به نام خدا

آزمایشگاه سیستم عامل

پروژه چهارم: همگام سازی

طراحان: پوریا تاج محرابی، حامد میرامیرخانی

مقدمه

در این پروژه با سازوکارهای همگام‌سازی[1] سیستم عامل‌ها آشنا خواهید شد. با توجه به این که سیستم عامل 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 سیستم عامل، 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 همگام سازی در هسته سیستم عامل ها

هسته سیستم عامل‌ها دارای مسیرهای کنترلی[3] 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 سیستم عامل‌ها دارای هسته با ورود مجدد[4] می باشند. بدین معنی که مسیرهای کنترلی این هسته‌ها قابلیت اجرای همروند[5] دارند. تمامی سیستم عامل های مدرن کنونی این قابلیت را دارند. مثلا ممکن است برنامه سطح کاربر در میانه اجرای فراخوانی سیستمی در هسته باشد که وقفه هایی 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 می پذیرد. 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 همگام سازی مناسب است. در این همگام سازی باید ماهیت های مختلف کدهای اجرایی هسته لحاظ گردد.به عنوان مثال از قفل گذاری، 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[6] اجرا می‌گردد. در حالی که کدی که در نتیجه وقفه اجرا می‌گردد در متن وقفه[7] 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 سیستم عامل‌ها کدهای وقفه قابل مسدود شدن نیستند. ماهیت این کدهای اجرایی به

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

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، مثلا از قفلهای چرخشی[8] 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 1573) و release (خط 1601) 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.

1) علت غیرفعال کردن وقفه در هنگام استفاده از این نوع قفل چیست؟ چرا ممکن است CPU با مشکل deadlock رو به رو شود؟

دسته دوم شامل توابع acquiresleep() (خط ۴۰۶۳) و releasesleep() (خط ۴۰۱۱) بوده که مشکل انتظار مشغول را حل نموده و امکان تعامل میان پردازه ها 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.

2) حالات مختلف پردازه ها در xv6 را توضیح دهید. تابع sched() چه وظیفه‌ای دارد؟

3) در مجموعه دستورات RISC-V، دستوری با نام amoswap وجود دارد. دلیل تعریف و نحوه کار آن را توضیح دهید.

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.

4) 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.

---

[8] Spin Locks
[9] Busy Waiting
[10] Owner

5) روشی دیگر برای نوشتن برنامه‌ها استفاده از الگوریتم های lock-free است. مختصری راجع به آن ها توضیح داده و از مزایا و معایب آن‌ها نسبت به برنامه نویسی با lock بگویید.

پیاده سازی سازوکارهای همگام سازی جدید

میوتکس با ورود مجدد

در این بخش از پروژه، پیاده سازی mutex با قابلیت ورود مجدد مدنظر است. همانطور که میدانید پس از در در اختیار گرفتن mutex توسط یک پردازه، تا زمان آزادسازی این mutex 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 بدنه‌ی این تابع بازگشتی، یک mutex را بگیرد. در این پروژه شما باید میوتکسی با قابلیت اخذ چندباره توسط پردازه مالک را پیاده‌سازی کنید.

برنامه تست میتوتکس

تست کردن mutex ای که ساخته اید نیازمند یک برنامه آزمایشی است. برای نشان دادن تفاوت بین پردازه های دیگر و پردازه ای که در هر مرحله قفل را در اختیار دارد، برنامه ای با مشخصات زیر بنویسید.

نکته: نیاز است این برنامه به شکل یک system call قابل دسترسی باشد زیرا نیازمند ذخیره سازی اطلاعاتی به صورت global و مشترک بین چند process است که استفاده از آن در فضای حافظه ای kernel ممکن و راحت تر است (در غیر این صورت آزمایش اجرای موازی mutex با قابلیت ورود مجدد نیازمند thread ها خواهد بود که از پیاده سازی آن ها صرف نظر شده).

برنامه شما نیاز به یک system call با نام sys_futex(n) دارد. یک user program به نام futex بنویسید که n را ورودی گرفته و پس از اجرا دو پردازه تولید کرده که با استفاده از این System Call عدد فیبوناچی n ام را محاسبه می کنند.

بدنه این System Call باید طوری طراحی و پیاده سازی شود که با استفاده از نوعی پیاده سازی بازگشتی فیبوناچی

به صورت هم روند اجازه محاسبه عدد n ام دنباله را دهد تا حالت های مختلف استفاده از mutex قابل نمایش باشد.

راهنمایی می توانید از کد ها و توضیحات زیر به عنوان راهنمایی بهره جویید

```
// reentrant_mutex.h
#ifndef REENTRANT_MUTEX_H
#define REENTRANT_MUTEX_H

#include "types.h"
#include "spinlock.h"

struct reentrant_mutex {
  struct spinlock lk; // Spinlock for atomic operations
  struct proc *owner; // Owner of the mutex
  int owner_pid;
  int count;          // Count of recursive acquisitions
};

void reentrant_mutex_init(struct reentrant_mutex *rm);
void reentrant_mutex_lock(struct reentrant_mutex *rm);
void reentrant_mutex_unlock(struct reentrant_mutex *rm);

#endif // REENTRANT_MUTEX_H
```

```
struct reentrant_mutex futex_mutex;
struct fib_helper
{
    int fib__0;
    int fib__1;
    int last_calculated_index;
};
```

- هر process در تکه های کد که critical section حساب می شوند قفل مد نظر را روی یک struct که حاوی اطلاعات run فیبوناچی (توسط پردازه قبلی) است می گیرد. سپس با این اطلاعات، عدد بعدی در سری را محاسبه کرده و اطلاعات struct را برای run بعدی فیبوناچی آپدیت می کند و سپس قفل را رها

کرده و recur میکند.

انتظار داریم که در این برنامه، بعضی مواقع پردازه اول عدد بعدی را پردازش و قرار دهد و در بعضی مواقع پردازه دوم و هیچ کدام در حالتی که run قبلی فیبوناچی توسط iteration قبلی خود حساب شده است توسط تابع اخذ قفل mutex بلاک نشوند.

- برای اینکه نشان دهید mutex قفل شده فقط وقتی یک پردازه را هنگام اخذ قفل بلاک می کند که پردازه ای غیر از پردازه مالک mutex باشد، نیاز به قرار دادن log های متفاوت در روند برنامه خود هستید. توجه کنید که بدون این log های خروجی اطمینان از صحت عملکرد برنامه شما در هنگام تحویل ممکن نخواهد بود. برای تولید این log ها از **cprintf** در داخل kernel استفاده کنید.
- برای تولید و خلق زمان بندی مناسب بین دو پردازه ممکن است نیاز باشد در بعضی شرایط پردازه ای را مجبور به صبر کردن کنید تا حالات مد نظر کار کردن با قفل را ایجاد کنید. تابع sleep زیر در این شرایط به شما کمک خواهد کرد.این تابع به صورت پیش فرض در xv6 وجود ندارد و می توانید از تابع زیر در کد خود استفاده کنید.

```
#include "types.h"
#include "x86.h"
#include "defs.h"
#include "date.h"
#include "param.h"
#include "memlayout.h"
#include "mmu.h"        You, 10 minutes ago • A
#include "proc.h"
```

```
void sleep_seconds(int seconds)
{
    uint ticks0;

    acquire(&tickslock);
    ticks0 = ticks;
    while (ticks - ticks0 < seconds * 100)
    {
        sleep(&ticks, &tickslock);
    }
    release(&tickslock);
}
```

سایر نکات:

- 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.
- کدهای خود را مشابه پروژه‌های پیشین در Github یا Gitlab بارگذاری نموده و آدرس مخزن، شناسه آخرین Commit و گزارش پروژه را در سایت بارگذاری نمایید.
- 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.
- هرگونه سوال در مورد پروژه را از طریق ایمیل‌های طراحان می‌توانید مطرح نمایید.

pouriyatajmehrabi1381@gmail.com

hamedmiramirkhani@gmail.com

موفق باشید