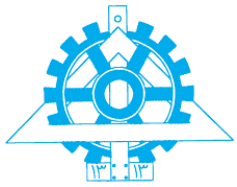

به نام خدا

آزمایشگاه سیستم‌عامل

پروژه‌ی چهارم: هم‌گام‌سازی

طراحان: امیرحسین احمدی - سعید زنگنه

مقدمه

در این پروژه با سازوکارهای هم‌گام‌سازی[۱] سیستم‌عامل‌ها آشنا خواهید شد. با توجه به این که سیستم‌عامل xv6 از ریسه‌های[۲] 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.

---

[۱] Synchronization Mechanisms

[۲] 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

---

[۳] Control Paths

[۴] Reentrant Kernel

[۵] Concurrent

[۶] Process Context

وقفه اجرا می‌گردد در متن وقفه[۷] 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[۸] 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[۹] شده و در حین اجرای ناحیه بحرانی[۱۰] 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.

---

[۷] Interrupt Context

[۸] Spinlocks

[۹] Busy Waiting

[10] Critical Section

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

یک مشکل در توابع دسته دوم عدم وجود نگه‌دارنده[۱۱] قفل است. به این ترتیب حتی پردازه‌ای که قفل را در اختیار ندارد می‌تواند با فراخوانی تابع releasesleep() قفل را آزاد نماید.

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

پیاده‌سازی سازوکار همگام‌سازی با قابلیت اولویت دادن

در این قسمت می‌خواهیم سازوکاری برای همگام‌سازی پیاده‌سازی کنیم که در آن پردازه‌ها می‌توانند برای ورود به ناحیه بحرانی دارای اولویت باشند. حال پردازه‌ها می‌توانند برای ورود به ناحیه بحرانی درخواست دهند و پردازه با اولویت بالاتر ابتدا وارد ناحیه بحرانی می‌شود. بعد از آن نیز پردازه‌ها که در یک صف دارای اولویت هستند، به ترتیب وارد می‌شوند. در زمانی که یک پردازه قفل را در اختیار دارد و در ناحیه بحرانی هست نیز ممکن است پردازه‌های جدیدی با اولویت‌های متفاوت وارد شوند که در این صورت ترتیب صف اولویت به صورت پویا عوض می‌شود..

برای پیاده‌سازی فرض کنید که اولویت پردازه‌های متفاوت، شماره پردازه آن می‌باشد.

شما نیاز است که برنامه سطح کاربری بنویسید که درستی پیاده‌سازی را نشان دهد. برای این کار چند پردازه در سطح کاربر ایجاد کنید که هر پردازه به دنبال ورود به ناحیه بحرانی و دریافت قفل می‌باشد. حال در ناحیه بحرانی یک کار زمان‌بر انجام دهید تا پردازه‌های دیگر خود را به صف اضافه کنند. در

[۱۱] Owner

هر مرحله نمایش دهید که پردازه با چه اولویتی وارد ناحیه بحرانی شد. همچنین صف اولویت را نیز نمایش دهید.

آیا این پیاده‌سازی ممکن است که دچار گرسنگی شود؟ راه‌حلی برای برطرف کردن این مشکل ارائه دهید. روش ارائه شده توسط شما باید بتواند شرایطی را که قفل‌ها دارای اولویت یکسان می‌باشند را نیز پوشش دهد. (نیازی به پیاده‌سازی برای این قسمت نیست)

یک نوع پیاده‌سازی همگام‌سازی توسط قفل بلیت[۱۲] انجام می‌شود. آن را بررسی کنید و تفاوت‌های آن با روش همگام‌سازی بالا را بیان کنید.

پیاده‌سازی متغیرهای مختص هر هسته پردازنده

یکی از روش‌های افزایش کارایی در پردازنده‌ها استفاده از حافظه نهان[۱۳] است. حافظه‌های نهان در سطوح[۱۴] مختلف وجود داشته و می‌توانند محلی[۱۵] یا مشترک[۱۶] باشند. به عنوان مثال، معمولاً حافظه نهان سطح یک[۱۷] مختص هر هسته پردازنده بوده و لذا محلی است. بدین ترتیب هرگاه پردازه‌ای در یک متغیر حافظه بنویسد، مقادیر نگه‌داری شده برای این متغیر در حافظه‌های نهان سطح یک دیگر هسته‌های پردازنده را نامعتبر[۱۸] می‌نماید.

۴) سازوکاری جهت حل این مشکل در سطح سخت‌افزار وجود دارد. مختصراً توضیح دهید.

معتبرسازی مقادیر حافظه نهان محلی، سربار قابل‌توجهی داشته و می‌تواند کارایی سیستم را پایین بیاورد.

---

12 Ticket Lock
13 Cache
14 Levels
15 Local
16 Shared
17 L1 Cache
18 Invalid

۵) همان‌طور که پیش‌تر ذکر شد، یکی از روش‌های همگام‌سازی استفاده از قفل‌هایی موسوم به قفل بلیت است. این قفل‌ها را از منظر مشکل مذکور در بالا بررسی نمایید.

در بسیاری از کاربردها می‌توان با استفاده از متغیرهای مختص هر هسته، مشکل را حل نمود. به این ترتیب که به جز در موارد ضروری، دسترسی و به‌روزرسانی را در نسخه مختص هسته جاری از متغیر انجام می‌دهند. بدین ترتیب با کاهش تعداد معتبرسازی، سربار کاهش می‌یابد.

۶) چگونه می‌توان در لینوکس داده‌های مختص هر هسته را در زمان کامپایل تعریف نمود؟ (راهنمایی: به منبع [۱] مراجعه نمایید.)

با استفاده از این روش، یک فراخوانی سیستمی تعریف نمایید که تعداد فراخوانی‌های سیستمی اجرا شده در یک بار کاری را روی یک سیستم چهارهسته‌ای برمی‌گرداند.

باید به تعداد کافی پردازه ایجاد نمایید که در فایل‌هایی می‌نویسند. هم‌چنین جهت اطمینان از صحت عملکرد باید یک نسخه مشترک میان همه هسته‌ها تعریف شده و با مقدار برگشتی مقایسه گردد. دقت کنید همواره باید متغیر مربوط به هسته در حال اجرا، به‌روزرسانی گردد. (راهنمایی: روش پیاده‌سازی قفل چرخشی xv6 می‌تواند راه‌گشا باشد.)

نیاز به ترازبندی حافظه نهان[۱۹] نیست.

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

[19] Cache Alignment

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

موفق باشید

مراجع

[1] Robert Love. 2010. *Linux Kernel Development* (3rd ed.). Addison-Wesley Professional.