به نام خدا

آزمایشگاه سیستم عامل

پروژه چهارم: همگام سازی

طراحان: امیرحسین عباسکوهی، سجاد علی‌زاده

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 این سیستم عامل، همگام سازی در سطح هسته صورت خواهد گرفت. به همین سبب مختصری راجع به این قسم از همگام سازی توضیح داده خواهد شد.

[1] Synchronization Mechanisms

ضرورت همگام سازی در هسته سیستم عامل ها

هسته سیستم‌عامل‌ها دارای مسیرهای کنترلی[۲] 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[۳] می باشند. بدین معنی که مسیرهای کنترلی این هسته‌ها قابلیت اجرای همروند[۴] دارند. تمامی سیستم عامل‌های مدرن کنونی این قابلیت را دارند. مثلا ممکن است برنامه سطح کاربر در میانه اجرای فراخوانی سیستمی در هسته باشد که وقفه‌هایی 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 همگام سازی 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 می گردد 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

[2] Control Path
[3] Reentrant Kernel
[4] Concurrent
[5] Process Context
[6] Interrupt Context

همگام‌سازی آنها نباید منجر به مسدود شدن آنها گردد، مثلا از قفلهای چرخشی[7] استفاده گردد یا در پردازنده های تک هسته ای وقفه غیر فعال گردد.

همگام‌سازی در xv6

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

۳) حالات مختلف پردازه‌ها در xv6 را توضیح دهید. تابع sched() چه وظیفه ای 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.

۴) 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.

---

[7] Spin Locks
[8] Busy Waiting
[9] Producer Consumer
[10] Owner

۵) یکی از روش‌های افزایش کارایی در بارهای کاری چندریسه‌ای استفاده از حافظه تراکنشی[۱۱] بوده که در کتاب نیز به آن اشاره شده است. به عنوان مثال این فناوری در پردازنده‌های جدیدتر اینتل[۱۲] تحت عنوان افزونه‌های همگام‌سازی تراکنشی[۱۳] (TSX) پشتیبانی می‌شود.[۱۴] آن را مختصراً شرح داده و نقش حذف قفل[۱۵] را در آن بیان کنید؟

شبیه‌سازی مسئله تولیدکننده-مصرف‌کننده

در این قسمت شما می‌بایست مسئله تولیدکننده-مصرف‌کننده[۱۶] را به وسیله سمافور[۱۷] حل کنید. در ابتدا نیاز است که کد سمافور را در سطح هسته پیاده‌سازی کنید. سمافور شمارشی[۱۸] یک نوع سازوکار همگام‌سازی می‌باشد که اجازه حضور تعدادی پردازه[۱۹] در هر لحظه در ناحیه بحرانی را داده و در صورتی که تعداد پردازه‌های درون ناحیه بحرانی آن به تعداد حداکثر برسد، پردازه‌های بعدی پشت سمافور منتظر می‌مانند.

در اینجا می‌خواهیم نوعی از سمافور را پیاده‌سازی کنیم که در صورتی که اجازه ورود به ناحیه بحرانی را نداشتند، به حالت خواب رفته و در یک صف قرار داده می‌شوند. سپس هنگامی که یکی از پردازه‌ها از ناحیه بحرانی خارج شد برای اینکه مشکل گرسنگی[۲۰] پیش نیاید، پردازه‌ها را به ترتیب زمان ورود از صف خارج می‌کنیم.

در ابتدا می‌بایست یک آرایه پنج تایی از سمافور در سطح سیستم ایجاد کنید که برنامه‌های سطح کاربر، از طریق فراخوانی‌های سیستمی زیر می‌توانند به آن‌ها دسترسی داشته باشند:

---

11 Transcational Memory
12 Intel
13 Transactional Synchronization Extensions
۱۴ به علت وجود اشکال‌های امنیتی، در اکثر ریزمعماری‌های کنونی، غیرفعال شده است. اما ظاهراً در آینده همچنان پشتیبانی خواهد شد.
15 Lock Elision
16 Producer-Consumer
17 Semaphore
18 Counting Semaphore
19 Process
20 Starvation

sem_init(i, v): سمافور در خانه i ام آرایه را با تعداد v برای حداکثر پردازه‌های درون ناحیه بحرانی ایجاد می‌کند.

sem_acquire(i): زمانی که یک پردازه بخواهد وارد ناحیه بحرانی شود، این فراخوانی سیستمی را صدا می‌زند.

sem_release(i): زمانی که یک پردازه بخواهد از ناحیه بحرانی خارج شود، این فراخوانی سیستمی را صدا می‌زند.

حال می‌خواهیم به وسیله فراخوانی‌های سیستمی سمافور پیاده‌سازی شده در قسمت قبل، مسئله تولیدکننده-مصرف‌کننده را پیاده‌سازی کنیم. در اینجا اندازه بافر را پنج در نظر بگیرید. می‌بایست کد سطح کاربر تولیدکننده و مصرف‌کننده و برنامه آزمونی بنویسید که صحت کد شما را با لاگ‌های مناسب نشان دهد. توجه کنید برای حل این مسئله باید از سه سمافور (مانند بخش 7.1.1 کتاب) استفاده کنید.

قفل با ورود مجدد

در این بخش از پروژه، پیاده‌سازی میوتکس[۲۱] با قابلیت ورود مجدد[۲۲] مد نظر 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 بدنه این تابع بازگشتی، یک میوتکس را در اختیار بگیرد. در این قسمت شما باید میوتکسی با قابلیت اخذ چندباره پردازه مالک پیاده‌سازی کنید. همچنین یک برنامه آزمون بنویسید تا صحت اجرای کد شما را نشان دهد.

[21] Mutex
[22] Reentrant Mutex

سایر نکات:

- تمیزی کد و مدیریت حافظه مناسب در پروژه از نکات مهم پیاده‌سازی است.
- از لاگ‌های مناسب در پیاده سازی استفاده نمایید تا تست و اشکال‌زدایی کد ساده‌تر شود. واضح است که استفاده بیش از حد از آنها باعث سردرگمی خواهد شد.
- فقط فایل‌های تغییر یافته و یا افزوده شده را به صورت ZIP 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 ایمیل‌های طراحان می‌توانید مطرح نمایید.

amirhossein.abaskohi@gmail.com
sajjadalizadeh2000@gmail.com

موفق باشید