# ساخت کلونی مورچگان

> مورچگان را چو بود اتفاق شیر ژیان را بدرانند پوست

## ساختار کلی

در این تمرین می‌خواهیم سیستمی برای مدیریت انجام کارها طراحی کنیم. ساختار کلی این سیستم به این شکل است که شامل یک سرور اصلی، چندین سرور فرعی و یک سرور مخزن می‌باشد (در ادامه توضیحات کامل‌تر ارائه می‌شود).

در این سیستم، کاربران می‌توانند با استفاده از یک سوکت (socket) به سرور اصلی درخواست انجام کار بدهند و آن را به لیست کارهای سرور اصلی اضافه کنند. سرور اصلی نیز با مشاهده‌ی لیست کارهای انجام نشده، باید آن‌ها را میان سرورهای فرعی تقسیم کند و انجام کارها را به سرورهای فرعی بسپارد و در نهایت نتیجه را به کاربر بازگرداند.

ارتباط تمامی اجزای سیستم با یکدیگر باید از طریق شبکه باشد.

در ادامه، توضیحات دقیق‌تری درباره‌ی هرکدام از بخش‌ها ارائه می‌شود.

## آغاز برنامه و سرور اصلی

اجرای برنامه با دادن ورودی به سرور اصلی شروع می‌شود. تعداد سرورهای فرعی ($n$) و تعداد برنامه‌ها ($k$) و سپس آدرس برنامه‌های $a_0, a_1, \ldots, a_{k-1}$ به عنوان ورودی به سرور اصلی داده می‌شوند (هرکدام از $a_i$ها یکی از برنامه‌ها می‌باشند). سرور اصلی پس از خواندن ورودی باید $n$ سرور فرعی را به عنوان یک پردازه (process) جدید ایجاد نماید و pid پردازه‌های ساخته شده را گزارش نماید و همچنین یک پردازه که بیانگر سرور مخزن می‌باشد را بسازد و pid آن را گزارش نماید.

## برنامه‌ها و زنجیره‌های اجرا

هر برنامه یک وزن دارد و وزن برنامه‌ی $a_i$، برابر با $w_i$ می‌باشد. کارکرد هر برنامه به این صورت است که در ابتدای اجرای خود یک عدد را به عنوان ورودی می‌گیرد و پس از گذشت زمانی نامشخص (زمانی که

برای اجرا صرف می‌کند)، عددی را به عنوان خروجی اعلام می‌کند.

یک زنجیره‌ی اجرا به صورت $i_m\ x \mid i_{m-1} \mid \ldots \mid i_2 \mid i_1$ می‌باشد و خروجی آن به صورت بازگشتی، برابر با خروجی زنجیره‌ی $i_{m-1}\ y \mid \ldots \mid i_2 \mid i_1$ می‌باشد که $y$ نتیجه‌ی اجرای برنامه‌ی $a_{i_m}$ با ورودی $x$ می‌باشد.

برای مثال اگر دو برنامه‌ی $a_1$ و $a_2$ داشته باشیم و خروجی $a_1$ به ازای ورودی $x$ برابر با همان $x$ باشد و خروجی $a_2$ به ازای ورودی $x$ برابر با $x-1$ باشد، در این صورت نتیجه‌ی زنجیره‌ی اجرای $2|2|1\ 3$ برابر با ۱ می‌باشد (ابتدا ورودی ۳ به برنامه‌ی $a_1$ داده می‌شود و عدد ۳ برگردانده می‌شود. سپس عدد ۳ به برنامه‌ی $a_2$ داده می‌شود و ۲ برگردانده می‌شود و در نهایت عدد ۲ به برنامه‌ی $a_2$ داده می‌شود و خروجی آن ۱ می‌شود).

## سرور فرعی

یک سرور فرعی به صورت همزمان می‌تواند یک یا چند اجرا از هر یک از برنامه‌های گفته شده را به صورت همزمان به عنوان زیرپردازه‌های (subprocess) خود اجرا کند به شرطی که مجموع وزن برنامه‌های در حال اجرا در هر لحظه حداکثر $W$ باشد.

سرور فرعی می‌تواند تنها با گرفتن هربار درخواست از سرور اصلی که شامل نام یک برنامه و یک ورودی است، برنامه مورد نظر را با آن ورودی اجرا کرده و خروجی را به سرور اصلی بلافاصله پس از اتمام اجرای آن برنامه ارسال نماید.

## کاربر و نحوه‌ی ارسال درخواست

هر کاربر می‌تواند با استفاده از یک socket به سرور اصلی متصل شده و یک زنجیره‌ی اجرا به سرور اصلی ارسال کند و سرور اصلی با دریافت این زنجیره‌ی اجرا، باید حاصل آن را بدست بیاورد و بلافاصله نتیجه را به کاربر برگرداند. توجه کنید که هر کاربر حداکثر یک زنجیره‌ی اجرا را ارسال می‌کند.

**سرور اصلی اجازه اجرای برنامه‌ها را ندارد** ولی همانطور که گفته شد می‌تواند با ارسال درخواست به یک سرور فرعی می‌تواند نتیجه اجرای یک برنامه روی ورودی داده شده را بدست آورد.

## سرور مخزن (cache)

سرور مخزن محلی است که نتیجه‌ی اجراهای پیشین در آن نگهداری می‌شود تا اگر در آینده درخواستی مشابه آمده بود، سریعا پاسخ آن را برگردانیم.

هرگاه جواب اجرای یک برنامه روی یک ورودی محاسبه شد نیاز است که این نتیجه بر روی سرور مخزن ذخیره شود تا از اجرای دوباره این برنامه با این ورودی جلوگیری شود. سرور اصلی در طول زمان اجرای خود اجازه دارد که هر برنامه را با هر ورودی حداکثر یکبار به سروری فرعی جهت محاسبه خروجی آن ارسال نماید. نتیجه هر یک از این اجراها باید در سرور مخزن نگهداری شود تا اگر در آینده نیاز به جواب یک برنامه روی ورودی‌ای را داشتیم که قبلا یکبار محاسبه کرده بودیم، جواب آن را مستقیم و بدون معطلی از سرور مخزن بخوانیم.

همچنین توجه کنید که اگر برنامه‌ی $i$ با ورودی $x$ بر روی یکی از سرورهای فرعی در حال اجرا باشد، در صورتی که چند زنجیره‌ی اجرا همزمان به نتیجه‌ی اجرای برنامه‌ی $i$ام با ورودی $x$ نیاز داشته باشند، اولین درخواست بر روی یکی از سرورهای فرعی می‌رود و بقیه‌ی زنجیره‌های اجرا باید صبرکنند تا از نتیجه‌ی آن اجرا استفاده کنند (یعنی فقط یک بار ورودی $x$ بر روی برنامه‌ی $i$ام اجرا می‌شود).

## نحوه‌ی اجرای دستورات کاربران

سرور اصلی برای اجرای دستورات کاربران، یک صف اجرا دارد که زنجیره‌های اجرا در آن هستند و بر اساس زمانی که کاربر درخواستش را ارسال کرده است مرتب شده‌اند (یعنی همیشه قدیمی‌ترین درخواست انجام نشده، در ابتدای صف می‌باشد).

**در هر لحظه از زمان،** در صورتی که یک زنجیر اجرایی در صف اجرا وجود داشته باشد که نتیجه‌ی بخشی از اجرایش در سرور مخزن (cache) وجود داشته باشد، بدون نوبت باید با استفاده از سرور مخزن، آن زنجیر را به روز رسانی کنیم. به طور دقیق‌تر، اگر زنجیر اجرایی به شکل $i_m\ x \mid i_{m-1} \mid \ldots \mid i_2 \mid i_1$ در صف اجرا باشد و نتیجه‌ی اجرای ورودی $x$ بر روی برنامه‌ی $a_{i_m}$ نیز در سرور مخزن موجود باشد، باید این زنجیر اجرا را با زنجیر $i_{m-1}\ y \mid \ldots \mid i_2 \mid i_1$ جابجا کنیم که $y$ همان نتیجه‌ی اجرای $x$ بر روی برنامه‌ی $a_{i_m}$ می‌باشد.

سرور اصلی برای محاسبه‌ی یک زنجیره‌ی اجرا نیاز به ارسال برنامه‌های زنجیره به صورت تکی تکی به سرورهای فرعی را دارد. در هر لحظه از زمان، سرور اصلی در صورت خالی نبودن صف اجرا، پر اولویت‌ترین زنجیر اجرا را در نظر می‌گیرد و درخواست اجرای برنامه با ورودی مشخص شده را به سرور فرعی‌ای می‌دهد که مجموع وزن برنامه‌های در حال اجرا بر روی آن در آن لحظه کمینه باشد و همچنین با ارسال

این درخواست، مجموع وزن کارهای در حال اجرا در آن سرور فرعی بیشتر از $W$ نشود. در صورتی که چندین سرور فرعی با چنین شرایطی وجود داشتند، به دلخواه به یکی از آن‌ها این درخواست را می‌فرستد. توجه کنید وقتی این درخواست به یکی از سرورهای فرعی فرستاده شود، آن زنجیر اجرا به صورت موقت از صف اجرا حذف می‌شود و پس از اینکه نتیجه‌ی اجرای ورودی مورد نظر بر روی برنامه‌ی مورد نظر محاسبه شد، دوباره این زنجیر اجرا به صف اجرا برمی‌گردد تا بقیه‌ی برنامه‌هایش اجرا شود (توجه کنید که هر زنجیر اجرا، متشکل از چند برنامه می‌باشد که باید اجرا شوند و هرکدام از این برنامه‌ها ممکن است بر روی یک سرور فرعی متفاوت اجرا شود). همچنین در صورتی که هیچ سرور فرعی‌ای قادر به اجرای این دستور نباشد، سرور اصلی هیچ‌کاری نمی‌کند و به سراغ زنجیرهای اجرای بعدی **نمی‌رود** (در این شرایط سرور اصلی باید صبرکند تا یکی از سرورهای فرعی ظرفیتش خالی شود تا بتواند این برنامه را اجرا کند یا اگر این برنامه قبلا در حال اجرا بود، اجرای آن تمام شود).

همچنین اگر در یک لحظه از زمان بتوانیم پر اولویت‌ترین زنجیر اجرا را به یکی از سرورهای فرعی بدهیم تا آن را اجرا کنند، باید اینکار را بلافاصله انجام دهیم.

همچنین اگر نتیجه جواب اجرای برنامه‌ی اول یک زنجیره اجرا در سرور مخزن موجود باشد باید بلافاصله نتیجه این اجرا بر روی زنجیره اعمال شود و زنجیره مورد نظر کوتاه تر شود. اگر دو زنجیره اجرا باشند که مقدار ورودی و برنامه‌ی اول آن‌ها یکی بود، باید یکی از آنها در سرورهای فرعی اجرا شوند و دیگری نتیجه اجرا را از طریق سرور مخزن دریافت کند.

## بسته شدن برنامه

در برخی مواقع ممکن است یکی از سرورهای فرعی و یا سرور اصلی بسته شوند.

### بسته شدن سرور اصلی

با بسته شدن پردازه‌ی سرور اصلی توسط سیستم‌عامل باید تمام پردازه‌های مربوط به بخش‌های دیگر که پردازه مربوط به سرور اصلی یا ساب پردازه‌های آن اجرا کرده‌اند بسته‌شود.

### بسته شدن سرور فرعی

با بسته شدن پردازه‌ی مربوط به یک سرور فرعی باید تمام کارهایی که در حال انجامش می‌باشد نیز متوقف شوند. سرور اصلی موظف است تا هنگامی که یکی از سرورهای فرعی بسته شدند، زنجیره‌های

اجرایی که در حال اجرا بودند را به صف اجرا برگرداند و سپس دوباره یک سرور فرعی جدید جایگزین سرور فرعی بسته شده‌ی قبلی بسازد و pid آن را هم گزارش کند.

## ورودی

هنگام اجرا شدن سرور اصلی اطلاعات زیر در این قالب از طریق استریم `System.in` داده می‌شود.

```
[port master]
[number of workers]
[w]
[number of common args]
[common args] (each in new line)
[number of programs]
[class name] [weight] (each in new line)
```

`port master` : شماره‌ی پورتی است که در آن کاربر درخواست‌های خود را می‌فرستد.

`number of workers` : تعداد سرورهای فرعی است.

`w` : محدودیت کلی مجموع وزن برنامه‌ها بر روی هر سرورهای فرعی

`common args` : آرگومان‌هایی که برای اجرای یک برنامه‌ی جاوا نیاز است. به طور کلی ممکن است اجرا کردن برنامه‌ی جاوا در سیستم‌های مختلف دستورات مختلفی داشته باشد. این آرگومان‌ها قسمت ابتدایی اجرا کردن یک برنامه‌ی جاوا است و با اضافه کردن نام کامل یک کلاس (به همراه نام پکیج) می‌توانید متود main داخل آن کلاس را به عنوان یک پردازه‌ی جدا اجرا کنید.

`number of programs` : تعداد برنامه‌هایی که باید توسط سرورهای فرعی اجرا شود.

`class name` : نام کامل کلاس مربوط به آن برنامه.

`weight` : وزن مربوط به آن برنامه.

یک مثال از ورودی بالا به صورت زیر است:

```
16543
2
20
```

```
3
java
-classpath
out/production/OS-HW1/
2
os.hw1.programs.Program1 2
os.hw1.programs.Program2 5
```

که به در این مثال برای اجرا کردن برنامه‌ی اول باید دستور زیر اجرا شود:

```
java -classpath out/production/OS-HW1/ os.hw1.programs.Program1
```

همچنین کاربرها با باز کردن یک سوکت بر روی پورت مربوط به سرور اصلی در قالب زیر درخواست‌های خود را به سرور اصلی می‌دهند و منتظر جواب عملیات خود می‌مانند. درخواست‌ها به شکل زیر می‌باشد:

```
[i_1]|[i_2]|...|[i_m] input
```

که بیانگر یک زنجیره اجرا است. شماره‌ی برنامه‌ها از یک شروع می‌شود.تضمین می‌شود `input` عدد صحیح است. یک مثال از این درخواست به صورت زیر است:

```
1|2|1 12
```

## خروجی

به طور کلی باید اتفاقاتی را در خروجی `System.out` لاگ کنید. ممکن است در ادامه و پس از اضافه شدن تست‌ها تعداد و نوع این اتفاقات بیشتر شود. در حال حاضر باید شروع شدن یا بسته شدن پردازه‌های مربوط به سرور اصلی، سرورهای فرعی و سرور کش را در فرمت زیر لاگ کنید:

```
[component name] [worker id (only for workers)] [stop|start] [pid] [port]
```

`component name` : نوع سرور را مشخص می‌کند. برای سرور اصلی از عبارت `master` برای سرورهای فرعی از عبارت `worker` و برای سرور مخزن از عبارت `cache` استفاده کنید.

`worker id` شماره‌ی سرور فرعی است. این شماره‌ها از صفر شروع می‌شوند. این شماره برای سرور اصلی و مخزن لازم نیست.

`stop|start` : شروع شدن یا بسته شدن پردازه را مشخص می‌کند.

`pid` : آی‌دی پردازه مورد نظر است.

`port` : پورت مربوط به پردازه مورد نظر.

چند نمونه از این لاگ‌ها به صورت زیر است:

```
master start 158361 16543
cache start 158395 16542
worker 0 start 158398 16544
worker 1 start 158410 16545
worker 0 stop 158398 16544
worker 0 start 158538 16544
```

همچنین پاسخ درخواست‌های کاربرها را باید در قالب یک عدد در یک خط در خروجی سوکت باز شده توسط کاربر باید قرار دهید. حتما قبل از بستن سوکت یک کاراکتر `\n` نیز در خروجی سوکت چاپ کنید.

## آنچه باید آپلود کنید

این بخش پس از اضافه شدن تست‌ها کامل می‌شود.

## نکات بیشتر

- نام کلاسی متود `main` مربوط به سرور اصلی قرار دارد را `MasterMain` قرار دهید و در پکیج `os.hw1.master` قرار دهید.
- هاست مربوط به همه‌ی سرورها `localhost` می‌باشد ولی برای گرفتن هاست از متود `InetAddress.getLocalHost()` استفاده کنید.