استاد: **رامتین خسروی**
طراحان: **سعید زنگنه**، **محمد مریدی**، **صدف صادقیان**
مهلت تحویل: شنبه ۲۵ اسفند ۱۳۹۷، ساعت ۲۳:۵۵

هدف از این تمرین، آشنایی شما با مفاهیم اولیه طراحی شیءگرای[1] یک مسأله است. از آنجایی که استفاده از این مفاهیم، در پیاده‌سازی سایر تمارین این درس لازم است، پیشنهاد می‌شود به این پروژه زمان کافی را اختصاص دهید.

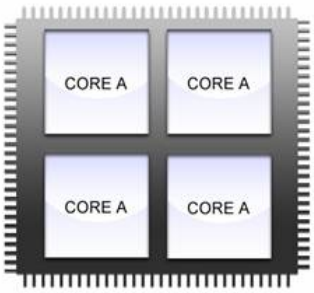
۱ مقدمه

همانطور که می‌دانید، پردازنده[2]ای با یک واحد پردازشی[3]، در یک زمان مشخص نمی‌تواند بیشتر از یک کار را انجام دهد؛ این عبارت بدین معنا است که وقتی دستورهای[4] مربوط به یک برنامه[5] بر روی پردازنده در حال اجرا است، این امکان وجود ندارد که دستورهای مربوط به برنامه‌ای دیگر نیز به طور همزمان بر روی همان پردازنده در حال اجرا باشد. اما همه ما می‌دانیم که در این پردازنده‌ها نیز، می‌توان چندین برنامه را به صورت هم‌روند[6] اجرا کرد. اجرای هم روند با اجرای موازی[7] متفاوت است. وقتی دو برنامه به صورت موازی اجرا می‌شوند، بدین معنا است که هر دو برنامه در زمانی مشخص بطور همزمان در حال اجرا می‌باشند. اما اصطلاح هم‌روندی، به معنی به اشتراک گذاری زمان[8] است که این احساس را بوجود می‌آورد که هر دو برنامه به طور همزمان اجرا می‌شوند. اشتراک‌گذاری زمان برای دو برنامه بدین صورت است که پردازنده به مدت زمان t_1 در اختیار برنامه اول قرار می‌گیرد و سپس به مدت زمان t_2 در اختیار برنامه دوم قرار می‌گیرد. پس از آن دوباره برنامه اول پردازنده را در اختیار گرفته و این چرخه تا پایان یکی از دو برنامه ادامه پیدا می‌کند. بدین ترتیب اگر t_1 و t_2 به اندازه کافی کوچک اختیار شوند، این احساس به کاربر القا می‌شود که هر دو برنامه به طور همزمان در حال اجرا می‌باشند.

۲ پیش زمینه

۱.۲ پردازنده‌ی چندهسته‌ای[9]

پردازنده چندهسته‌ای یک مولفه محاسباتی[10] است که از دو یا تعداد بیشتری واحد پردازشی به نام هسته[11] تشکیل شده است. یک هسته، وظیفه اجرای دستورهای یک برنامه را بر عهده دارد. در این پردازنده، این امکان وجود دارد که چندین دستور بطور همزمان بر روی هسته‌های مختلف و به صورت موازی اجرا شوند.



[1] Object-Oriented Design
[2] Processor
[3] Processing Unit
[4] Instructions
[5] Program
[6] Concurrent
[7] Parallel
[8] Time-Sharing
[9] Multicore Processor
[10] Computing Component
[11] Core

۲.۲ پردازه[۱۲]

یک پردازه نمونه‌ای از یک برنامه است که داخل حافظه[۱۳] بارگذاری[۱۴] شده است و در حال اجرا است. برای مثال زمانی که شما برنامه خود را کامپایل می‌کنید، کامپایلر یک فایل اجرایی متناظر با کد شما ایجاد می‌کند که شامل کد قابل فهم برای ماشین شما است. زمانی که شما این فایل اجرایی را با استفاده رابط خط فرمان[۱۵] (و یا به هر صورت دیگری) اجرا می‌کنید، سیستم‌عامل[۱۶] به برنامه شما، منابع[۱۷] لازم را اختصاص داده و آن را در حافظه اصلی بارگذاری می‌کند که به آن پردازه گفته می‌شود.

۳.۲ ریسه[۱۸]

در علم کامپیوتر، ریسه‌ای از اجرا، کوچکترین دنباله‌ای از دستورها است که می‌توانند به‌طور مجزا، مدیریت و برنامه‌ریزی شوند. در حالت کلی، مجموعه‌ای از ریسه‌ها، یک پردازه را تشکیل داده و این ریسه‌ها می‌توانند بصورت هم‌روند اجرا شوند.

۴.۲ برنامه‌ریز[۱۹]

برنامه‌ریزی[۲۰] در سطح ریسه‌ها، بدین معنا است که به هر کدام از ریسه‌ها، منابع لازم ــ که در برنامه‌ی شما، هسته‌های در حال برنامه‌ریزی است ــ برای انجام کامل کار خود اختصاص داده شود. رویکردهای گوناگونی برای نحوه اختصاص منابع به ریسه‌ها وجود دارد که در این تمرین، با یک نوع از آن که جلوتر توضیح داده خواهد شد، آشنا می‌شوید.

۵.۲ برنامه‌ریز دوره‌ای

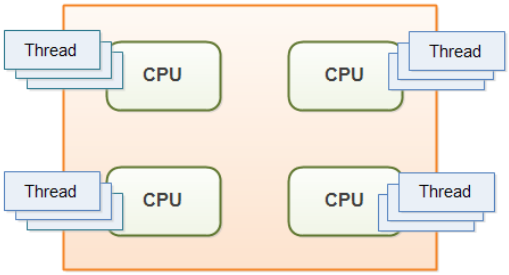
ایده اصلی برنامه‌ریز دوره‌ای [۲۱] بدین صورت است که برنامه‌ریز، به هر ریسه بازه زمانی معینی را اختصاص می‌دهد و منابع مورد نیاز آن را تأمین می‌کند. تخصیص منابع در ابتدای بازه انجام می‌شود و انتهای بازه نیز منابع پس گرفته می‌شود. به این بازه زمانی که منابع در اختیار یک ریسه است، یک برش زمانی[۲۲] گفته می‌شود. در این روش، به هر ریسه عددی تحت عنوان **تعداد برش‌های زمانی** که برای انجام کار خود نیاز دارد، نسبت داده می‌شود. این عدد، بیان‌کننده تعداد بازه‌های زمانی است که برنامه‌ریز، منابع را به این ریسه اختصاص می‌دهد. در پیاده‌سازی این روش، یک صف از ریسه‌ها به ازای هر هسته در نظر گرفته می‌شود. عملیات برنامه‌ریزی بدین صورت انجام می‌شود که منابع از ریسه‌ای که بر روی هسته در حال اجرا بوده است(ریسه ابتدای صف)، گرفته شده و در صورتی که تعداد برش‌های زمانی این ریسه به پایان نرسیده باشد، به انتهای صف اضافه می‌شود (در صورت اتمام برش‌های زمانی، از صف برنامه‌ریزی حذف می‌شود). سپس منابع لازم، به ریسه‌ای که در ابتدای صف وجود دارد تخصیص داده می‌شود.
برای مثال فرض کنید یک هسته داریم که در صف آن به ترتیب ریسه A با تعداد ۲ برش زمانی، ریسه B با ۱ برش زمانی و ریسه C با ۱ برش زمانی قرار دارند. در روش برنامه ریز دوره‌ای در این هسته ابتدا یک برش زمانی از ریسه A اجرا می‌شود، سپس این ریسه به انتهای صف منتقل می‌شود، در ادامه یک برش زمانی از ریسه B و سپس یک برش زمانی از ریسه C اجرا می‌شود. چون اجرا ریسه های B و C تمام شده است حال فقط ریسه A در صف باقی مانده‌است و در نهایت یک برش زمانی از ریسه A اجرا می‌شود تا دیگر ریسه‌ای در صف هسته باقی نماند.

---

[12] Process
[13] Memory
[14] Load
[15] Command-Line Interface
[16] Operating System
[17] Resources
[18] Process
[19] Scheduler
[20] Scheduling
[21] Round-Robin Scheduler
[22] Time Slice

۳ شرح تمرین

در این تمرین، شما به شبیه‌سازی یک برنامه‌ریز دوره‌ای می‌پردازید. این برنامه‌ریز باید این توانایی را داشته باشد که هنگامی که پردازه‌ای به سیستم اضافه شد، ریسه‌های مربوط به این پردازه را بین هسته‌های پردازنده، به صورت متعادل توزیع کند. دستورهایی که برنامه شما باید از طریق **جریان ورودی استاندارد**[۲۳] گرفته و پردازش کند در ادامه‌ی شرح تمرین آمده است.



۱.۳ اضافه‌کردن یک هسته به هسته‌های قابل برنامه‌ریزی

این دستور، یک هسته به هسته‌های قابل برنامه‌ریزی توسط برنامه‌ریز می‌افزاید. هر هسته دارای عددی یکتا، بعنوان شماره هسته است که برنامه‌ی شما باید آن را خودکار و به طور افزاینده از عدد ۱ به هسته‌ها اختصاص دهد.

دستور ورودی

```
add_core
```

خروجی

```
Core with core ID = <core_id> successfully added!
```

۲.۳ اضافه‌کردن یک پردازه

با اجرای این دستور، یک پردازه به پردازه‌های تحت برنامه ریزی اضافه شده و به آن شماره‌ای یکتا نسبت داده می‌شود. دومین آرگومان این دستور، تعداد ریسه‌های آن پردازه بوده و آرگومان‌های بعدی، به ازای هر ریسه تعداد برش‌های زمانی مورد نیاز آن ریسه را تعیین می‌کند. دقت کنید شماره‌های یکتایی که به ریسه‌های یک پردازه نسبت می‌دهید باید از ۱ شروع شوند. برای اختصاص دادن ریسه های یک پردازه به هسته‌ها، به ترتیب از اولین ریسه شروع کرده و هر ریسه را به اولین هسته‌ای که تعداد ریسه‌های کمتری در صف خود دارد اختصاص می‌دهیم.

دستور ورودی

```
add_process <number_of_process_threads> <number_of_time_slices_per_first_thread> ...
```

خروجی

```
Process with pid = <process_id> added!
```

[23] Standard Input Stream

به عنوان مثال با اجرای دستور زیر، پردازه‌ای که حاوی دو ریسه است به سامانه اضافه شده است. همچنین آرگومان‌های بعدی این دستور، تعداد برش‌های زمانی هر کدام از این ریسه‌ها را تعیین می‌کند که به ترتیب برابر با سه و یک است. با فرض این که این پردازه، اولین پردازه سامانه است، شماره پردازه برای آن یک در نظر گرفته می‌شود.

نمونه ورودی

```
add_process 2 3 1
```

نمونه خروجی

```
Process with pid = 1 added!
```

۳.۳ نمایش وضعیت هسته‌ها

کاربر با اجرای دستور زیر می‌تواند وضعیت تمام هسته‌های موجود در برنامه‌ریز و ریسه‌های درون صف هر هسته را مشاهده کند. در صورتی که صف یک هسته خالی بود نیز اطلاعاتش باید در خروجی بیاید.

دستور ورودی

```
show_cores_stat
```

خروجی به ازای هر هسته

```
Core number : <core_num>

(for each thread in the queue)

Process ID : <pid> - Thread ID : <tid>

Number of time slots : <number_of_remaining_time_slots>
```

خروجی نمونه

```
Core number : 1

Process ID : 4 - Thread ID : 2

Number of time slots : 3

Process ID : 3 - Thread ID : 2

Number of time slots : 1

Core number : 2

Core number : 3

Process ID : 3 - Thread ID : 1

Number of time slots : 1
```

۴.۳ فعالسازی هسته‌ها

در هر لحظه از اجرای برنامه، کاربر می‌تواند با وارد کردن این دستور، تمام هسته‌ها را برای مدت یک برش زمانی اجرا کند. اجرای هسته به این معنا است که یک ریسه را از ابتدای صف خود اجرا کرده و یک واحد از برش‌های زمانی آن می‌کاهد. بعد از بررسی غیر صفر بودن برش‌های زمانی آن، ریسه را به انتهای صف اضافه می‌کند (توجه شود که در صورت صفر شدن این مقدار، ریسه باید از صف هسته حذف شود). خروجی این دستور نیز، شماره پردازه و شماره ریسه‌هایی است که در هر هسته اجرا شده است. اگر در یک برش زمانی صف یک هسته خالی بود نام این هسته **نباید** در خروجی آن برش زمانی باشد.

دستور ورودی

```
run_cores
```

خروجی به ازای هر هسته

```
Core number : <core_num>

Process ID : <pid> - Thread ID : <tid>
```

نمونه خروجی

```
Core number : 1

Process ID : 4 - Thread ID : 2

Core number : 2

Process ID : 3 - Thread ID : 1
```

۵.۳ اتمام کار تمام ریسه‌های موجود

در هر لحظه از اجرای برنامه، کاربر می‌تواند با وارد کردن این دستور، تمام هسته‌ها را تا اتمام کار تمامی ریسه‌های موجود در صف‌های هسته‌ها فعال کند. خروجی این دستور نیز به ازای هر برش زمانی، مانند دستور فعالسازی یک هسته است. اگر در یک برش زمانی صف یک هسته خالی بود نام این هسته نباید در خروجی آن برش زمانی باشد. عدد برش‌های زمانی نیز در هر بار اجرای این دستور از عدد ۱ شروع می‌شود.

دستور ورودی

```
finish_tasks
```

خروجی

```
Time Slice : 1

Core number : <core_num>

Process ID : <pid> - Thread ID : <tid>

.

.

.

Time Slice : 2

Core number : <core_num>

Process ID : <pid> - Thread ID : <tid>

.

.

.

Time Slice : n

Core number : <core_num>

Process ID : <pid> - Thread ID : <tid>

.

.

.
```

۴ نکات تکمیلی

در این تمرین، اختصاص عددی یکتا به یک موجودیت[۲۴]، بدین صورت است که مقدار اولیه عددی که به موجودیت‌ها تخصیص داده می‌شود، برابر با **یک** بوده و با اضافه شدن هر موجودیت، یک واحد به این شمارنده افزوده می‌شود.

[24] Entity

۵ نحوه‌ی تحویل

- تمام فایل‌های برنامه‌ی خود را در قالب یک فایل با پسوند **zip** و نام A4-SID.zip در صفحه‌ی CECM درس بارگذاری کنید که SID شماره‌ی دانشجویی شماست؛ برای مثال اگر شماره‌ی دانشجویی شما ۸۱۰۱۹۷۹۹۹ باشد، نام پرونده‌ی شما باید A4-810197999.zip باشد.
- برای اطمینان از درستی قالب فایل آپلودی خود، اسکریپت `test.sh` و دو فایل `sample.in` و `sample.out` را که در صفحه درس بارگذاری شده‌اند را در کنار فایل خود قرار دهید و دستور `./test.sh SID` را اجرا کنید. دقت کنید خروجی این دستور بعد از عبارت `##### DIFF #####` باید اختلاف خروجی برنامه شما با خروجی مورد انتظار از برنامه را نشان دهد که درصورت پیاده‌سازی صحیح، نباید خروجی‌ای نشان داده شود. هر خروجی دیگری غیرقابل قبول است.
- برنامه‌ی شما باید در سیستم‌عامل لینوکس و با مترجم g++ با استاندارد c++11 ترجمه و در زمان معقول برای ورودی‌های آزمون اجرا شود.
- از صحت قالب[۲۵] ورودی‌ها و خروجی‌های برنامه‌ی خود مطمئن شوید.
- با توجه با این که این اولین پروژه‌ی Multifile شماست به این نکته توجه داشته باشید که برای ساخت[۲۶] برنامه‌ی خود حتما از Makefile استفاده کنید و فایل اجرایی نهایی شما اسم Scheduler.out را داشته باشد. در صورت عدم رعایت این نکات نمره‌ای از آزمون‌های خودکار ورودی و خروجی به شما تعلق نخواهد گرفت.
- طراحی درست، رعایت سبک برنامه‌نویسی درست و تمیز بودن کد برنامه‌ی شما در نمره‌ی تمرین تأثیر زیادی دارد.
- هدف این تمرین یادگیری شماست. لطفاً تمرین را خودتان انجام دهید. در صورت کشف تقلب مطابق قوانین درس با آن برخورد خواهد شد.

[25] Format
[26] Build