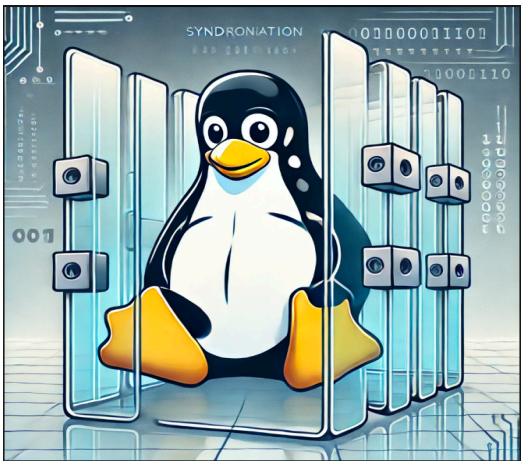
به نام خدا

آزمایشگاه سیستم عامل - پاییز ۱۴۰۳

پروژه ی چهارم: همگام سازی

طراحان: مهدی حاجی ، علی عطاالهی



مقدمه

در این پروژه با سازوکارهای همگام سازی[1] سیستم عامل ها آشنا خواهید شد. با توجه به این موضوع که سیستم عامل xv6 از ریسه[2] های سطح کاربر پشتیبانی نمی کند، همگام سازی در سطح پردازه ها مطرح خواهد بود. همچنین به علت عدم پشتیبانی از حافظه مشترک در این سیستم عامل ، همگام سازی در سطح هسته صورت خواهد گرفت. به همین سبب مختصری راجع به این قسم از همگام سازی توضیح داده خواهد شد.

[1] Synchronization

[2] Theard

ضرورت همگام سازی در هسته سیستم عامل ها

هسته سیستم عامل ها دارای مسیرهای کنترلی[3] مختلفی می باشد . به طور کلی ، دنباله دستورالعمل های اجرا شده توسط هسته جهت مدیریت فراخوانی سیستمی[4]، وقفه یا استثنا این مسیرها را تشکیل میدهند. در این میان برخی از سیستم عامل ها دارای هسته با ورود مجدد[5] می باشند ; بدین معنی که مسیرهای کنترلی این هسته ها قابلیت اجرای همروند[6] دارند. تمامی سیستم عامل های مدرن کنونی این قابلیت را دارند. مثلاً ممکن است برنامه سطح کاربر در میانه اجرای فراخوانی سیستمی در هسته باشد که وقفه ای 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 همگامسازی باید ماهیتهای مختلف کدهای اجرایی هسته لحاظ گردد.

یک مثال معروفی که برای توضیح همگام سازی و قفل گذاری استفاده می شود به صورت زیر است:

به عنوان مثال از قفل گذاری، پلی را تصور کنید که دارای محدودیت وزنی بر روی خود می باشد. 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 می کند 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[7] اجرا می گردد. در حالی که کدی که در نتیجه وقفه اجرا می گردد در متن وقفه [8] 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 عامل ها 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

[3] Control Flow (Control Path)
[4] System call
[5] Reentrant Kernel
[6] Concurrent
[7] Process Context
[8] Interrupt Context

گردد. مثلاً از قفلهای چرخشی [9] استفاده گردد یا در پردازنده های تک هسته ای وقفه غیر فعال گردد.

چگونگی همگام سازی در سیستم عامل XV6

قفل گذاری در هسته xv6 توسط دو سری تابع صورت میگیرد. دسته اول شامل توابع release و acquire میشود که یک پیاده سازی ساده از قفلهای چرخشی هستند. این قفلها منجر به Busy waiting شده و در حین اجرای ناحیه بحرانی وقفه را نیز غیرفعال میکنند.

- 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 releasesleep و acquiresleep 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.

- حالات مختلف پردازه ها در xv6 را توضیح دهید. تابع sched چه وظیفه ای دارد؟

[9] Spinlock

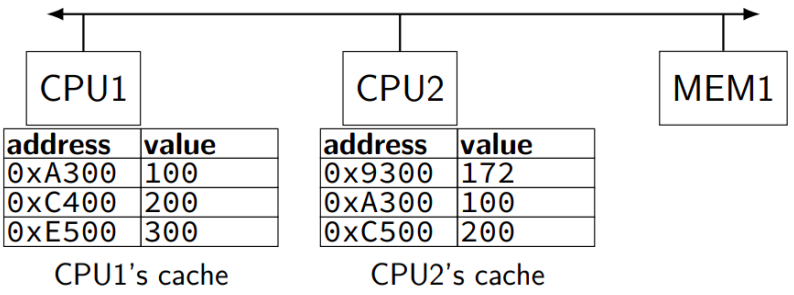
cache coherency در سیستم عامل XV6

یکی از روشهای افزایش کارایی در پردازنده ها استفاده از حافظه نهان [10] است. حافظه های نهان در سطوح مختلف وجود داشته و می توانند محلی [11]یا مشترک [12]باشند. به عنوان مثال، معمولاً حافظه نهان سطح یک [13]مختص هر هسته پردازنده بوده و لذا محلی است. بدین ترتیب هرگاه پردازه ای در یک متغیر حافظه بنویسد، مقادیر نگهداری شده برای این متغیر در حافظه های نهان سطح یک دیگر هسته های پردازنده را نامعتبر [14]می نماید.

البته معتبرسازی مقادیر حافظه نهان محلی، سربار قابل توجهی داشته و میتواند کارایی سیستم را پایین بیاورد.

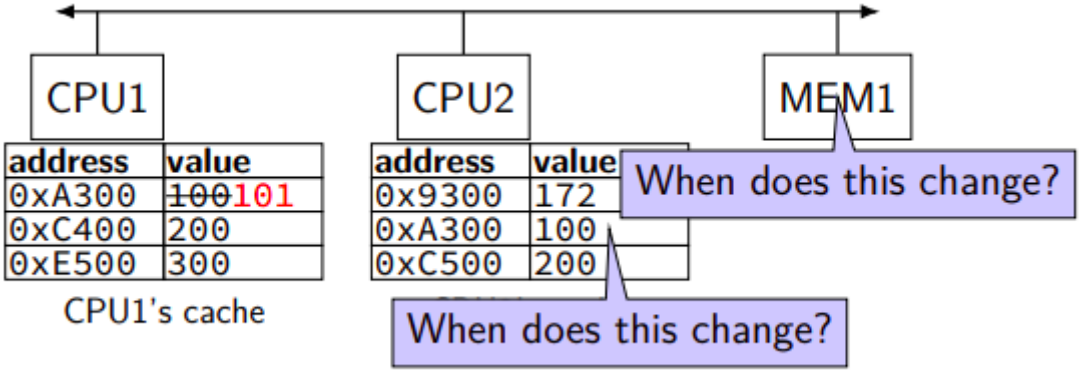
یک مثالی از این موضوع را باهم ببینیم :

فرض کنید استیت الان سیستم ما با دو پردازنده به صورت زیر است.



[10] cache
[11] local
[12] Global
[13] L1 cache
[14] invalid

حالا فرض کنید که در CPU1 ما مقدار 101 را در 0xA300 یادداشت کرده ایم :



مشکلی که حالا ایجاد خواهد شد این است که چه زمانی این مقدار در CPU2 و در MEM1 آپدیت خواهند شد ؟!

- یکی از روش های سینک کردن این حافظه های نهان با یکدیگر روش Modified-Shared-Invalid است. آن را به اختصار توضیح دهید. (اسلاید های موجود در منبع اول کمک کننده شما خواهند بود)
- یکی از روشهای همگام سازی استفاده از قفل هایی معروف به قفل بلیت[15] است. این قفلها را از منظر مشکل مذکور در بالا بررسی نمایید.

در بسیاری از موارد میتوان با استفاده از متغیرهای مختص هر هسته به جای یک متغیری که میان تمام CPU ها سینک باشد، مشکل را حل نمود. به این ترتیب که در موارد ضروری، دسترسی و به روزرسانی را در نسخه مختص هسته جاری از متغیر انجام میدهند. بدین ترتیب با کاهش تعداد معتبرسازی، سربار کاهش می یابد.

حالا مسئله ما به این شرح است :

ما سعی داریم که تعداد سیستم کال های اجرا شده توسط همه ی CPU های سیستم را داشته باشیم . اما به جای اینکه یک متغیر Global تعریف کنیم که در هر CPU یک متغیر به نام SysCallCounter تعریف می کنیم و انتظار داریم در نهایت جمع این متغیر بر روی تمام CPU ها برابر تعداد کل سیستم کال های فراخوانی شده باشد.

[15] Ticket Lock

با استفاده از این روش، یک فراخوانی سیستمی تعریف نمایید که تعداد فراخوانی های سیستمی اجرا شده در یک بار کاری را روی یک سیستم چهار هسته ای برمیگرداند. با این شرط که ما فرض میکنیم که سیستم کال Write دارای ضریب شمارشی 2 و سیستم کال Open دارای ضریب شمارشی 3 باشد و باقی سیستم کال ها ضریب ۱ داشته باشند .

System Call	Coeff. Of the system call's counter
Open	x3
Write	x2
Other system calls	x1

راهنمایی :

برای این منظور شما میتوانید یک متغیر جدید به struct موجود برای CPU اضافه کنید :

```
// Per-CPU state
struct cpu {
  uchar apicid;                // Local APIC ID
  struct context *scheduler;   // swtch() here to enter scheduler
  struct taskstate ts;         // Used by x86 to find stack for interrupt
  struct segdesc gdt[NSEGS];   // x86 global descriptor table
  volatile uint started;       // Has the CPU started?
  int ncli;                    // Depth of pushcli nesting.
  int intena;                  // Were interrupts enabled before pushcli?
  struct proc *proc;           // The process running on this cpu or null
};
```

همچنین تابع myCPU در فایل proc.c در هرلحظه CPU ای که در حال پردازش هست را به شما باز می گرداند :

```
40  // Must be called with interrupts disabled to avoid the caller being
41  // rescheduled between reading lapicid and running through the loop.
42  struct cpu*
43  mycpu(void)
44  {
45    int apicid, i;
46
47    if(readeflags()&FL_IF)
48      panic("mycpu called with interrupts enabled\n");
49
50    apicid = lapicid();
51    // APIC IDs are not guaranteed to be contiguous. Maybe we should have
52    // a reverse map, or reserve a register to store &cpus[i].
53    for (i = 0; i < ncpu; ++i) {
54      if (cpus[i].apicid == apicid)
55        return &cpus[i];
56    }
57    panic("unknown apicid\n");
58  }
```

همچنین برای هربار اضافه کردن این متغیر برای هر CPU میتواند در زمانی که سیستم عامل به trap میفتد اقدام کنید :

```
39  void
40  trap(struct trapframe *tf)
41  {
42    if(tf->trapno == T_SYSCALL){
43      if(myproc()->killed)
44        exit();
45      myproc()->tf = tf;
46      syscall();
47
48      if(myproc()->killed)
49        exit();
50      return;
51    }
52
```

برای اینکه از صحت اینکه مجموع تمام این متغیر ها برای هر CPU برابر کل سیستم کال های صدا زده شده است مطمئن شویم یک متغیر هم در خود Trap.c نگه دارید که هر بار یک سیستم کال صدا زده شد این متغیر اضافه شود .

برای اینکه بتوانید ضریب های متناسب را برای سیستم کال ها اختصاص دهید میتواند از TrapFrame ای که به تابع trap پاس داده میشود استفاده کنید :

```
147  //PAGEBREAK: 36
148  // Layout of the trap frame built on the stack by the
149  // hardware and by trapasm.S, and passed to trap().
150  struct trapframe {
151    // registers as pushed by pusha
152    uint edi;
153    uint esi;
154    uint ebp;
155    uint oesp;      // useless & ignored
156    uint ebx;
157    uint edx;
158    uint ecx;
159    uint eax;
160
161    // rest of trap frame
162    ushort gs;
163    ushort padding1;
164    ushort fs;
165    ushort padding2;
166    ushort es;
167    ushort padding3;
168    ushort ds;
169    ushort padding4;
170    uint trapno;
```

این struct در خود متغیر eax را دارد که این متغیر نشان دهنده id سیستم کالی است که باعث شده سیستم ما به حالت trap برود.

شماره سیستم کال ها در فایل syscall.h موجود است .

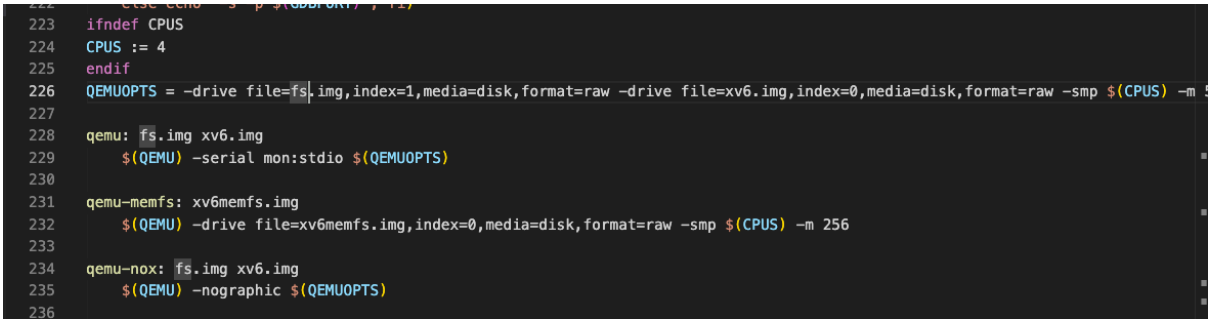
```
// System call numbers
#define SYS_fork    1
#define SYS_exit    2
#define SYS_wait    3
#define SYS_pipe    4
#define SYS_read    5
#define SYS_kill    6
#define SYS_exec    7
#define SYS_fstat   8
#define SYS_chdir   9
#define SYS_dup    10
#define SYS_getpid 11
#define SYS_sbrk   12
#define SYS_sleep  13
#define SYS_uptime 14
#define SYS_open   15
#define SYS_write  16
#define SYS_mknod  17
#define SYS_unlink 18
#define SYS_link   19
#define SYS_mkdir  20
#define SYS_close  21
```

در نهایت برای تست این سیستم کال باید اقدامات زیر را انجام دهید:

باید به تعداد کافی پردازه ایجاد نمایید که در فایل هایی می نویسند. همچنین جهت اطمینان از صحت عملکرد باید یک نسخه مشترک میان همه هسته ها تعریف شده و با مقدار برگشتی مقایسه گردد. دقت کنید همواره باید متغیر مربوط به هسته در حال اجرا، به روزرسانی گردد. (راهنمایی: روش پیاده سازی قفل چرخشی xv6 می تواند راهگشا باشد.)

و در نهایت مقدار متغیری که شمارش کل فراخوانی های سیستمی را در فایل trap.c نگه میدارد باید با مجموع تمامی متغیر های نسبت داده شده به هر CPU برابر باشد.

پیوست : در Makefile سیستم عامل XV6 شما میتوانید تعداد CPU ها را مشخص نمایید.



```
ifndef CPUS
CPUS := 4
endif
QEMUOPTS = -drive file=fs.img,index=1,media=disk,format=raw -drive file=xv6.img,index=0,media=disk,format=raw -smp $(CPUS) -m

qemu: fs.img xv6.img
	$(QEMU) -serial mon:stdio $(QEMUOPTS)

qemu-memfs: xv6memfs.img
	$(QEMU) -drive file=xv6memfs.img,index=0,media=disk,format=raw -smp $(CPUS) -m 256

qemu-nox: fs.img xv6.img
	$(QEMU) -nographic $(QEMUOPTS)
```

میوتکس پردازه مالک (Reentrant Mutex)[16]

در این بخش از پروژه، پیاده سازی میوتکس با قابلیت ورود مجدد مدنظر است. همانطور که میدانید پس از در اختیار گرفتن میوتکس توسط یک پردازه، تا زمان آزادسازی این میوتکس 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 بدنه ی این تابع بازگشتی، یک میوتکس را بگیرد ؛ در این صورت پردازه با بن بست[17] مواجه خواهد شد.

شما باید میوتکسی طراحی کنید که اگر پردازه در خود یک ناحیه بحرانی[18] دارد و در این ناحیه بحرانی تابع به صورت بازگشتی فراخوانی شود با بن بست مواجه نشود.

راهنمایی :

برای اینکه پروسس مالک بتواند با بن بست [19] مواجه نشود میتوانید خود مالک را به همراه تعداد فراخوانی های بازگشتی در یک struct ذخیره نمایید.

```
130  struct reentrantlock {
131    struct spinlock lock;  // Underlying spinlock for atomicity
132    struct proc *owner;    // Current owner of the lock
133    int recursion;         // Recursion depth for reentrancy
134  };
135
```

و احتمالا در ادامه نیازمند پیاده سازی این توابع با استفاده از struct بالا هستید :

- Initreentrantlock
- acquirereentrantlock
- releasereentrantlock

همچنین شما موظف به پیاده سازی یک برنامه کاربر[20] هستید تا صحت این قفل را تایید کنید.

---

[16] **Reentrant Mutex**: Keeps track of ownership and does not deadlock when the same thread reacquires the lock.
[17] Deadlock
[18] Critical section
[19] Deadlock
[20] User program

- دو مورد از معایب استفاده از قفل با امکان ورود مجدد را بیان نمایید.
- یکی دیگر از ابزار های همگام سازی قفل Read-Write lock است. نحوه کارکرد این قفل را توضیح دهید. و در چه مواردی این قفل نسبت به قفل با امکان ورود مجدد برتری دارد.

منابعی که می توانند مفید باشند

1. https://www.cs.virginia.edu/~cr4bd/4414/S2020/slides/20200211--slides-1up.pdf
2. https://pdos.csail.mit.edu/6.828/2023/lec/l-c_slides.pdf
3. https://en.wikipedia.org/wiki/Reentrant_mutex

سایر نکات

- **پروژه خود را در Github یا Gitlab پیش برده و در نهایت یک نفر از اعضای گروه کدها را به همراه پوشهٔ git. و گزارش زیپ کرده و در سامانه با فرمت OS-Lab4-<SID1>-<SID2>-<SID3>.zip آپلود نمایید.**
- **رعایت نکردن مورد فوق کسر نمره را به همراه خواهد داشت.**

- **بخش خوبی از نمرهٔ شما را پاسخ دهی به سوالات مطرح شده که با رنگ قرمز مشخص شده اند، تشکیل میدهد که به شما در درک نحوهٔ کارکرد xv6 و پیاده سازی قسمت سوالات کمک میکند.**

- پاسخ سوالها را تا حد ممکن کوتاه بنویسید.

- همه افراد می بایست به پروژه مسلط باشند و نمره تمامی اعضای گروه لزوما یکسان نیست.

- تمامی مواردی که در جلسه توجیهی، گروه اسکایپ و فروم درس مطرح می شوند، جزئی از پروژه خواهند بود. در صورت وجود هرگونه سوال یا ابهام میتوانید با ایمیل دستیاران مربوطه یا گروه اسکایپی درس در ارتباط باشید.

- این تمرین صرفا برای یادگیری شما طرح شده است. در صورت محرز شدن تقلب در تمرین، مطابق با قوانین درس برخورد خواهد شد.

موفق باشید