دانشکده مهندسی
کامپیوتر و فناوری اطلاعات

پروژه - فاز دوم

دکتر جوادی

سیستم‌های عامل

مقدمه:

در این فاز پروژه، هدف شما افزودن قابلیت Multi-Threading به کرنل آموزشی xv6 است. این قابلیت شامل ایجاد، مدیریت، و زمان‌بندی اجرای تردها در سیستم می‌شود. برای دستیابی به این هدف، باید مفاهیم زیر را به طور دقیق بررسی و پیاده‌سازی کنید:

آشنایی با مفاهیم اولیه:

۱. Scheduler

- کرنل xv6 به طور پیش‌فرض از **الگوریتم Round-Robin** برای زمان‌بندی استفاده می‌کند. این الگوریتم به صورت چرخشی به هر پردازه (قراره که ترد هم اضافه کنیم :)) زمانی برابر برای اجرا اختصاص می‌دهد.
- فانکشن Scheduler در فایل `kernel/proc.c` تعریف شده و مسئول جابجایی بین پردازه‌ها است. شما باید این الگوریتم را برای پشتیبانی از تردها بهینه‌سازی کنید.

۲. Context Switch

- **Context**:
  - ساختاری در xv6 که اطلاعات لازم برای ادامه اجرای یک پردازه را ذخیره می‌کند.
  - در فضای کرنل ذخیره می‌شود و برای انجام عملیات Context Switch استفاده می‌گردد.
- **Trapframe**:
  - ساختاری در xv6 که وضعیت یک پردازه (یا ترد) در فضای کاربر (User Space) را ذخیره می‌کند.
  - شامل Program Counter، Stack Pointer و مقادیر رجیسترها می باشد.
  - **نکته بسیار مهم**: برای هر ترد، باید یک trapframe جدید ایجاد شود. این trapframe باید به صورت **Dynamic** با استفاده از تابع `()kalloc` تخصیص داده شود.

- **Context Switch**:
  - فرآیندی که کرنل را قادر می‌سازد از اجرای یک پردازه به پردازه دیگر تغییر وضعیت دهد. این فرآیند شامل ذخیره context فعلی و بارگذاری context جدید است.

پیش نیاز: مطالعه pthread در زبان C

- برای درک نحوه کارکرد تردها، پیشنهاد می‌شود ابتدا با **کتابخانه pthread در زبان C** کار کنید. این کتابخانه امکان ایجاد، مدیریت، و همگام‌سازی تردها را فراهم می‌کند.
- موارد زیر را آزمایش کنید:
  - ایجاد یک ترد با تابع `pthread_create`.
  - همگام‌سازی تردها با تابع `pthread_join`.

فرق ترد و fork چیست؟

همانطور که در کلاس درس گفته شده - ترد یک نسخه lightweight تر از fork است و بر خلاف fork آدرس اسپیس را به اشتراک میگذارد. بنابراین به عنوان یک رفرنس از آنچه که میخواهیم انجام دهیم میتوانید در ابتدا نحوه پیاده سازی سیستم کال fork را بررسی کنید.

چه قابلیت هایی میخوایم پیاده سازی کنیم؟

در این فاز میخواهیم سه سیستم کال ابتدایی برای ایجاد ترد اضافه کنیم:

- **create_thread**

از این سیستم کال برای ساخت یک ترد جدید استفاده می شود. یک پوینتر به یک فانکشن (فانکشنی که ترد باید آن را اجرا کند) و یک arg ورودی برای ورودی به فانکشن مورد نظر گرفته می شود. در نتیجه خروجی آیدی ترد ساخته شده برگشت داده می شود. (یا یک عدد منفی در صورت خطا)

- **join_thread**

این سیستم کال آیدی یک ترد ساخته شده را ورودی می گیرد و سپس ترد فعلی را تا زمانی که ترد مورد نظر تمام نشده است متوقف می کند. و پس از اتمام ترد مورد نظر باید به کار خود ادامه دهد.

- **stop_thread**

این سیستم کال آیدی یک ترد را میگیرد و آن را متوقف می کند. اگر عدد منفی یک وارد شود ترد فعلی را متوقف می کند. در صورتی که ترد فعلی از قبل در حال اجرا نبود یک عدد منفی بر میگرداند و اگر ترد با موفقیت متوقف شد عدد 0 را بر میگرداند.

به چه استراکت های جدیدی نیاز داریم؟

برای پیاده سازی لازمه که استراکت های زیر رو به proc.h اضافه کنیم:

```
#define MAX_THREAD  4

enum threadstate {
    THREAD_FREE,
    THREAD_RUNNABLE,
    THREAD_RUNNING,
    THREAD_JOINED
};

struct thread {
    enum threadstate state;
    struct trapframe *trapframe;
    uint id;
    uint join;
};
```

و برای این که ترد رو به پردازه ها اضافه کنیم دو دیتای زیر رو هم به استراکت proc اضافه میکنیم:

```
// threads
struct thread threads[MAX_THREAD];
struct thread *current_thread;
```

دقت کنید که در شروع current_thread باید مقدارش NULL=0 باشد و به این معنا است که این پردازه تا به حال تردی ایجاد نکرده است. هنگام ایجاد اولین ترد باید اطلاعات فعلی مربوط به پردازه را در یک ترد ذخیره کنید. (در واقع پردازه به چند ترد شکسته خواهد شد و اولین ترد همان ترد Main خواهد بود.)

شما باید Scheduler را نیز طوری تغییر دهید تا ترد های مختلف بتوانند در این زمان بند نقش داشته باشند و برای اجرا شدن زمان بندی شوند.

اطلاعات مورد نیاز در Trapframe:

برای اجرای صحیح تردها، باید اطلاعات زیر در `trapframe` هر ترد تنظیم شود:

1. **Return Address `ra`:**
   - آدرسی که پس از اتمام اجرای تابع به آن باز می‌گردد.
   - مقدار این آدرس می‌تواند به گونه‌ای تنظیم شود که وقتی تابع اصلی ترد (entry point) اجرا شد، کنترل به یک تابع پایان‌دهنده بازگردد که منابع ترد را آزاد می‌کند.
2. **Program Counter `epc`:**
   - آدرسی که اجرای کد از آنجا شروع می‌شود. این آدرس باید به نقطه ورود تابع اصلی ترد (entry point) تنظیم شود.
3. **`a0-a4` (ورودی‌های تابع):**
   - پارامترهایی که میتوانیم استفاده کنیم تا به تابع اصلی ترد ارسال کنیم.
4. **Stack Pointer `sp`:**
   - آدرس شروع stack ترد.

نکات قابل توجه برای پیاده سازی:

هر ترد باید Stack مخصوص به خود داشته باشد:

- Stack هر ترد باید به صورت مستقل و در فضای **User Space** تخصیص داده شود.
- می‌توانید این آدرس را در ورودی تابع `create_thread` دریافت کنید. این به پردازه والد اجازه می‌دهد تا کنترل بیشتری روی تخصیص Stack برای ترد جدید داشته باشد.
- مقدار `sp` در trapframe باید به انتهای Stack تخصیص داده‌شده تنظیم شود.

شناسایی پایان یافتن اجرای ترد:

- برای شناسایی اتمام ترد، می‌توانید مقدار Return Address در `trapframe` را به یک مقدار نامعتبر تنظیم کنید.
- در هنگامی که ترد به پایان می‌رسد و تلاش می‌کند به آدرس `ra` بازگردد، یک Trap ایجاد می‌شود. این Trap را می‌توان در کرنل پردازش کرد تا تشخیص داده شود که ترد به پایان رسیده است.
- مقدار خروجی تابع ترد پس از اتمام آن در رجیستر a0 ذخیره می‌شود. در ترپ ایجاد شده می توانید آن را از `trapframe` بخوانید.
- در اینجا باید تمام منابع اشغالی برای ترد را آزاد کنید و استیت آن را به FREE تغییر دهید. سپس ترد جدیدی را برای اجرا آماده سازی کنید و عملیات Context Switch را مجددا انجام دهید.

تست:

بخش تست برنامه شما همچین حالتی خواهد بود:

```
volatile int a = 0, b = 0, c = 0;

void *my_thread(void *arg) {
    int *number = arg;
    for (int i = 0; i < 100; ++i) {
        (*number)++;

        if (number == &a) {
            printf("thread a: %d\n", *number);
        } else if (number == &b) {
            printf("thread b: %d\n", *number);
        } else {
            printf("thread c: %d\n", *number);
        }
    }
    return (void *) number * 2;
}

int main(int argc, char *argv[]) {
    int ta = create_thread(my_thread, (void *) &a);
    int tb = create_thread(my_thread, (void *) &b);
    int tc = create_thread(my_thread, (void *) &c);
    join_thread(ta);
    join_thread(tb);
    join_thread(tc);
    exit(0);
}
```

بخش امتیازی:

پس از انجام این بخش ما ترد ها را در xv6 خواهیم داشت! اما همچنان این ترد ها نمی توانند همزمان توسط cpu های مختلف به طور موازی اجرا شوند. بخش امتیازی این خواهد بود که بتوانید این مشکل را حل کنید تا ترد های مختلف توسط cpu های متفاوت قابل اجرا باشند.

توضیحات پایانی:

از شما درخواست داریم که یک **private repository در گیت هاب** درست کنید و تغییرات کد خود را مرحله به مرحله **Commit** کنید و در صورت تمایل می‌توانید هر یک از تدریس یاران را به پروژه خود اضافه کنید. دقت کنید که شما نبایستی برنامه‌های خود را با دیگر دانشجویان به اشتراک بگذارید.

توضیحات

- یک نفر از اعضای گروه کافیست که پروژه را آپلود کند.
- پروژه شما تحویل آنلاین خواهد داشت بنابراین از استفاده از کدهای دیگران یا کدهای موجود در وب که قادر به توضیح دادن عملکرد آنها نیستید، بپرهیزید.
- ابهامات خود را در گروه درس در تلگرام مطرح کنید و ما در سریع‌ترین زمان ممکن به آنها پاسخ خواهیم داد.

آنچه که باید ارسال کنید:

- یک فایل زیپ با نام **OS_P2_Sid1_Sid2.zip** که شامل دو مورد زیر است:
  1. **گزارش خیلی مختصر** از آنچه که انجام داده‌اید.
  2. پوشه ای که در آن کدهای شما وجود دارد. **دقت کنید که تنها و تنها فایل‌هایی را که تغییر داده‌اید یا اضافه کرده‌اید را برای ما بفرستید.**

موفق باشید

تیم تدریس یاری درس سیستم های عامل