اصول سیستم عامل

نیم‌سال دوم ۱۴۰۰-۱۴۰۱

مدرس: مجتبی‌استواری

دانشکده علوم ریاضی

تمرین سری دوم

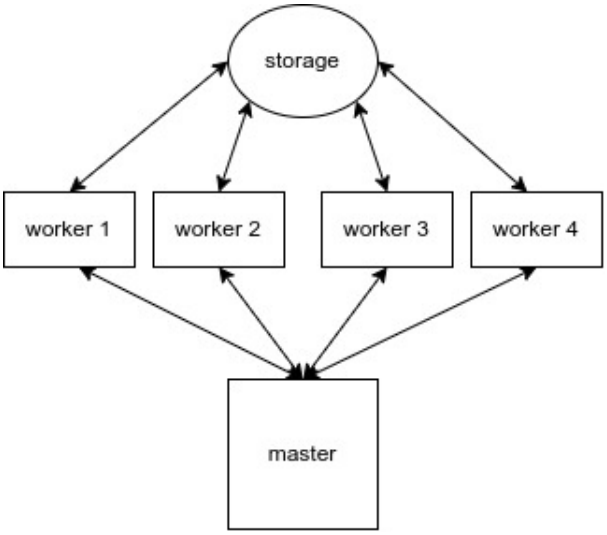
مقدمه

همان‌طور که می‌دانید یکی از وظایف سیستم عامل، مدیریت و زمان‌بندی عملیات‌های کاربر یا سیستم است. هدف این تمرین، شبیه‌سازی این وظیفه در سطح نرم‌افزار است. بنابراین در این تمرین به چالش‌هایی چون پیاده‌سازی الگوریتم‌های زمان‌بندی [۱] و context switch بین پردازه‌ها و رفع بن‌بست [۲] میان چند پردازه [۳] برخورد خواهید کرد.

ساختار کلی

جهت شبیه‌سازی عملکرد سیستم عامل، تعدادی کارگر نقش پردازنده [۴] را ایفا می‌کنند هر کارگر به طور مستقل می‌تواند یک کار [۵] را انجام دهد و نتیجه آن را به سیستم عامل برگرداند. کارها مانند پردازه‌ها [۶] می‌باشند و در طول برنامه ممکن است چندین بار متوقف [۷] و جایگزین [۸] شوند.

در اینجا عمده نقش سیستم عامل توسط پردازه اصلی ایفا می‌شود که همچنین وظیفه دارد پردازه‌های کارگر [۹] را بسازد و به آن‌ها متصل شود. پردازه‌ای دیگر نیز نقش حافظه [۱۰] را خواهد داشت که آن نیز روی شبکه TCP آماده می‌شود.



شکل ۱: شمای کلی سیستم

[۱] scheduling
[۲] deadlock
[۳] process
[۴] processor/core
[۵] task
[۶] process
[۷] interrupted
[۸] context switched
[۹] worker
[۱۰] storage

بخش‌های برنامه

سرور اصلی [11]

سرور اصلی پردازه‌های کارگر و حافظه را راه‌اندازی می‌کند و به آن‌ها متصل می‌شود. همچنین همه کارها را از همان ابتدای برنامه ورودی می‌گیرد و با توجه به الگوریتم زمان‌بندی مشخص شده شروع به اجرای آن‌ها می‌کند.

حافظه

سرور حافظه امکان دسترسی به خانه‌های مشخصی از حافظه را می‌دهد. پس از گرفتن دسترسی می‌توان بر روی این خانه‌ها نوشت، یا از آن‌ها خواند. در نهایت نیز می‌توان دسترسی به آن خانه را برای سایر کارگران آزاد کرد.

کارگر

هر کارگر یک پردازه است که در صورتی که به آن کاری اختصاص داده شده باشد آن را به موازات سایر کارگران اجرا می‌کند.

کار

هر کار دنباله‌ای از خواندن‌های متوالی از حافظه به همراه مقداری تاخیر [12] بین هر خواندن است. هدف محاسبه کردن جمع اعداد موجود در خانه‌های حافظه است. یک کار پس از خواندن آخرین خانه حافظه خواسته شده جمع کل را محاسبه می‌کند و نتیجه را برمی‌گرداند.

یک کارگر تنها پس از اتمام کامل کار، آغاز به آزادکردن قفل‌های گرفته شده می‌کند. همچنین دسترسی‌های پشت سر هم به یک خانه از حافظه مجاز است.

پردازش کارها و برنامه‌ریزی توسط سرور اصلی

سرور اصلی یک صف از کارهایی که به وی محول شده‌است به همراه وضعیت آن‌ها نگهداری می‌کند. سرور هر ۱ میلی‌ثانیه صف خود را بررسی می‌کند. سپس با توجه به نوع الگوریتم زمان‌بندی (که هنگام اجرا مشخص شده‌است) به یکی از سه شکل زیر اقدام به زمان‌بندی و پردازش کار می‌کند (دقت کنید که الگوریتم‌های معرفی شده تفاوت‌هایی با آن‌چه که در کتاب معرفی شده‌است دارند):

First Come First Serve (FCFS)

کارها به همان ترتیبی که وارد صف شده‌اند بین کارگرهای خالی توزیع می‌شوند. اگر کارگری خالی نبود در صف منتظر می‌مانند.

Shortest Job First (SJF)

در این حالت سرور اصلی همواره تلاش می‌کند از بین کارهایی که در صف اجرای خود دارد، اولویت را به کاری که کمترین زمان اجرا را دارد بدهد. زمان اجرای هر کار به طور تقریبی مجموع زمان‌های تعلیق (sleep) موجود در آن است.

Round Robin (RR)

در این حالت بازه زمانی یکسانی برای هر کار اختصاص داده می‌شود. پس از پایان هر بازه، کار از کارگر گرفته شده و کار بعدی ارسال می‌شود. این چرخه تا زمانی ادامه دارد که همه کارها تمام شوند.

توجه کنید چون تمام کارها از اول داده شده‌اند، در دو الگوریتم اول preemption رخ نمی‌دهد.

---

[11] master

[12] delay/sleep (به میلی‌ثانیه)

قواعد توقف و ادامه‌ی یک کار

در هر مرحله از اجرای یک کار ممکن است سرور بخواهد preemption انجام دهد و آن کار را از دست کارگری که در حال انجامش است خارج و متوقف کند. کارهای متوقف‌شده‌ای که تکمیل نشده‌اند براساس زمان‌بندی سرور در زمانی دیگر (یا بلافاصله) برای کارگری دیگر (یا همان کارگر) برای ادامه یافتن از همان‌جایی که متوقف شده‌اند ارسال خواهد شد. شما باید یک مکانیزم context switch پیاده سازی کنید. ممکن است که در هر لحظه‌ای از زمان برای یک کار جاری در یک کارگر، از سمت سرور درخواست توقف ارسال شود، اما لزومی ندارد که کارگر بلافاصله اجرای کار را متوقف کند .

برای توقف و ادامه یک کار، بسته به وضعیت فعلی کار آن کارگر، به یکی از طریق زیر عمل کنید:

۱. اگر کارگر در خواب بود، میزانی که تا کنون خوابیده را ذخیره می‌کند و هنگام ادامه، به همان میزان، کمتر می‌خوابد.

۲. اگر منتظر دریافت قفل حافظه بود، پس از ادامه از نو درخواست دسترسی به حافظه می‌دهد.

۳. اگر در حال انجام محاسبه باشد، ابتدا نتیجه را حساب و ذخیره می‌کند و سپس کار را متوقف می‌کند.

به عبارتی پس از توقف یک کار باید وضعیت آن به شکل مناسب ثبت و به سرور ارسال شود.

بن‌بست

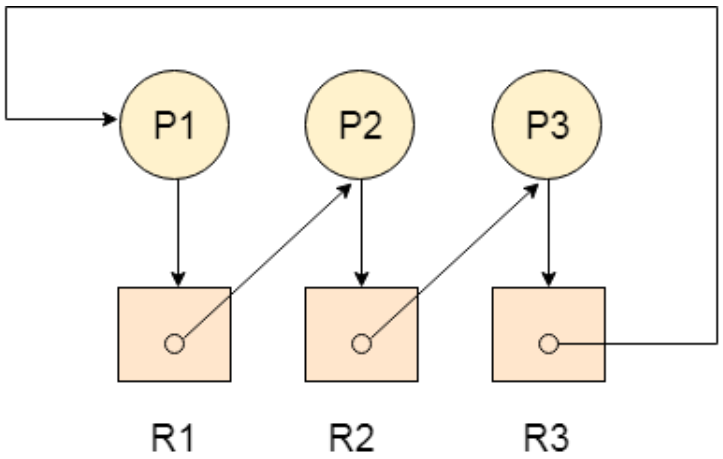
ممکن است دو یا چند کار به صورتی درخواست از منابع حافظه داشته باشند که دچار بن‌بست شوند. شما باید راه حلی برای این مشکل پیاده‌سازی کنید. این راه حل ممکن است مبتنی بر پیش‌گیری [۱۳] یا تشخیص بن‌بست [۱۴] باشد.

۱.۰ پیش‌گیری از بن‌بست

در این روش، یک کار تنها در صورتی برای اجرا به کارگرها ارسال می‌شود که تمام خانه‌هایی که در آینده خواهد خواند در دسترس باشند و در این صورت، قفل همه آن‌ها به کارگر مورد نظر اختصاص داده می‌شود.

۲.۰ تشخیص بن‌بست

در این روش با تشکیل گراف نیازمندی‌های کارها باید پیش از بالا آوردن آن کار مطمئن شویم با اضافه‌شدن آن در گراف مورد نظر دور ایجاد نمی‌شود. توجه کنید که دور در گراف به معنای بن‌بست خواهد بود.



شکل ۲: resource allocation graph

[۱۳] deadlock prevention
[۱۴] deadlock detection

اگر با هرکدام از دو روش بالا بن‌بست تشخیص دادید، بسته به نوع الگورتیم زمان‌بندی به شکل زیر عمل کنید:

۱. در FCFS آن کار را به ته صف ببرید.

۲. در SJF اولین کار کوتاه که به بن‌بست نمی‌خورد را انجام دهید.

۳. در RR آن کار یک نوبت خود را از دست می‌دهد و تا نوبت بعدی‌اش صبر میکند.

توجه کنید در صورت بن‌بست نبودن نیز کاملا ممکن است بعضی کارها بخش زیادی یا تمام نوبت خود را در انتظار دریافت قفل بگذرانند.

ورودی و خروجی برنامه

ورودی

هنگام اجرای برنامه ورودی‌های زیر داده می‌شود:

۱. تعداد آرگومان اجرا

۲. آرگومان‌های اجرا (هرکدام در یک خط)

۳. پورت سرور اصلی

۴. تعداد کارگر

۵. نوع الگوریتم زمان‌بندی (FCFS, SJF, RR)

۶. طول بازه‌های زمانی (فقط برای الگوریتم RR)

۷. نحوه مواجهه با بن‌بست (PREVENT, DETECT, NONE)

۸. پورت سرور حافظه

۹. محتویات حافظه

۱۰. تعداد کارهای ورودی

۱۱. کارها (هر کدام در یک خط)

آرگومان‌های اجرا آرگومان‌هایی اند که برای اجرای یک برنامه جاوا نیاز اند، شامل آدرس جاوا، و مسیر سیستم برای آدرس‌دهی کلاس‌ها. ابتدا پارامترهای کلی اجرای برنامه داده می‌شوند. سپس پارامترهای مربوط به حافظه و نهایتا تعداد کارهایی که باید پردازش شوند (n). در n خط بعدی، هر کار iام به همان ترتیبی که داده شده است وارد صف اجرا می‌شود و سپس توسط الگوریتم‌های زمان‌بندی مدیریت می‌شود.

نحوه ورودی گرفتن هر کار به شکل زیر است:

$$time_1\ index_1\ \cdots\ time_n\ index_n$$

خروجی

پس از پایان هر کار گزارشی [۱۵] با قالب زیر توسط برنامه خروجی داده می‌شود. دقت کنید که ترتیب گزارش‌ها (با توجه به الگوریتم زمان‌بندی داده شده) باید درست باشد:

```
task [i] executed successfully with result [result]
```

نمونه ورودی و خروجی

ورودی ۱

```
3 // java common args
 java
 -classpath
 bin/oshw2/
 8000 // master port
 2 // number of workers
 FCFS //scheduling algorithm
 NONE // deadlock handling
 8001 // storage port
 1 2 3 0 // storage data
 2 // number of tasks
 50 0 50 1 150 2 // task 0
 0 1 100 2 100 3 // task 1
```

خروجی ۱

```
task 0 executed successfully with result 6
 task 1 executed successfully with result 5
```

ورودی ۲

```
3
 java
 -classpath
 bin/
 8000
 2
 RR
 100 // interrupt interval
 NONE
 8001
 0 1 2 3
 4
 100 0 100 0 100 0 100 3
 100 1 100 1 100 2 100 1 100 1
 100 2 100 3 100
 100 3
```

[۱۵] log

خروجی ۲

```
task 3 executed successfully with result 3
 task 2 executed successfully with result 5
 task 0 executed successfully with result 3
 task 1 executed successfully with result 6
```

ورودی ۳

```
8000
 2
 RR
 100
 NONE
 8001
 0 1 2 3
 4
 50 0
 100 1 100 1 50 1
 50 0 100 0 50 0
 100 2 50 3 100
```

خروجی ۳

```
task 0 executed successfully with result 0
 task 2 executed successfully with result 0
 task 1 executed successfully with result 3
 task 3 executed successfully with result 5
```

نحوه ارسال درخواست به حافظه

درخواست‌های حافظه به صورت زیراز طریق TCP ارسال می‌شوند.

نوشتن

write memory_index value

دریافت قفل و خواندن

obtain memory_index

آزاد کردن قفل

release memory_index

توجه کنید هر کار یک شناسه مخصوص خود دارد. یعنی در صورت دریافت قفل، تنها همان کار می‌توانند قفل را باز کند. همچنین اگر آن کار در حین انتظار برای قفل، جانشین شود و متوجه اختصاص قفل به او نشود (پیام تایید از سمت حافظه دریافت نکند)، در صورت اجرای دوباره، از نو اقدام برای دریافت قفل و خواندن خانه مورد نظر می‌کند.