فراخوانی سیستمی و استثناها در متن پردازه 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[۷] استفاده گردد یا در پردازنده‌های تک‌هسته‌ای وقفه غیرفعال گردد.

---

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

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[8] شده و در حین اجرای ناحیه بحرانی وقفه را نیز غیرفعال می‌کنند.

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

۲) چرا قفل مذکور در سیستم‌های تک‌هسته‌ای مناسب نیست؟ روی کد توضیح دهید.

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.

۳) مختصری راجع به تعامل میان پردازه‌ها توسط دو تابع مذکور توضیح دهید. چرا در مثال تولیدکننده/مصرف کننده[9] استفاده از قفل‌های چرخشی ممکن نیست.

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

یک مشکل در توابع دسته دوم عدم می‌تواند عدم وجود مالک[10] قفل باشد.[11] به این ترتیب حتی پردازه‌ای که قفل را در اختیار ندارد می‌تواند با فراخوانی تابع releasesleep() قفل را آزاد نماید.

---

[8] Busy Waiting
[9] Producer Consumer
[10] Owner
[11] البته در کاربردهایی مانند ارسال سیگنال در سمافورها و متغیرهای شرط نیاز است همه پردازه‌ها قادر به ارسال سیگنال باشند. در حالی که در حفاظت از ناحیه بحرانی، وجود مالک یکتا اهمیت دارد.

۵) تغییری در توابع دسته دوم داده تا تنها پردازه صاحب قفل، قادر به آزادسازی آن باشد. قفل معادل در هسته لینوکس را به طور مختصر معرفی نمایید.

پیاده سازی قفل های جدید

قفل با ورود مجدد

در این بخش از پروژه، پیاده‌سازی میوتکس با قابلیت ورود مجدد مدنظر[۱۲] است. همانطور که می‌دانید پس از در اختیار گرفتن میوتکس توسط یک پردازه، تا زمان آزادسازی این میوتکس توسط آن پردازنده، 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 بدنه این تابع بازگشتی، یک میوتکس را بگیرد. در این پروژه شما باید میوتکسی را با قابلیت اخذ چندباره پردازه مالک پیاده‌سازی کنید.

قفل خوانندگان-نویسندگان

بخش اصلی پروژه مربوط به پیاده‌سازی قفل‌های خوانندگان-نویسندگان[۱۳] است. هدف از این پروژه شبیه‌سازی یک روش هم‌گام‌سازی در سطح هسته xv6 می‌باشد. قطعه کد زیر که از کتاب استخراج شده است ساختار یک پردازه نویسنده را نشان می‌دهد که در یک حلقه به طور متوالی در حال نوشتن است.

12 Reentrant Mutex
13 Readers-Writers Locks

```
while (true) {
   wait(rw_mutex);
      . . .
   /* writing is performed */
      . . .
   signal(rw_mutex);
}
```

قفل خوانندگان-نویسندگان را برای حالت تقدم خوانندگان و با قابلیت به‌خواب رفتن در حین انتظار پیاده‌سازی نمایید. بدین‌منظور دو فراخوانی سیستمی rwinit() و rwtest(uint pattern) را به سیستم‌عامل xv6 اضافه نمایید. تابع نخست، مقداردهی اولیه را انجام می‌دهد. در rwtest() باید الگوی دسترسی به داده مشترک به صورت یک پارامتر صحیح مثبت موسوم به pattern به هسته داده شود. به این ترتیب که سمت چپ‌ترین بیت همواره یک بوده و بیت‌های بعدی ترتیب زمانی خواندن یا نوشتن داده مشترک را مشخص کنند. صفر معادل خواندن و یک معادل نوشتن خواهد بود. مثلاً عدد ۱۸ در فرم دودویی به صورت زیر نوشته می‌شود:

10010

با صرفنظر کردن از پرارزش‌ترین بیت، به ترتیب خواندن، خواندن، نوشتن و خواندن رخ خواهد داد. به این ترتیب هر پردازه در سطح هسته، متناسب با دسترسی مورد نظر، قفل خواندن یا نوشتن را درخواست می‌نماید.

در این‌جا نیز می‌توان فرض نمود که متغیر مشترک یک عدد بوده که با هر بار نوشتن یک واحد به مقدار آن افزوده می‌شود. توابع مربوط به دریافت و رهاسازی هر نوع قفل نیز در هسته پیاده‌سازی می‌گردد. همانند حالت قفل بلیت نیاز به بررسی صحت اجرای کد می‌باشد. در همین راستا باید مواردی از قبیل

افزایش متناسب مقدار متغیر مشترک و امکان دسترسی هم‌زمان چندین خواننده به متغیر و تقدم خوانندگان بر نویسندگان نمایش داده شود. فرض بر این است که مشابه کتاب، تقدم با خوانندگان است.

امتیازی: پیاده سازی تقدم با نویسندگان ۱۰ نمره اضافه خواهد داشت.

سایر نکات:

- تمیزی کد و مدیریت حافظه مناسب در پروژه از نکات مهم پیاده سازی است.
- از لاگ های مناسب در پیاده سازی استفاده نمایید تا تست و اشکال زدایی کد ساده تر 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 سوالات را در کوتاهترین اندازه ممکن در گزارش خود بیاورید.
- همه افراد باید به پروژه مسلط باشند و نمره تمامی اعضای گروه لزوما 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.
- هرگونه سوال در مورد پروژه را از طریق ایمیل های طراحان می توانید بپرسید.

azarenejad99@gmail.com

ali.khoshtinattt@gmail.com