OS LAB 4
Process Synchronisation and CPU coherency

Alireza Hosseini 810100125
AmirAli Shahriary 810100173
SMahdi HajiSeyedHossein 810100118

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

دلیل آن اطمنیان از این است 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 وقفه ها غیرفعال شده باشند. از طرفی در سیستم عامل ها وقفه ها قابل مسدود کردن نیستند.
وقفه می تواند باعث ایجاد هم روندی حتی در یک پردازه شده و اگر وقفه ها فعال باشند ، کد هسته می تواند در هر مقطعی متوقف شود تا مدیریت وقفه به جای آن انجام شود که می تواند منجر به deadlock در کل سیستم شود.

2: توابع pushcli و popcli به چه منظوری استفاده شده اند و چه تفاوتی cli و sti دارند؟

عملیات سوال پیشین به کمک این دو تابع انجام شده ؛ به کمک تابع pushcli وقفه ها را غیر فعال میکنیم و از acquire استفاده میکنیم و سپس پس از اتمام ناحیه critical و release تابع popcli صدا میشود تا وقفه ها مجدد فعال شوند. خود pushcli و popcli از sti و cli استفاده میکنند اما نکته قابل اهمیت این است که فقط تابع هایی بر روی اینها نیستند و قابلیتهای دیگری هم دارند. از توابع cli و sti به ترتیب برای دستور های فعال و غیر فعال کردن وقفه های x86 استفاده می گردد.
تفاوت آنها این است که در pushcli و popcli قابلیت شمارش داریم یعنی مشخص است که هر کدام چقدر اجرا شده اند که میتواند در کنترل کردن کمک کند

3: چرا قفل مذکور در سیستم های تک هسته ای مناسب نیست؟ روی کد توضیح دهید.

```
24  void
25  acquire(struct spinlock *lk)
26  {
27    pushcli(); // disable interrupts to avoid deadlock.
28    if(holding(lk))
29      panic("acquire");
30
31    // The xchg is atomic.
32    while(xchg(&lk->locked, 1) != 0)
33      ;
34
35    // Tell the C compiler and the processor to not move loads or stores
36    // past this point, to ensure that the critical section's memory
37    // references happen after the lock is acquired.
38    __sync_synchronize();
39
40    // Record info about lock acquisition for debugging.
41    lk->cpu = mycpu();
42    getcallerpcs(&lk, lk->pcs);
43  }
```

همانگونه که در کد مشخص است بنا به شرط holding این قفل ها در xv6 به شکل busy waiting پیاده سازی شده و مشکل آنها در پردازه های تک هسته این است که در صورتی که یک پردازه به مدت طولانی قفل را در اختیار داشته باشد ، بقیه چون busy waiting هستند مشکل ایجاد می گردد.
Busy waiting ها در سیستم های چند پردازه ای باعث هدر رفتن زمان پردازنده و افت بهینه و کارایی سیستم می گردد ؛ در صورتی که در سیستم های تک پردازنده ای این اتفاق در بدترین حالت به deadlock منجر شده. مثلا چنانچه پردازه ای قفلی را اختیار کند و پردازه ای دیگر تلاش می کند قفل را به روش فوق بدست آورد در این صورت پردازه فوق هیچگاه خروجی از حلقه ندارد و پردازه های دیگر زمانبندی نمیشوند.

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

دستور amoswap یک 32 signed data بیتی را از آدرس rs1 در رجیستر rd به صورت اتمی لود می کند و جابجایی مقدار و مقدار اورجینال 32 بیتی signed vale در rs2 و نتیجه را در آدرس rs1 ذخیره می کند.
علت تعریف آن تامین وسیله ای برای پردازنده ها یا هسته های متعدد است. تا ثبات و صحت تضمین و دسترسی به حافظه مشترک هماهنگ گردد . بدون عملیات اتمی ، تضمین نظم و توالی های خوانداری و ویرایش و نوشتاری هنگامی که رشته های متعددی درگیر هستند دشوار بوده و می تواند منجر به داده های خراب گردد.

5: مختصری راجع به تعامل میان پردازه ها توسط دو تابع مذکور توضیح دهید.

```c
// Long-term locks for processes
struct sleeplock {
  uint locked;       // Is the lock held?
  struct spinlock lk; // spinlock protecting this sleep lock

  // For debugging:
  char *name;        // Name of lock.
  int pid;           // Process holding lock
};
```

```c
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);
}
```

همانگونه در sleeplock مشخص است به کمک locked وضعیت قفل بودن مشخص می شود و spinlock برای محافظت از کل اجزای استراکت بهره می بریم .
در acquiresleep یک پردازنده به روی آدرس قفلی که به آن پاس داده شده sleep کرده تا زمانی که فرصتی برای درست گرفتن قفل مذکور یافت نکند ؛ در release ریسه ای در sleeplock را نگه داشته تمام پردازه هایی که روی آن قفل sleep کرده اند را بیدار کرده و وضعیت را به RUNNABLE تغییر می دهد.

6:حالات مختلف پردازه ها در xv6 را توضیح دهید. تابع sched چه وظیفه ای دارد؟
حالت یک پردازه در متغیر state که از جنس enum procstate نگه می داریم.

- UNUSED : از پردازه ای استفاده نشده است.
- EMBRYO : در ابتدا حالت پردازه اینگونه است و هنگامی که از حالت UNUSED تغییر کرده و به این حالت می رویم پردازه دیگر UNUSED نمی باشد.
- SLEEPING : منبع پردازه هنوز آماده نبوده و پردازه در CPU نمیباشد که به این منزله است که scheduler از آن استفاده نمی کند ؛ می تواند داوطلبانه یا توسط کرنل به این حالت هدایت شود.
- RUNNABLE : قادر است که توسط scheduler به cpu اختصاص پیدا کند و اکنون در صف اجرای scheduler قرار دارد و به RUNNING خواهد شد.
- RUNNING : پردازه ای را گویند که CPU به آن اختصاص داده شده و در حال اجرا است.
- ZOMBIE : پردازه ای که کارش تمام شده اما پردازه پدر wait را صدا نکرده و اطلاعات این پردازه هنوز در ptable موجود است عملا وقتی میخواهد exit بکند، ابتدا ZOMBIE شده ؛ یعنی مستقیم به UNUSED نمیرود.

تابع sched برای context switch و در پایان کار یک پردازه صدا زده میشود وcontext فعلی ذخیره میکند و context با scheduler جایگزین میشود (به context پردازه RUNNABLE دیگری تعویض میکند.)

7: 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.

میتوان با ایده اینکه تنها صاحب پردازه بتواند عملیات را انجام دهد و دیگر مشکل موردنظر را نخواهیم داشت. عملا بایستی شرط اینکه چه پردازه ای در حال صدا کردن تابع است ، بررسی گردد در sleeplock از pid که برای اهداف دیباگینگ است برای چک کردن صاحب قفل استفاده می کنیم. در کد لینوکس در بخش mutex یک owner ای تعریف شده است که این موضوع با کمک آن چک می شود در لینک فوق از گیت هاب لینوکس ، در فایل mutex.h آمده است (در لینک فوق) که در اینجا نیز یک owner تعریف شده و شرط اینکه چه پردازه ای در حال صدا کردن تابع است بررسی می شود (این فیلد در حین رها کردن قفل چک میشود تا تنها صاحب قفل مجاز به این کار باشد.)

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

در الگوریتم های بدون قفل از قفلی و همچنین mutex استفاده نمی شود این عملیات اتمی (دستور های مقایسه و تعویض) ، عملیات اتمی از آنجایی که تقسیم ناپذیر هستند ضمانت می کنند که تنها یک فرآیند یا رشته می تواند فایلی را در یک زمان و بدون دخالت دیگران تغییر دهد. از چالش های الگوریتم های بدون قفل میتوان به منجر به نشت حافظه ، و پیچیدگی پیاده شوند .

از این الگوریتم ها معمولا در multithreading ها بهره می بریم و از ساختمان داده های معروف آن میتوان به concurrent stack اشاره داشت که push و pop در استک توسط ترد بدون نیاز به lock صورت میپذیرد.

از مزایا آن میتوان به افزایش و بهبود عملکرد و performance اشاره داشت و سربار کمتری دارند ؛ چون قفل نداریم منجر به بن بست نمی شود . ؛ اگر برنامه ای حین اجرا قفلی را اختیار کند و بدون ازاد کردن آن crash کند ، همه پردازه ها که از منبع فوق الذکر قفل شده استفاده می کردند از کار می افتند؛ که این در lock free رخ نمیدهد - همچنین پاسخگویی را بهبود بخشیده زیرا که ضمانت میکنند برخی از فرآیند ها یا رشته ها همواره پیشرفت کرده به یک فایل بستگی دارند.
از معایب آن به پیاده سازی سخت تر و اشکال زدایی دشوار تر میتوان اشاره کرد - همچنین الگوریتم های بدون قفل میتوانند سبب نشت حافظه بشوند.

پیاده سازی متغییر های مختص هر هسته پردازنده (Cache levels) :

الف : روشی جهت حل این مشکل در سخت افزار وجود دارد . مختصرا آن را توضیح دهید .

دو پروتکل برای حل این مشکل فراهم شده است :

1. Invalidation-Based Protocol

در این روش وقتی یک CPU در حافظه مینویسد ، یک پیغام "INVALIDATION" به باقی CPU ها داده میشود که میگوید این دیتا در کش شما لیبل invalidated دارد و دیگه معتبر نیست .
در دفعه بعدی که اون CPU ها به اون مقدار نامعتبر دسترسی بخوان داشته باشن ، fetch میکنند و مقدار جدید رو از main memory دریافت میکنند .

2. Update-Based Protocol

به جای اینکه در پروتکل قبلی یک پیغام invalidation برای باقی CPUها بفرستیم ، کافی است که مقدار آپدیت شده را به باقی CPU ها بدیم ، این پیعام شامل مقدار جدید است ، و CPU هایی که پیغام رو دربافت میکنند ، کش خود را بر اساس اون مسیج اپدیت میکنند ، اما این روش کمتر استفاده میشود چون میتواند با توجه به مقدار اپددیتی و نوع متغییر اپدیتی ترافیک ها بیشتری برای ارسال پیغام ایجاد کند .

ب : یکی از روش های همگام سازی استفاده از قفل هایی مرسوم به قفل بلیت است . این قفل هارا از منظر cache coherency بررسی کنید .

این مکانیزم با این که fair است اما میتواند مانند مکانیزم های دیگه با توجه به سربار زیاد برای update کردن و نگه داشتن cache coherency باعث مشکلاتی شود .
کارایی این روش بستگی به این دارد که CPU ها چند وقت یک باز cache invalidation , یا update cache ها رخ میدهد .
در بسیاری از کاربردها میتوان با استفاده از متغیرهای مختص هر هسته، مشکل را حل نمود. به این ترتیب که به جز در موارد ضروری، دسترسی و بهروزرسانی را در نسخه مختص هسته جاری از متغیر انجام میدهند. بدین ترتیب با کاهش تعداد معتبرسازی، سربار کاهش مییابد

ج) چگونه میتوان در لینوکس داده های مختص هر هسته را در زمان کامپایل تعریف نمود؟

(https://litux.nl/mirror/kerneldevelopment/0672327201/ch11lev1sec11.html#:~:text=Per%2DCPU%20Data%20at%20Compile%2DTime&text=DEFINE_PER_CPU(type%2C%20name)%3B,DECLARE_PER_CPU(type%2C%20name)%3B)

https://0xax.gitbooks.io/linux-insides/content/Concepts/linux-cpu-1.htm

https://www.cs.umd.edu/class/fall2017/cmsc412/project3.pdf

میتوان با استفاده از دستورات Per-CPU این اقدامات را انجام داد .

The kernel provides an API for creating per-cpu variables - the `DEFINE_PER_CPU` macro:

```
#define DEFINE_PER_CPU(type, name) \
        DEFINE_PER_CPU_SECTION(type, name, "")
```

تفاوت میان priorityLcok , ticketLock:

در ticketLock ما مشابه FIFO عمل میکنیم در واقع بهترین مثالی که میتوان برای این نوع از لاک زد تیکت ورود به بانک است که هر کس زودتر برسد زودتر به اون critical section خواهد رسید .
اما در priority Queue به این صورت نیست چون ممکن است یکی جز اولین نفر ها درخواست ورود به critical section را داشته باشد اما چون اولویت پایینی دارد به اواخر صف برود .

آیا این پیاده سازی ممکن است که دچار گرسنگی شود؟ راه حلی برای برطرف کردن این مشکل ارائه دهید. روش ارائه شده توسط شما باید بتواند شرایطی را که قفلها دارای اولویت یکسان می باشند را نیز پوشش دهد

بله ممکن است دچار starvation شود . در اینجا با توجه به این که PID ما مشخص میکند که چه پروسس ای اجرا شود و وقتی که ما وارد critical section شده ایم ممکن است پروس هایی که زودتر تولید شده اند ، بخواهد وارد critical شوند برای همین در این سناریو ممکن است starvation رخ دهد .

راه حل هایی از جمله aging مشابه با انچه در scheduling مطرح شد میتواند کمک کننده باشد .

خروجی userProgram برای تست PriorityLock :