



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

می شوند و چه تفاوتی با `cli` و `sti` دارند؟

دلیل این است که می خواهیم مطمئن شویم که کدهایی که می خواهیم اجرا کنیم به صورت `atomic` اجرا شوند. یعنی کدهای وقفه را نمی توان مسدود کرد و برای محافظت از ناحیه `critical` باید وقفه ها غیرفعال شده باشند. این عملیات به کمک دو تابع `pushcli` و `popcli` انجام می شود به این صورت که با استفاده از `pushcli` وقفه ها را غیر فعال می کنیم و از `acquire` استفاده می کنیم و سپس پس از اتمام ناحیه `critical` و `release` تابع `popcli` صدا می شود تا وقفه ها دوباره فعال شوند.

خود `pushcli` و `popcli` از `cli` و `sti` استفاده می کنند اما نکته قابل اهمیت این است که فقط تابع هایی بر روی این ها نیستند و قابلیت های دیگری هم دارند. فرق این است که `pushcli` و `popcli` قابلیت شمارش هم دارند یعنی مشخص است که هر کدام چقدر اجرا شده اند که می تواند 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 پردازش‌های تک‌هسته‌ای پیش می‌آید این است که اگر یک پردازش به مدت طولانی قفل را در اختیار بگیرد بقیه چون busy waiting هستند مشکل ایجاد می‌شود.

۳. مختصری راجع به تعامل میان پردازنده‌ها 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`، ریشه‌ای که `sleeplock` را نگه داشته، تمام پردازنده‌هایی که روی چنل قفل `sleep` کرده‌اند را بیدار می‌کند و وضعیت آن پردازنده‌ها از `SLEEPING` به `RUNNABLE` تغییر می‌کند.

در مسئله producer/consumer اگر فقط از spinlock بخوایم 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 دست نمی‌گیرد.

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

تابع sched در پایان کار یک پردازنده 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 پردازنده 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 پردازش می‌تواند عملیات را انجام دهد و دیگر مشکل موردنظر را نداریم.

یعنی باید شرط اینکه چه پردازشی 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:

```
Group #17:
1- Amin Setayesh
2- Daneshvar Amrollahi
3- Alireza Tavakoli
$ rwtest 58 0
111010
Writer PID 4: behind while
Writer PID 4: passed while
Writer PID 4: writing started
Writer PID 5: behind while
Reader PID 6: behind while
Writer PID 7: behind while
Reader PID 8: behind while
Writer PID 4: writing done
Writer PID 4: exiting
Reader PID 6: passed while
Reader PID 6: reading started
Reader PID 8: passed while
Reader PID 8: reading started
Reader PID 6: reading done
Reader PID 8: reading done
Reader PID 6: exiting
Reader PID 8: exiting
Writer PID 5: passed while
Writer PID 5: writing started
Writer PID 5: writing done
Writer PID 5: exiting
Writer PID 7: passed while
Writer PID 7: writing started
Writer PID 7: writing done
Writer PID 7: exiting
```

نویسنده شماره ۵، خواننده شماره ۶، نویسنده شماره ۷ و خواننده شماره ۸ پشت while منتظر هستند تا کار نویسنده شماره ۴ تمام شود. با خروج نویسنده شماره ۴، بین ۴ تا پردازش‌ای که پشت حلقه هستند، اولویت با خواننده‌ها است که به درستی خواننده شماره ۶ برای ورود انتخاب شده است. پس از آن خواننده شماره ۸ وارد شده است. پس از اتمام کار خواننده‌ها، نویسنده‌ها به نوبت وارد می‌شوند (در هر لحظه حداکثر یک نویسنده داخل فضای فرضی است)

مثال اجرای مسئله reader/writer با اولویت writer:

```
Group #17:
1- Amin Setayesh
2- Daneshvar Amrollahi
3- Alireza Tavakoli
$ rwtest 69 1
1000101
Reader PID 4: behind while
Reader PID 4: passed while
Reader PID 4: reading started
Reader PID 5: behind while
Reader PID 5: passed while
Reader PID 5: reading started
Reader PID 6: behind while
Reader PID 6: passed while
Reader PID 6: reading started
Writer PID 7: behind while
Reader PID 8: behind while
Writer PID 9: behind while
Reader PID 4: reading done
Reader PID 4: exiting
Reader PID 5: reading done
Reader PID 5: exiting
Reader PID 6: reading done
Reader PID 6: exiting
Writer PID 7: passed while
Writer PID 7: writing started
Writer PID 7: writing done
Writer PID 7: exiting
Writer PID 9: passed while
Writer PID 9: writing started
Writer PID 9: writing done
Writer PID 9: exiting
Reader PID 8: passed while
Reader PID 8: reading started
Reader PID 8: reading done
Reader PID 8: exiting
```

در ابتدا خواننده‌های ۴ و ۵ و ۶ وارد می‌شوند. سپس نویسنده ۷ می‌خواهد وارد شود اما باید صبر کند کار خواننده‌های قبلی تمام شود. همچنین خواننده شماره ۸ هم می‌خواهد وارد شود اما چون اولویت با نویسنده شماره ۷ است که هنوز داخل نرفته، دیگر وارد نمی‌شود و باید صبر کند. با خروج ۳ خواننده اول، نویسنده شماره ۷ وارد می‌شود. سپس نویسنده شماره ۹ وارد می‌شود (نسبت به خواننده شماره ۸ که منتظر است اولویت دارد). نهایتاً خواننده شماره ۸ وارد می‌شود.