به نام خدا

سیستم‌های عامل (پاییز ۱۴۰۱)

فاز دوم

موازی‌سازی و زمان‌بندی

استاد درس:

آقای دکتر جوادی

مهلت نهایی ارسال پاسخ:

ساعت ۲۳:۵۹ روز ۵ بهمن ۱۴۰۱

مقدمه

در این فاز از پروژه قصد داریم که امکان تعریف چندین نخ[1] سطح کاربر[2] را به برنامه‌های سطح کاربر خود اضافه کنیم. همچنین در ادامه با بررسی الگوریتم زمان‌بندی پیشفرض سیستم عامل xv6 و تحلیل عملکرد آن، می‌خواهیم یک الگوریتم زمان‌بندی جدید با عنوان Lottery Scheduling به این سیستم عامل اضافه کرده و تفاوت عملکرد آن با الگوریتم پیش‌فرض را ارزیابی کنیم.

بخش اول: پیاده سازی User level thread

۱) آماده سازی پروژه

این قسمت از پروژه نیازمند تغییراتی در سیستم عامل پیش‌فرض xv6 است. برای دریافت این تغییرات، ابتدا کد سیستم عامل را از مخزن گیت‌هاب با کمک دستور زیر دریافت کنید:

```
git clone git://g.csail.mit.edu/xv6-labs-2022
```

پس از دریافت کردن مخزن، با دستورات زیر به برنچ مربوطه سوییچ می‌کنیم:

```
git checkout thread
make clean
```

پس از انجام این دستورات، فایل‌های user/uthread.c و user/uthread_switch.S مقصد تغییرات این بخش از پروژه خواهند بود.

۲) پیاده‌سازی

پس از دریافت کدهای پروژه اکنون می‌خواهیم نخ‌های سمت کاربر را به گونه‌ای پیاده سازی کنیم که با اجرای برنامه uthread پس از لود شدن سیستم عامل، با خروجی‌ای شبیه به خروجی زیر مواجه شویم:

[1] Thread
[2] User-Level

```
$ make qemu
...
$ uthread
thread_a started
thread_b started
thread_c started
thread_c 0
thread_a 0
thread_b 0
thread_c 1
thread_a 1
thread_b 1
...
thread_c 99
thread_a 99
thread_b 99
thread_c: exit after 100
thread_a: exit after 100
thread_b: exit after 100
thread_schedule: no runnable threads
$
```

با این هدف، باید توابع **thread_create()** و **thread_schedule()** را در فایل user/thread.c و تابع (مجموعه دستورات) **thread_switch** در فایل user/uthread_switch.S را تغییر دهید.

هدف از این تغییرات این است که با فراخوانی thread_schedule() و اجرای اولین نخ آماده، تابعی که در thread_create() به آن پاس داده شده را در **پشته‌ی[3] اختصاصی همان نخ** اجرا کند.

همچنین فراخوانی thread_switch (در اسمبلی) باید مقادیر رجیسترهای نخی که داریم از آن سوییچ می‌کنیم را ذخیره کند و مقادیر رجیسترهای نخ جدید را به گونه‌ای که سیستم آماده **ادامه اجرای** نخ جدید باشد، لود کند. این فراخوانی باید در تابع thread_schedule انجام شود و ۲ نخی که باید context switch بین آن‌ها انجام شود به عنوان پارامتر به این تابع پاس داده شوند.

چند نکته:

- برای ذخیره کردن رجیسترها می‌توانید استرکت thread را به گونه‌ای که تغییر دهید که قابلیت ذخیره و بازیابی مقادیر را داشته باشد.
- تغییرات اسمبلی فایل uthread_switch.S بسیار شبیه context switch ای که کرنل برای process ها انجام می‌دهد است و می‌توانید از کد مربوط به این قسمت کمک بگیرید.

[3] Stack

- برای درک بهتر این قسمت از پروژه می‌توانید یک breakpoint روی خط ۵۶ از فایل thread.c قرار دهید و با استفاده از GDB مراحل اجرا و محتوی Memory را بررسی کنید.

مواردی که باید تحویل دهید:

در نهایت با استفاده از دستور make grade می‌توانید صحت کار خود را بررسی کنید. برای این کار باید خروجی «**Test uthread**» برابر PASS باشد. **خروجی سایر قسمت‌ها مهم نیست**. تغییرات خود را به همراه سایر کدهای پروژه در کورسز آپلود کنید.

در تحویل آنلاین پروژه باید بتوانید منطق عملکرد multithreading و سوییچ شدن بین نخ‌های مختلف را به همراه نحوه به کارگیری آن توضیح دهید.

بخش دوم: پیاده سازی الگوریتم زمان‌بندی Lottery

۱) بررسی الگوریتم زمان‌بندی پیشفرض سیستم عامل

برای این بخش از پروژه **از همان کدی که در فاز قبل دریافت کرده‌اید،** استفاده می‌کنیم.

نحوه زمان‌بندی و انتخاب یک process را در اجرای main پیدا کنید و این روند انتخاب و اجرا را برای یک پردازه از ابتدا تا زمان محول کردن کنترل به CPU و انتخاب یک پردازه دیگر توسط GDB دنبال کنید. در انتها شما باید **روند طی شده و اتفاقاتی که در این میان در سطح کرنل رخ می‌دهد** را توضیح دهید.

توصیه می‌شود که فایل‌های زیر را بررسی کنید:

- kernel/main.c
- kernel/proc.c
- kernel/swtch.S

برای این بخش از پروژه، توصیه اکید ما مطالعه فصل ۷ ام از کتاب xv6 book تا زیربخش ۷,۷ است.

۲) اضافه کردن پیش‌نیازها برای الگوریتم زمان‌بندی جدید

برای آشنایی با نحوه عملکرد و توضیحات الگوریتم زمان‌بندی lottery based می‌توانید از این منابع استفاده کنید:

- زیر بخشی از کتاب OSTEP (پیشنهادی)
- Wikipedia
- GeeksForGeeks

به طور خلاصه، در این الگوریتم هر پردازه مقداری بلیط در اختیار دارد که درصدی از کل بلیط‌های قابل ارائه است، پردازنده در زمان انتخاب یک پردازه برای اجرا، یک انتخاب بین بلیط پردازه‌ها انجام داده و پردازه دارای بلیط را بر می‌گزیند. با این الگوریتم اگر به عنوان مثال ۷۵ درصد بلیط‌ها به پردازه A و ۲۵ درصد آن‌ها به پردازه B داده شود. با فرض صحیح بودن randomness انتخاب، می‌توانیم در درازمدت مطمئن باشیم که پردازه‌ها متناسب با مقدار بلیطی که دارند و با همین نسبت از CPU برای اجرا استفاده می‌کنند.

برای پیاده سازی این الگوریتم در ابتدا نیاز به پیاده‌سازی ۲ سیستم کال داریم.

int settickets(int number)

این سیستم کال مقدار بلیط‌های پردازه‌ای که سیستم کال را اجرا می‌کند برابر مقدار پاس داده شده تعریف می‌کند. این سیستم کال باید در صورت موفقیت ۰ و در صورت خطا عدد -۱ برگرداند.

در پیاده‌سازی این سیستم کال دقت داشته باشید که باید استراکت proc را تغییر دهید، همچنین در هنگام fork شدن و تولید یک پردازه child باید تعداد بلیط‌های پردازه child برابر تعداد بلیط‌های پردازه والد باشد.

int getprocessesinfo(struct processes_info *p)

این سیستم کال اطلاعاتی را درمورد سابقه اجرای **تمامی پردازه‌های تعریف شده در سیستم** به ما می‌دهد.

استراکت processes_info به شکل زیر تعریف می‌شود:

```
struct processes_info {
    int num_processes;
    int pids[NPROC];
    int ticks[NPROC];       // ticks = number of times process has been
scheduled
    int tickets[NPROC];     // tickets = number of tickets set by
settickets()
};
```

- مقدار فیلد num_processes برابر تعداد پردازه‌هایی است که فیلد state آن‌ها مخالف UNUSED باشد.
- مقدار سایر فیلدها برای عنصر i ام به معنی i امین پردازه در process table سیستم است:
  - pids[i] برابر process_id برای این پردازه است.
  - ticks[i] برابر تعداد دفعاتی است که این پردازه توسط scheduler انتخاب شده است.
  - tickets[i] برابر تعداد بلیط‌های این پردازه است.

این سیستم کال باید برای هر فراخوانی اطلاعات خواسته شده را برگرداند، در صورت بروز هرگونه خطا باید مقدار 1- و در صورت موفقیت آمیز بودن باید 0 برگردانده شود.

در ساختار proc باید تعداد بلیط‌های پردازه را به همراه tick هایی که اجرا شده است نگهداری کنید. برای نگهدار ticks در نظر داشته باشید که در سیستم‌عامل چنین متغیری برای شمردن واحدهای زمانی طی شده وجود دارد و باید از همین متغیر برای محاسبه واحد های زمانی طی شده استفاده کنید.

دقت کنید که ticks هر پردازه در هنگام fork شدن باید با مقدار ۰ مقدار دهی شود.

دقت کنید که اضافه کردن سیستم‌کال‌هایی که آرگومان دارند در سطح کرنل نیازمند تدبیرهای خاص است، با مطالعه کد سیستم‌کال‌های مشابه، نحوه به کار گیری argint و یا argptr را برای تعریف این سیستم‌کال‌ها پیدا کنید.

۳) پیاده‌سازی الگوریتم

یک مقدار پیشفرض را برای بلیط‌های اولیه هر پردازه در هنگام ساخته شدن در نظر بگیرید (مثلا ۱۰). همچنین یک سقف قابل قبول نیز برای بیشینه تعداد بلیط‌های هر پردازه باید در نظر بگیرید (مثلا ۱۰۰۰۰).

سپس باید scheduler سیستم‌عامل که عملکرد آن را در زیربخش اول بررسی کردیم به گونه‌ای تغییر دهیم که بر اساس تعداد بلیط‌های اختصاص داده شده به هر پردازه، به صورت رندم یک پردازه را برای اجرا انتخاب کند. **دقت کنید** که استفاده از توابع standard library در داخل کرنل امکان پذیر نیست، فلذا باید توابع مربوط به ایجاد یک عدد رندم را به کرنل اضافه کنید. برای اینکار مجاز به استفاده از خود توابع standard library یا کدهای موجود در اینترنت هستید.

مواردی که باید تحویل دهید

در قالب یک برنامه سطح کاربر، دو سیستم‌کال نوشته شده را تست کنید. سناریو تست به انتخاب خود شما است ولی می‌توانید از سناریوی پیشنهادی ما کمک بگیرید:

ابتدا در پردازه خود یکبار settickets را صدا زده و سپس یک fork از پردازه ایجاد کنید. داخل کد parent (قبل از به اتمام رسیدن child) سیستم‌کال getprocessesinfo را صدا زده و اطلاعات دریافت شده را با استفاده از یک حلقه در خروجی چاپ کنید.

(امتیازی) برای تست الگوریتم زمان‌بندی خود نیز باید یک برنامه سطح کاربر دیگر توسعه دهید. در این برنامه مجدد چندین child process ایجاد کرده و مقدار ticket آن‌ها را پس از ساختن تغییر دهید. در ابتدا تعداد بلیط‌های هر پردازه را چاپ کنید که نسبت آن‌ها قابل مقایسه باشد. پس از آن برای مدت زمان مشخصی اجازه دهید که هر پردازه اجرا شود (با استفاده از حلقه)، پس از آن با استفاده از سیستم کال getprocessesinfo تعداد ticks هریک از این child process ها را نشان دهید. با تغییر بازه زمانی اجرای

هر پردازه **باید به مقداری با نسبت‌های مشابه tickets ها برسید.** یعنی اگر پردازه اول ۱۰۰ بلیط، پردازه دوم ۲۰۰و پردازه سوم ۳۰۰ بلیط داشته باشند، نسبت ticks این ۳ پردازه باید ۱ به ۲ به ۳ باشد.

همچنین باید بتوانید نحوه عملکرد الگوریتم و زمان‌بندی را در ارائه آنلاین به صورت کامل توضیح دهید.

نحوه تحویل پروژه

- فایل‌های اضافه شده یا تغییر داده شده در XV6 را در قالب یک فایل zip با نام project_report_group_id_sid1_sid2.zip بارگذاری کنید.
- پروژه دارای تحویل به صورت آنلاین است و توضیح کد و توضیح عملکرد سیستم عامل پرسیده می شود و ***بخش مهمی از نمره*** را در بر می‌گیرد. پبس ضروری است که همه اعضای گروه به XV6 و پیاده‌سازی انجام شده تسلط داشته باشند.
- با توجه به انجام پروژه به شکل گروهی، تعداد commit های هر فرد باید متناسب باشد و خود گیت در حین تحویل چک می‌شود.
- لازم به ذکر است که تمامی پروژه‌ها پس از تحویل بررسی می‌شوند و هرگونه شباهت، به منزله نمره 0 برای تمام پروژه و تمرین‌ها خواهد بود.

منابع

- https://pdos.csail.mit.edu/6.828/2022/labs/thread.html
- https://www.cs.virginia.edu/~cr4bd/4414/S2019/lottery.html

موفق باشید

تیم درس سیستم‌های عامل