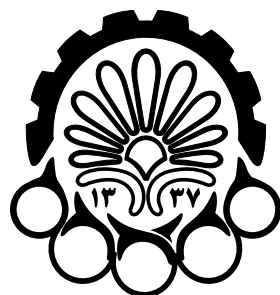


به نام خدا



دانشگاه صنعتی امیرکبیر  
( پلی تکنیک تهران )

سیستم‌های عامل (پاییز ۱۴۰۰)

## فاز دوم

استاد درس:

آقای دکتر جوادی

مهلت نهایی ارسال پاسخ:

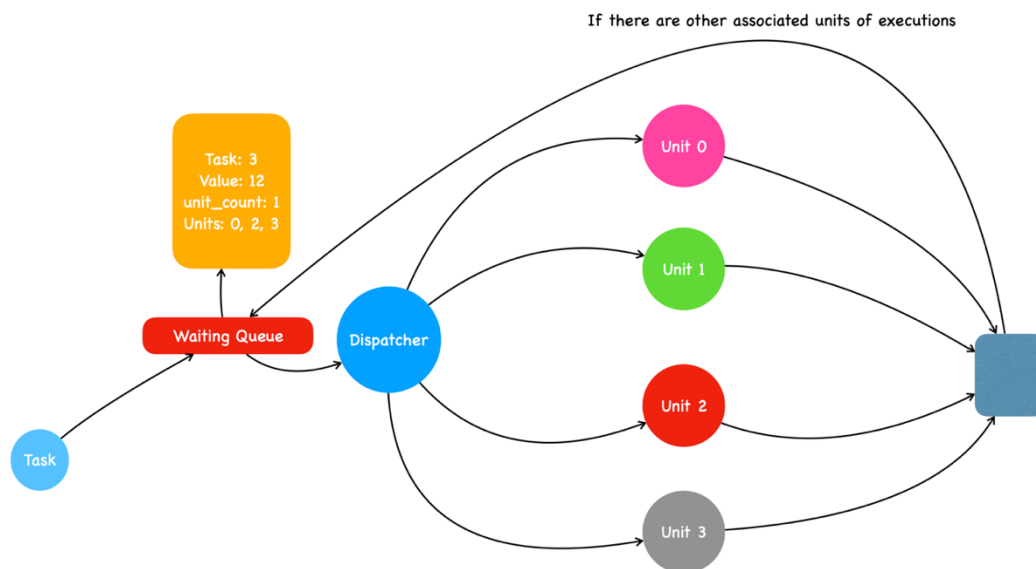
۲۶ آذر ماه ۱۴۰۰

دقت کنید که امکان تمدید وجود ندارد.

## توصیف پروژه

یک سیستم پیچیده مانند یک سازمان بزرگ را در نظر بگیرید که واحدهای پردازشی جدایی برای هر کار دارد. در این سازمان برای آنکه هر کدام از کارها بطور کامل انجام شوند نیازمند مجموعه‌ای از پردازش‌ها در هر کدام از واحدهای پردازشی هستند. تمامی کارها در صف انتظار منتظر می‌مانند تا توزیع کننده، کار مورد نظر را انتخاب کند و به واحد پردازشی مربوطه تحویل دهد.

شکل زیر ساختار سیستم موردنظر را نشان می‌دهد :



شما باید بتوانید چنین مکانیزمی را در پروژه خود طراحی کنید.

## جزئیات پیاده‌سازی

به منظور پیاده‌سازی مکانیزم خواسته شده باید فراخوانی‌های سیستمی زیر را پیاده‌سازی کنید :

```
int thread_create(void *stack, int status)
```

```
int thread_wait()
```

```
int unit0_operation()
```

```
int unit1_operation()
```

```
int unit2_operation()
```

```
int unit3_operation()
```

همچنین لازم است که یک تابع سطح کاربر با امضای زیر پیاده‌سازی کنید که به عنوان تابع پوششی برای `thread_create` ایفای نقش می‌کند و هدف آن استفاده آسان‌تر از این فراخوانی سیستمی است.

```
int thread_creator(void (*fn) (void *), void *arg, int status)
```

فراخوانی سیستمی `thread_create` بسیار شبیه به `fork` عمل می‌کند با این تفاوت که در این تابع به جای کپی کردن فضای آدرس به یک `page directory` جدید، این فراخوانی سیستمی وضعیت پردازش جدید را طوری مقداردهی می‌کند که پردازش فرزند و پردازش پدر فضای آدرس را به اشتراک بگذارند (به عبارت دیگر از یک `page directory` مشابه استفاده کنند). اگر این فراخوانی سیستمی به این شکل نوشته شود دو پردازش فضای حافظه را به اشتراک خواهند گذاشت و این دو عملاً ریسمان هستند.

اگرچه دو ریسمان پدر و فرزند فضای آدرس را به اشتراک می‌گذارند، هر کدام به یک پشته جداگانه نیاز دارند. فضای پشته جدید احتمالاً با استفاده از `malloc` تخصیص داده می‌شود. به عنوان مثال فرض کنید که ریسمان `T1` یک کپی از خود به عنوان `T2` را ایجاد می‌کند. `T1` ابتدا حافظه‌ای به اندازه یک صفحه که `page-aligned` است را به پشته `T2` اختصاص داده و اشاره‌گر به آن را به فراخوانی سیستمی `thread_create` ارسال می‌کند. از طرفی دیگر، فراخوانی سیستمی `thread_create` پشته ریسمان `T1` را به این نقطه از حافظه کپی می‌کند و ثبات‌های پشته و پایه `T2` را تغییر می‌دهد تا از پشته جدید استفاده کند. توجه شود که `T2` باید از سابقه اجرای `T1` مطلع باشد و این یعنی باید بتواند به آیتم‌های موجود در پشته `T1` به عنوان مقدار اولیه پشته جدید دسترسی داشته باشد.

تابع `thread_creator`، اشاره‌گر به یک تابع و آرگومان‌های ریسمان و یک `status` را به عنوان ورودی می‌گیرد و کارهای زیر را انجام می‌دهد:

- یک فضای `page-aligned` به پشته اختصاص دهد.
- فراخوانی سیستمی `thread_create` را صدا بزند که `thread ID` را به پدر بر می‌گرداند.
- در ریسمان فرزند، `thread_creator` باید اشاره‌گر به تابع ارسالی را صدا بزند و آرگومان‌های ارسالی به عنوان پارامترهای ورودی در اختیار این تابع قرار دهد، همچنین باید مقدار `status` نیز پاس داده شود.
- هنگامی که اجرای تابع ارسالی به اتمام رسید، `thread_creator` باید پشته را خالی کرده و `exit` را صدا بزند.
- فراخوانی سیستمی `thread_wait` بسیار شبیه فراخوانی سیستمی `wait` می‌باشد و در واقع منتظر می‌ماند تا اجرای ریسمان فرزند تمام شود و `wait` منتظر می‌ماند تا پردازش فرزند تمام شود.

متغیر `status` در پیاده‌سازی شما سه حالت دارد:

۱. اگر صفر باشد یعنی `thread` ای که باید ساخته شود باید از نوع `Unit` باشد.

۲. اگر یک باشد یعنی ریسمانی که باید ساخته شود باید از نوع `Task` باشد.

۳. اگر دو یا بیشتر باشد باشد یعنی ریسمانی که باید ساخته شود باید از نوع معمولی می باشد (هیچ یک از Task یا Unit نیست).

در struct proc باید یک متغیر به اسم status تعریف شود که مشخص کند که ترد ساخته شده از نوع Unit می باشد یا Task.

### مشخصات ریسمان از نوع Unit:

تردهایی که از نوع Unit می سازید ویژگی های زیر را دارا هستند :

مقدار متغیر status برای آنها در struct proc برابر با صفر می باشد.

یک متغیر به عنوان unit\_num که مشخص کننده شماره واحد پردازشی موردنظر می باشد.

### مشخصات ریسمان از نوع Task :

ریسمان هایی که از نوع Task می سازید ویژگی های زیر را دارا هستند:

- مقدار متغیر status برای آنها در struct proc برابر با یک می باشد.
- یک متغیر unit\_count در struct proc مربوط به این ریسمان ها تعریف می شود که از صفر شروع می شود و با هر بار ورود به یک واحد پردازشی و پردازش شدن، مقدار آن یک واحد زیاد می شود.
- یک لیست از واحدهای پردازشی که Task مورد نظر نیاز دارد تا پردازش های لازم بر روی آن صورت گیرد.
- یک متغیر Value که مقدار ریسمان Task مورد نظر را در خود نگهداری می کند.

### فراخوانی های سیستمی مربوط به هر واحد پردازشی:

- `int unit0_operation()`

این فراخوانی سیستمی پردازش مربوط به واحد پردازشی اول را انجام می دهد که در واقع مقدار Value یک Task را می گیرد و عملیات زیر را بر روی آن انجام می دهد:

Adds 7 to the value then modulate by M

- `int unit1_operation()`

این فراخوانی سیستمی پردازش مربوط به واحد پردازشی دوم را انجام می دهد که در واقع مقدار Value یک Task را می گیرد و عملیات زیر را بر روی آن انجام می دهد:

Multiplies by 2 then modulate by M

- `int unit2_operation()`

این فراخوانی سیستمی پردازش مربوط به واحد پردازشی سوم را انجام می‌دهد که در واقع مقدار Value یک Task را می‌گیرد و عملیات زیر را بر روی آن انجام می‌دهد:

Multiplies by 3 then modulate by M

- `int unit3_operation()`

این فراخوانی سیستمی پردازش مربوط به واحد پردازشی چهارم را انجام می‌دهد که در واقع مقدار Value یک Task را می‌گیرد و عملیات زیر را بر روی آن انجام می‌دهد:

Prints out the value

متغیر M یک مقدار **const** است، مقدار آن را ۲ قرار دهید.

نکته : در واقع برای پیاده‌سازی خود باید هر یک از فراخوانی‌های سیستمی مربوط به هر واحد پردازشی را در یک تابع جدا فراخوانی کرده و اشاره‌گر به این تابع را به تابع پوششی `thread_creator` متناظر با آن واحد پردازشی پاس دهید.

دقت کنید که هر واحد پردازشی باید بین تمامی ریسمان‌های Task جستجو کند و ریسمانی را پردازش می‌کند که `list(unit_count)` برای آن Task برابر با واحد پردازشی فعلی باشد.

توجه شود که در حالت عادی ریسمان‌های متعلق به یک پردازش باید `process id` یکسان داشته باشند. با این وجود در این پروژه اشکالی ندارد که `getpid` برای ریسمان‌های متعلق به یک پردازش مقادیر متفاوتی را برگرداند. در واقع شما می‌توانید از `pids` به عنوان `thread ids` استفاده کنید تا بتوانید با کمینه تغییرات، پیاده‌سازی خود را انجام دهید.

توجه شود که باید اطمینان حاصل کنید که در هنگام اجرای `exit` توسط یک ریسمان، پیاده‌سازی شما جدول صفحه (page table) این ریسمان را در صورت وجود دیگر ریسمان‌هایی که به این جدول صفحه اشاره می‌کنند آزاد (free) نمی‌کند، برای این کار می‌توانید از مکانیزم شمارش ارجاعات (reference counting) استفاده کنید یا می‌توانید با اسکن کردن جدول پردازش (process table) چک کنید که اگر ارجاع دیگری به جدول صفحه پیدا نشد، جدول صفحه در نتیجه ی `exit` آزاد شود.

همچنین برای پیاده‌سازی فراخوانی سیستمی `thread_create` باید به نکات زیر توجه داشته باشید:

(۱) جلوگیری از شرایط مسابقه یا `race condition` در هنگام گسترش فضای آدرس

(۲) آزاد (free) نکردن جدول صفحه (page table) تا وقتی که تمامی ریسمان‌های متعلق به یک پردازش به اتمام برسند.

راهنمایی : می‌توانید یک تابع `dispatcher` درست کنید و در این تابع چهار ریسمان جدا برای هر واحد پردازشی بسازید و برای هر واحد پردازشی تابعی تعریف کنید که سیستم کال مربوط به واحد پردازشی موردنظر را فراخوانی کند و این تابع را به

thread\_creator مربوط به واحد پردازشی موردنظر پاس دهید و همچنین چند ریسمان از نوع Task ایجاد کنید و آنها را به حالت Busy waiting یا sleep ببرید تا پردازش های مورد نیاز آنها در هر کدام از واحدهای پردازشی انجام شود (دقت کنید که هر واحد پردازشی خود جدول پردازها اسکن می کند و تردهایی که نیاز به پردازش دارند را پردازش می کند (در واقع این پیاده سازی راحت ترین نوع پیاده سازی است که می توانید انجام دهید 😊)، در واقع ریسمان های Task کاری انجام نمی دهند و فقط منتظر آن هستند که پردازش های آنها توسط واحدهای پردازشی موردنظر انجام شده تا تمام شوند).

همچنین می توانید از فراخوانی سیستمی (getprocs) که در فاز اول پروژه پیاده سازی کردید، که تعداد پردازش های فعال در سیستم را به شما برمی گرداند، کنترل کنید که تعداد پردازش های سیستم در هر لحظه چگونه تغییر می کند و با استفاده از این سیستم کال می توانید متوجه شوید که چه تعداد از Task شما تمام شده اند و چه تعداد همچنان نیاز به پردازش دارند.

توجه شود که تستی که می نویسید باید بتواند عملکرد سیستم را به خوبی نشان دهد.

توجه شود که تصویری که در ابتدای پروژه قرار داده شده است صرفاً برای دید بهتر شما نسبت به عملکرد سیستم قرار داده شده است و پیاده سازی شما باید مطابق توضیحات گفته شده باشد.

موفق باشید

تیم درس سیستم های عامل