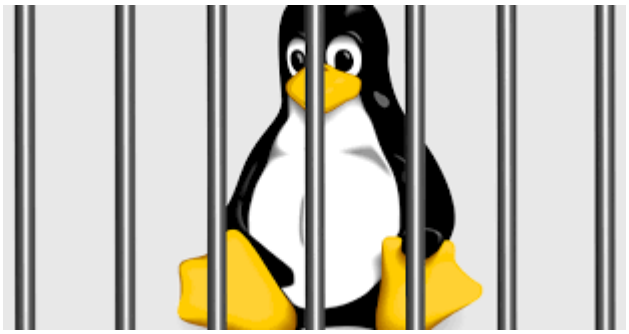
به نام خدا

آزمایشگاه سیستم‌عامل

پروژه چهارم: هم‌گام‌سازی

طراحان: امیرحسین احمدی – حسین سلطانلو



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.

---

[3] Control Paths
[4] Reentrant Kernel
[5] Concurrent

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 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 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

---

[6] Process Context
[7] Interrupt Context
[8] Spinlocks
[9] Busy Waiting

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 می‌تواند عدم وجود مالک[۱۰] قفل باشد.[11] 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 مسئله تولیدکننده-مصرف‌کننده به وسیله سمافور

در این قسمت شما می‌بایست مسئله تولیدکننده-مصرف‌کننده[۱۲] را به وسیله سمافور[۱۳] حل کنید. در ابتدا نیاز است که کد سمافور را در سطح هسته پیاده‌سازی کنید. سمافور شمارشی[۱۴] یک نوع سازوکار همگام‌سازی می‌باشد که اجازه حضور تعدادی پردازه 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

---

[10] Owner

[۱۱] 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.

[12] Producer-Consumer

[13] Semaphore

[14] Counting Semaphore

از ناحیه بحرانی خارج شد برای اینکه مشکل گرسنگی[15] 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.

semaphore_initialize(i, v, m): سمافور در خانه iام ارایه را با تعداد حداکثر پردازه درون ناحیه بحرانی v ایجاد می‌کند و مقدار m را برابر تعداد اولیه پردازه‌های درون ناحیه بحرانی قرار می‌دهد.

semaphore_aquire(i): هنگامی که یک پردازه می‌خواهد وارد ناحیه بحرانی شود، این فراخوانی سیستمی را فرا می‌خواند.

semaphore_release(i): هنگامی که یک پردازه از ناحیه بحرانی خارج می‌شود، این سیستم‌کال را فرا می‌خواند.

امتیازی

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 سطح‌کاربر تولیدکننده و مصرف‌کننده و برنامه آزمونی را بنویسید که صحت کد شما را با لاگ‌های مناسب نشان دهد.

تعیین ترتیب اجرای دو پردازه با متغیر شرط و پیاده‌سازی مسئله خوانندگان-نویسندگان

وظیفه شما در این بخش این است که ابتدا مشکل تعیین ترتیب اجرای دو پردازه را با استفاده از متغیر

[15] starvation

شرط[۱۶] در فضای کاربر حل نمایید. سپس با استفاده از متغیر شرط و قفل چرخشی، مسئله خوانندگان-نویسندگان را پیاده‌سازی نمایید. بدین‌منظور نیاز است تا متغیر شرط پیاده‌سازی شود که در پیاده‌سازی ما به قفل چرخشی[۱۷] وابسته است. اگر کمی دقت کنید می‌بینید که در xv6 و در فایل spinlock.c پیاده‌سازی مکانیزم قفل چرخشی وجود دارد، اما این قفل مربوط به استفاده درون هسته است و علاوه بر اطلاعاتی که برای دیباگ در خود نگه‌داری می‌کند، پیچیدگی‌هایی مثل وقفه‌ها را نیز کنترل می‌کند. بدین منظور نیاز داریم تا نسخه ساده‌تری از آن را برای استفاده در فضای کاربر پیاده‌سازی کنیم. می‌توانید از ساختار spinlock [۱۸] که در فایل spinlock.h وجود دارد استفاده کرده و تنها متغیر locked از این ساختار را با استفاده از توابع مربوطه کنترل کنید.

۴) فرض کنید در متن پردازه A، قفل spinlock موجود در کد منبع xv6 فعال شده است. آیا امکان رهاسازی آن در متن پردازه دیگر B وجود دارد؟ با استدلال توضیح دهید.

برای درک بهتر نحوه کار این قفل پیشنهاد می‌شود به آشنایی با نحوه عملکرد تابع xchg بپردازید. این نسخه باید به شکل زیر در فضای کاربر قابل استفاده باشد:

```
struct spinlock lk;
init_lock(&lk);
lock(&lk);
// critical section
unlock(&lk);
```

که تابع init_lock کار مقداردهی اولیه آن را انجام داده و توابع lock و unlock به ترتیب وظیفه اخذ و آزادسازی قفل را بر عهده دارند.

در قدم بعد و پس از پیاده‌سازی قفل چرخشی، اقدام به پیاده‌سازی متغیر شرط می‌کنیم. از متغیر

16 Condition Variable
17 Spinlock
18 Struct

شرط برای تعیین ترتیب اجرا استفاده می‌شود. در این بخش دو فراخوانی سیستمی cv_wait() و cv_signal() را به مجموعه فراخوانی‌های سیستمی اضافه می‌کنیم که به ترتیب اولی به منظور خواباندن پردازه‌ها و دومی به منظور بیدارکردن پردازه‌هایی استفاده می‌شود که روی یک متغیر شرط خوابیده‌اند. به‌خاطر داشته باشید که هردوی این فراخوانی‌ها زمانی صورت می‌گیرد که یک قفل توسط پردازه اخذ شده باشد. به همین دلیل است که در ابتدای کار، قفل چرخشی را پیاده‌سازی کردیم. از همین رو متغیر شرط ما یک ساختار با نام condvar خواهد بود که خود این ساختار شامل یک متغیر از جنس ساختار قفل چرخشی است. به عنوان مثال ممکن است برنامه به ترتیب زیر از متغیر شرط استفاده می‌کند:

```
lock (&condvar.lock);
cv_wait(&condvar);
unlock(&condvar.lock);
```

پیاده‌سازی ما از متغیر شرط وابستگی زیادی به پیاده‌سازی مکانیزم‌های sleep و wakeup دارد که می‌توانید آن‌ها را در فایل proc.c مشاهده کنید. در نتیجه ما نسخه‌ای سبک‌تر از تابع sleep را پیاده‌سازی می‌کنم که از قفل‌های چرخشی که در ابتدا ایجاد کردیم استفاده می‌کند و بخش‌های اضافی آن حذف شده است. نام این تابع را sleep1 بگذارید. نهایتاً نیاز است تا به پیاده‌سازی فراخوانی‌های سیستمی cv_wait() و cv_signal() بپردازیم که هر دو یک متغیر شرط دریافت کرده و سپس توابع sleep1() و wakeup() را فراخوانی می‌کنند. تعریف این توابع را با رابط زیر انجام دهید:

```
cv_wait(struct condvar *);
cv_signal(struct condvar *);
```

۵) به نظر شما چرا در این‌جا مانند قفل چرخشی به تعریف توابع در سطح کاربر بسنده نکرده و فراخوانی‌های سیستمی جدیدی را تعریف کردیم؟

در نهایت می‌خواهیم در قطعه کد زیر، در سطح کاربر ترتیب اجرای دو پردازه را کنترل کنیم، به ترتیبی که همواره پردازه دو پس از اتمام کار پردازه یک اجرا شود. یعنی Child 1 Executing تحت هر شرایطی پیش از Child 2 Executing چاپ شود. این کار را با استفاده از توابعی که تعریف کردیم و با استفاده از متغیر شرط انجام دهید.

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

int main()
{
  int pid = fork();
  if (pid < 0)
  {
    printf(1, "Error forking first child.\n");
  }
  else if (pid == 0)
  {
    printf(1, "Child 1 Executing\n");
  }
  else
  {
    pid = fork();
    if (pid < 0)
    {
      printf(1, "Error forking second child.\n");
    }
    else if (pid == 0)
    {
      printf(1, "Child 2 Executing\n");
    }
    else
    {
      int i;
      for (i = 0; i < 2; i++)
        wait();
    }
  }
  exit();
}
```

۶) به نظرتان امکان بن‌بست در این مسئله وجود دارد؟ برای جلوگیری از این اتفاق در مثال بالا می‌توانید در اولین خطی که از پردازه اول اجرا می‌شود sleep با مقدار زمان دلخواه قرار دهید. برای پیاده‌سازی و جلوگیری از این مشکل به روش اصولی چه راه‌حل‌هایی پیشنهاد می‌کنید؟ فقط در گزارش

خود توضیح دهید.

در بخش نهایی نیز وظیفه شما این است که مسئله خوانندگان-نویسندگان را با استفاده از متغیر شرط و قفل چرخشی، با تقدم خوانندگان پیاده‌سازی نمایید. در این شرایط مختار هستید هر نوع پیاده‌سازی مربوط به قفل چرخشی یا متغیر شرط را به درون هسته منتقل کنید تا در فراخوانی سیستمی مربوط به مسئله خوانندگان-نویسندگان بتوانید از آن‌ها بهره ببرید، اما بین توابع مربوط به این بخش با بخش قبلی، یعنی تعیین ترتیب اجرای دو پردازه، تمایز قائل شوید تا همه موارد قابل تست و بررسی باشند. (کد مربوط به هر دو بخش را نگه‌داری کنید)

جهت آزمایش می‌توانید یک متغیر مشترک عددی در نظر بگیرید که با هر بار نوشتن روی آن، یک واحد به مقدار آن افزوده می‌شود. دقت داشته باشید که به میزان کافی و در بخش‌های مختلف لاگ قرار دهید تا درستی پیاده‌سازی‌هایتان قابل بررسی باشد. در همین راستا باید مواردی از قبیل افزایش متناسب مقدار متغیر مشترک، امکان دسترسی هم‌زمان چندین خواننده به متغیر و تقدم خوانندگان بر نویسندگان نمایش داده شود.

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.
- آدرس مخزن و شناسه آخرین تغییر خود را در محل بارگذاری در سایت درس، 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