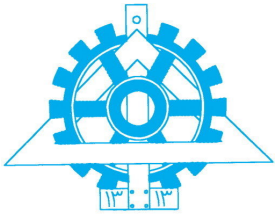
پروژه آزمایشگاه شماره ۴



دانشکده مهندسی برق و کامپیوتر

سیستم عامل - پاییز ۱۳۹۹

استاد : **دکتر کارگهی**

گروه ۱۷

اعضای گروه:

دانشور امراللهی، علیرضا توکلی، امین ستایش

---

۱. علت غیرفعال کردن وقفه چیست؟ توضیح دهید که توابع pushcli و popcli به چه منظور استفاده می‌شوند و چه تفاوتی با cli و sti دارند؟

دلیل این است که می‌خواهیم مطمئن شویم که کدهایی که می‌خواهیم اجرا کنیم به صورت atomic اجرا شوند. یعنی کدهای وقفه را نمی‌توان مسدود کرد و برای محافظت از ناحیه critical باید وقفه‌ها غیرفعال شده باشند.

این عملیات به کمک دو تابع pushcli و popcli انجام می‌شود به این صورت که با استفاده از pushcli وقفه‌ها را غیر فعال می‌کنیم و از acquire استفاده می‌کنیم و سپس پس از اتمام ناحیه critical و release تابع popcli صدا می‌شود تا وقفه‌ها دوباره فعال شوند.

خود pushcli و popcli از sti و cli استفاده می‌کنند اما نکته قابل اهمیت این است که فقط تابع‌هایی بر روی این‌ها نیستند و قابلیت‌های دیگری هم دارند. فرق این است که pushcli و popcli قابلیت شمارش هم دارند یعنی مشخص است که هر کدام چقدر اجرا شده‌اند که می‌تواند در کنترل کردن کمک کند.

۲. چرا قفل مذکور در سیستم‌های تک‌هسته‌ای مناسب نیست؟ روی کد توضیح دهید.

کد acquire را بررسی می‌کنیم.

```
vod
        acquire(struct spinlock *lk)
        {
         pushcli(); // disable interrupts to avoid deadlock.
         if(holding(lk))
           panic("acquire");


         // The xchg is atomic.
         while(xchg(&lk->locked, 1) != 0)
           ;


         // Tell the C compiler and the processor to not move loads or stores
         // past this point, to ensure that the critical section's memory
         // references happen after the lock is acquired.
         __sync_synchronize();


         // Record info about lock acquisition for debugging.
         lk->cpu = mycpu();
         getcallerpcs(&lk, lk->pcs);
        }
```

طبق کد بالا که کد acquire است می‌بینیم که با توجه به شرط holding، این قفل‌ها در xv6 به صورت busy waiting پیاده‌سازی شده‌اند و مشکلی که در پردازه‌های تک هسته‌ای پیش می‌اید این است که اگر یک پردازه به مدت طولانی قفل را در اختیار بگیرد بقیه چون busy waiting هستند مشکل ایجاد می‌شود.

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

```
void
acquiresleep(struct sleeplock *lk)
{
 acquire(&lk->lk);
 while (lk->locked) {
   sleep(lk, &lk->lk);
 }
 lk->locked = 1;
 lk->pid = myproc()->pid;
 release(&lk->lk);
}

void
releasesleep(struct sleeplock *lk)
{
 acquire(&lk->lk);
 lk->locked = 0;
 lk->pid = 0;
 wakeup(lk);
 release(&lk->lk);
}
```

در acquiresleep یک پردازنده روی آدرس قفلی که به آن پاس داده شده، sleep می‌کند تا موقعی که فرصتی برای در درست گرفتن قفل مورد نظر پیدا نکند. در releasesleep، ریسه‌ای که sleeplock را نگه داشته، تمام پردازه‌هایی که روی چنل قفل sleep کرده‌اند را بیدار می‌کند و وضعیت آن پردازه‌ها از SLEEPING به RUNNABLE تغییر می‌کند.

در مسئله producer/consumer اگر فقط از spinlock بخواهیم استفاده کنیم (از semaphore استفاده نکنیم)، نمی‌توانیم تضمین کنیم که بلافاصله بعد از آزاد شدن قفل توسط مصرف‌کننده، قفل به مصرف‌کننده برسد. یعنی ممکن است شرایطی پیش بیاید که buffer خالی است اما تولیدکننده قفل را برای مدتی در درست نمی‌گیرد.

۴. حالات مختلف پردازه‌ها در xv6 را توضیح دهید. تابع sched چه وظیفه‌ای دارد؟

حالت‌های زیر را می‌توانند داشته باشند که هر کدام به اختصار توضیح داده شده است.

- UNUSED: از پردازه استفاده‌ای نشده است.
- EMBRYO: وقتی که از حالت UNUSED تغییر می‌کنیم به این حالت می‌رویم و پردازه دیگر unused نیست.
- SLEEPING: پردازه در cpu نیست و به منبعی نیاز دارد که هنوز آماده نیست. یعنی scheduler از آن استفاده نمی‌کند.
- RUNNABLE: این پردازه می‌تواند توسط scheduler به cpu اختصاص پیدا کند.
- RUNNING: پردازه‌ای که cpu به آن اختصاص داده شده است و در حال اجرا است.
- ZOMBIE: پردازه‌ای که کارش تمام شده اما پردازه پدر wait را صدا نکرده و اطلاعات این پردازه هنوز در ptable موجود است.

تابع sched در پایان کار یک پردازه صدا زده می‌شود و عملکرد آن به این صورت است که context فعلی ذخیره می‌شود و context با scheduler جایگزین می‌شود.

۵. 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.

در لینوکس mutex به این صورت است که در لینک زیر کد آن آمده است.

https://github.com/torvalds/linux/blob/master/include/linux/mutex.h

در این حالت تنها صاحب قفل آن را آزاد می‌کند و busy waiting هم در نتیجه نداریم.

تغییری که داده شده و در mutex هم قابل مشاهده است این است که فقط صاحب پردازه می‌تواند عملیات را انجام دهد و دیگر مشکل موردنظر را نداریم.

یعنی باید شرط اینکه چه پردازه‌ای در حال صدا کردن تابع است بررسی شود.

در کد mutex در لینوکس میبینیم که یک owner تعریف شده است که با استفاده از آن این موضوع چک می‌شود.

مثال اجرای مسئله reader/writer با اولویت reader:

```
$ rwtest 180 0
10110100
Reader PID 21: behind while
Reader PID 21: passed while
Reader PID 21: reading started
Reader PID 21: reading done
Reader PID 21: exiting
Writer PID 22: behind while
Writer PID 22: passed while
Writer PID 22: writing started
Writer PID 22: writing done
Writer PID 23: behind while
Writer PID 22: exiting
Writer PID 23: passed while
Writer PID 23: writing started
Writer PID 23: writing done
Writer PID 23: exiting
Reader PID 24: behind while
Reader PID 24: passed while
Reader PID 24: reading started
Reader PID 24: reading done
Reader PID 24: exiting
Writer PID 25: behind while
Writer PID 25: passed while
Reader PID 26: behind while
Writer PID 25: writing started
Writer PID 25: writing done
Writer PID 25: exiting
Reader PID 26: passed while
Reader PID 27: behind while
Reader PID 27: passed while
Reader PID 26: reading started
Reader PID 27: reading started
Reader PID 26: reading done
Reader PID 27: reading done
Reader PID 26: exiting
Reader PID 27: exiting
```

خواننده‌های ۲۶ و ۲۷ همزمان در فضای فرضی مشغول خواندن بوده‌اند. از ورود خواننده ۲۶ جلوگیری شده تا کار نویسنده قبلی تمام شود.