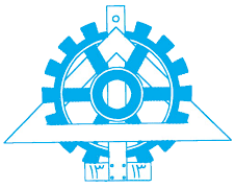

به نام خدا

آزمایشگاه سیستم‌عامل

پروژه‌ی سوم: هم‌گام‌سازی

طراحان: سحر رجبی-غزل صاحب‌جمع

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.

ضرورت هم‌گام‌سازی در هسته سیستم‌عامل‌ها

[1] Synchronization Mechanisms
[2] Threads

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.

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 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

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

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) مختصری راجع به تعامل میان پردازه‌ها توسط دو تابع مذکور توضیح دهید. چرا در مثال تولیدکننده/مصرف‌کننده[10] استفاده از قفل‌های چرخشی ممکن نیست؟

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

یک مشکل در توابع دسته دوم عدم وجود نگه‌دارنده[11] 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 Producer-Consumer
11 Owner

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.

پیاده‌سازی قفل‌های جدید

قفل بلیت

۵) یکی از انواع قفل‌ها قفل بلیت[۱۲] نام دارد. مزیت اصلی این قفل چیست؟ این قفل را با قابلیت به‌خواب رفتن پیاده‌سازی کنید. (راهنمایی: می‌توانید با تغییری در تابع sleep() (خط ۲۸۷۳) این امکان را فراهم آورید.)

دو فراخوانی سیستمی ticketlockinit() و ticketlocktest() را به سیستم‌عامل اضافه کنید. فراخوانی سیستمی نخست، قفل را مقداردهی اولیه نموده و دومی توسط پردازه‌هایی به طور موازی از آن استفاده می‌نماید تا از صحت عملکرد آن اطمینان حاصل شود. برنامه سطح کاربر باید چیزی مشابه برنامه زیر باشد.

12 Ticket Lock

```c
#include "types.h"
#include "user.h"

#define NCHILD 10

int main()
{
   int pid;

   ticketlockinit();

   pid = fork();
   for (int i = 1; i < NCHILD; i++)
      if (pid > 0)
         pid = fork();

   if (pid < 0)
   {
      printf(2, "fork error\n");
   }
   else if (pid == 0)
   {
      printf(1, "child adding to shared counter\n");
      ticketlocktest();
   }
   else
   {
      for (int i = 0; i < NCHILD; i++)
         wait();
      printf(1, "user program finished\n");
   }

   exit();
}
```

در پیاده‌سازی قفل باید از Inline Assembly استفاده نمایید. کد مربوطه باید در فایل x86.h اضافه شود.

۶) در مورد کد اسمبلی اضافه شده و اجزای آن توضیح دهید. چرا منجر به حالت مسابقه نمی‌شود؟

داده مشترک و توابع دریافت و رهاسازی قفل در هسته پیاده‌سازی خواهند شد. داده مشترک یک عدد خواهد بود که با هر دسترسی مقدارش یک واحد افزایش می‌یابد.

جهت اطمینان از صحت کد خود می‌توانید از ایجاد تأخیرهایی در مقاطع مختلف اجرای کد استفاده نمایید. (راهنمایی: می‌توانید از فراخوانی سیستمی sys_uptime() الگوبرداری نمایید.)

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.

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

۷) 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. ساختار توابع، مشابه توابع پیاده‌سازی شده برای قفل بلیت است. با این تفاوت که 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.

13 Readers-Writers Locks

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: استفاده از قفل بلیت در پیاده‌سازی، ۵ نمره و پیاده‌سازی تقدم با نویسندگان نیز ۵ نمره اضافی خواهد داشت.

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.
- فقط فایل‌های تغییریافته و یا افزوده شده را به صورت zip 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 :)