به نام آن که جان را فکرت آموخت

دانشگاه صنعتی شریف | **تمرین برنامه‌نویسی صفرم*** | دانشکده‌ی مهندسی کامپیوتر

مدرس: مهدی خرازی | **شبکه‌های رایانه‌ای** | زمستان ۱۳۹۴

## ۱. هدف‌ها

- آشنایی با سامانه‌عامل UNIX،
- آشنایی با Unix Socket API،
- آشنایی با پروتکل HTTP،
- آشنایی با متوزان‌کننده‌ی بار،
- نحوه‌ی ارسال صحیح تمرین.

## ۲. کلیات

در تمرین صفرمِ درس شبکه، دانشجویان پس از آشنایی با برنامه‌نویسی سوکت، برنامه‌ی سوکت ساده‌ای در سمت کارگزار[۱] توسعه می‌دهند. در بخش بعدی تمرین صفرم، یک ماژول متوازن‌کننده‌ی بار[۲] سمتِ کارگزار توسط دانشجویان توسعه پیدا می‌کند که هدف از طراحی این ماژول، مدیریت درخواست‌های ارسالی به چندین کارگزار وب با سرویس مشابه است. مسئله‌ی پاسخ‌گوییِ قابل اعتماد به تعداد زیادی کارخواه[۳] در شبکه یکی از مهم‌ترین چالش‌های ارائه‌دهندگان سرویس است. این مسأله خصوصاً در وب مورد توجه بوده و لازم است برای سرویس‌هایی با درخواست بالا، همه‌ی ترافیک وب پس از دریافت با توزیع مناسبی بین گره‌های مناسب (در این‌جا کارگزارهای وب[۴]) توزیع شود.

*طراحی تمرین توسطِ سولماز سلیمی، زینب ساسان، پیمان عزتی، رضا میرعسگری، پارسوآ خورسند

[۱] Server
[۲] Load Balancer
[۳] Client
[۴] Web Server

## ۳. مقدمه‌ای بر HTTP

پروتکل انتقال فرامتن HTTP[۵] پروتکلی در لایه‌ی کاربرد است که هسته‌ی اصلی وب را تشکیل می‌دهد. این پروتکل در دو برنامه‌ی سمت کارخواه (مرورگر وب) و سمت کارگزار پیاده‌سازی می‌شود. در عمل این پروتکل مشخص می‌کند چگونه مرورگر وب از یک کارگزار وب، منابعی را درخواست می‌کند و کارگزار چگونه پاسخ می‌دهد.
ارتباطات پروتکل HTTP از راه تراکنش‌ها ایجاد می‌شود. یک تراکنش شامل درخواستی است که از سمت کارخواه به سمت کارگزار ارسال می‌شود و سپس در مرحله‌ی بعدی همین کارخواه پاسخِ کارگزار را دریافت و آن را می‌خواند. پیام‌های درخواست و پاسخ از یک قالب مشترک پایه استفاده می‌کنند:

- یک خط آغازی (مشخص کننده‌ی درخواست یا پاسخ)
- صفر یا چند خط سرآیند
- یک خط خالی (CRLF)
- بدنه‌ی پیام.

برای اغلب تراکنش‌های HTTP، چهار گام معمول زیر طی می‌شود:

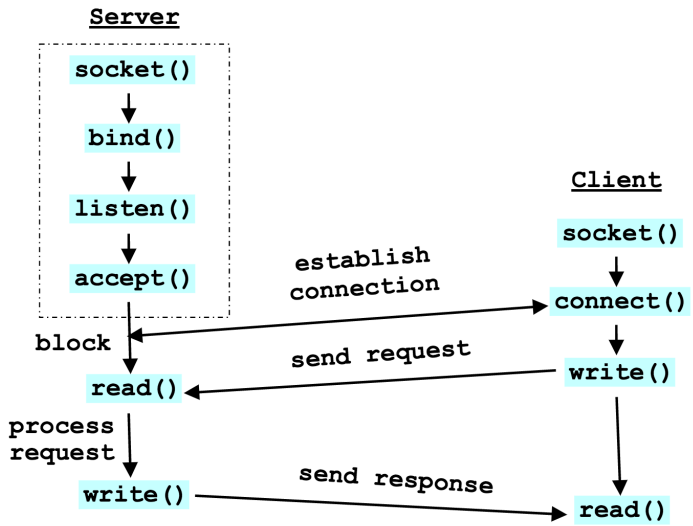
۱. کارخواه یک ارتباط با کارگزار ایجاد می‌کند.
۲. کارخواه از راه ارسال یک خط متن به کارگزار درخواست خود را ارسال می‌کند. **خط درخواست** شامل یک تابع HTTP مانند GET یا POST، یک آدرس درخواست و نسخه‌ای پروتکل HTTP مورد استفاده است. در انتها این خطِ درخواست با خطوط سرآیند و بدنه‌ی پیام همراه می‌شود، که معمولاً بدنه‌ی پیام درخواست HTTP خالی است.
۳. کارگزار پیام پاسخ را به سمت کارخواه ارسال می‌کند. خط اول پیام پاسخ نشان‌دهنده‌ی وضعیت[۶] است و مشخص‌کننده‌ی موفقیت‌آمیز بودن یا نبودن آن است. بخش‌های بعدی مانند قالب پیام درخواست است، با این تفاوت که بخش بدنه‌ی پیام خالی نیست و شامل پاسخی است که کارخواه درخواست کرده است.
۴. به محض اینکه پاسخ کارگزار به سمت کارخواه ارسال شد، ارتباط از سمت کارگزار بسته می‌شود.

[۵] Hypertext Transfer Protocol
[۶] Status

# ۴. بخش اول: برنامه‌نویسی سوکت

## ۱.۴ مقدمه‌ای بر ارتباطات سوکت

سوکت‌ها امکان برقراری ارتباط بین فرآیندهای مختلف بر روی یک سامانه یا سامانه‌های مختلف را فراهم می‌کنند. یک سوکت یونیکس در چارچوب کاربردی کارگزار–کارخواه به کار گرفته می‌شود. سوکت‌ها نقاط ارتباطی برای تبادل اطلاعات در سامانه‌های رایانه‌ای مختلف هستند. فرآیندها در لایه‌ی کاربرد، داده را از طریق سوکت‌ها به لایه‌ی انتقال ارسال می‌کنند. در معماری کارگزار–کارخواه، فرآیند برقراری ارتباط بین کارگزار و کارخواه در قالب پروتکل TCP، مانند شکل زیر است:




شکل ۱: فرآیندبرقراری ارتباط بین کارگزار و کارخواه در قالب پروتکل TCP [مرجع]

فرآیندها به شکل زیر خواهد بود:

- سمت کارخواه
  ۱. ایجاد سوکت
  ۲. اتصال به آدرس کارگزار هدف
  ۳. ارسال و دریافت داده
- سمت کارگزار
  ۱. ایجاد سوکت
  ۲. انتساب آدرس مشخص شده برای کارگزار به سوکت ایجادشده
  ۳. گوش دادن به درخواست‌ها

۴. پذیرش درخواست ارتباط از سمت کارخواه

۵. ارسال و دریافت داده

### ۲.۴. روش کار

هدف از این تمرین نوشتن برنامه‌ی سوکت سمت کارگزار است؛ بطوری‌که درخواست‌های کارخواه را دریافت کرده و از سرآیند درخواست‌های HTTP <مقدار، کلید> مربوط به کوکی را استخراج و ذخیره نماید. برای نوشتن این برنامه می‌توانید از زبان برنامه‌نویسی جاوا، C و یا پایتون استفاده کنید.

### ۳.۴. پیاده سازی

در طراحی این برنامه ابتدا سوکتی با آدرس localhost ایجاد می‌کنید که بر روی پورت شماره ۸۰ به درخواست‌ها گوش می‌کند. به محض دریافت درخواست، آن را قبول کرده و سپس بسته‌ی HTTP را برای پیدا کردن <مقدار، کلید> مربوط به کوکی تجزیه می‌کند. اگر بسته‌ی درخواستِ دریافتی حاوی کوکی باشد، آن را در پرونده‌ای با نام `Cookie` ذخیره می‌کند.

- در این تمرین، مرورگر وب شما نقش کارخواه را داشته و باید درخواست‌ها از این کارخواه به سمت کارگزاری که طراحی کرده‌اید ارسال شود. در تنظیمات مرورگر آدرس و پورت بخش کارگزار پیش‌کار[۷] را به localhost و ۸۰ تغییر دهید. پس از اتمام تمرین، تنظیمات مرورگر را به حالت قبل برگردانید!
  همچنین باید توجه داشته باشید که به دلیل محدودیت در قابلیت‌های کارگزاری که طراحی کرده‌اید، درخواست‌های ارسالی از مرورگر به سمت کارگزار وب اصلی ارسال نخواهد شد و شما در پاسخ به درخواست، پیغام بازنشانی ارتباط را در مرورگر خود مشاهده خواهید کرد.
- برای تست برنامه‌ای که طراحی کرده‌اید، در بخش تنظیات مرورگر، وب‌گاه‌هایی که در مرورگر شما کوکی ایجاد کرده را مشاهده نموده و در مرورگر آدرس این وب‌گاه‌ها را جستجو کنید. نتایج ارسال این درخواست‌ها به کارگزار شما (کوکی استخراج شده) باید در ترمینال نمایش داده شده و در پرونده‌ی مربوطه ذخیره شود.

## ۵. بخش دوم: متوازن‌کننده‌ی بار

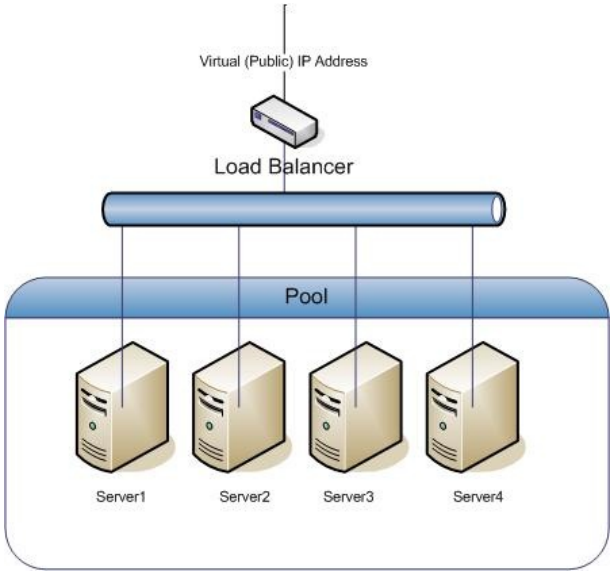
### ۱.۵. مقدمه‌ای بر متوازن‌کننده‌ی بار

وب‌گاه‌های مشهور نمی‌توانند تنها به یک کارگزار وب قدرتمند متکی باشند. با استفاده از معماری کارگزارهای توزیع‌شده میزان احتمال دسترسی و انعطاف‌پذیری بیشتر می‌شود. در معماری کارگزاری توزیع‌شده درخواست‌های

[۷] Proxy Server

کاربران بین کارگزارهای موجود زمان‌بندی خواهد شد.

**متوازن‌کننده‌ی بار:** متوازن‌کننده‌ی بار می‌تواند کارایی و قابلیت اطمینان را با توزیع کردن بارِ کاری بین چندین کارگزار فرآهم آورد. متوازن‌کننده‌ی بار بین چندین نمونه از کارگزاهای مربوط به یک سرویس یکپارچه است که برای افزایش بهره‌وری از منابع، افزایش گذردهی و کاهش تأخیر مورد استفاده قرار می‌گیرد.




شکل ۲: مثالی از به‌کارگیری یک متوازن‌کننده‌ی بار با ۴ کارگزارِ سرویس‌دهنده

گفتنی است متوازن‌کننده‌ی بار به عنوان یک موجودیت مستقل عمل می‌کند و نمونه‌های متن‌باز زیادی از متوازن‌کننده‌های بار موجود هستند؛ به همین دلیل وابستگی خاصی برای آن‌ها تعریف نمی‌شود.

### ۱.۱.۵. الگوریتم‌های مورد استفاده در متوازن‌کننده‌ی بار

- چرخش عادلانه[۸]: درخواست‌های مربوط به کارگزارها به‌صورت گردشی بین آن‌ها توزیع می‌شود.
- کمترین ارتباط[۹]: درخواست بعدی به کارگزاری با کمترین تعداد ارتباط فعال اختصاص داده می‌شود. در شرایطی که برخی از درخواست‌ها مدت زمان طولانی‌تری برای پاسخ‌دهی نیاز دارند، این روش کنترل عادلانه بر روی بار کاری را فرآهم می‌آورد.
- درهم‌سازی آدرس آی‌پی[۱۰]: از یک تابع درهم‌سازی برای مشخص کردن کارگزار مسئول برای هر درخواست استفاده می‌شود. (بر اساس آدرس شبکه‌ی کارخواه) در شرایطی که نیاز است تا درخواست هر کارخواهِ مشخص،

[۸] Round Robin
[۹] least-connected
[۱۰] ip-hash

به یک کارگزار خاص تحویل داده شود و به عبارتی ویژگی اتصال وجود داشته باشد، از این روش متوزان‌کننده‌ی بار استفاده می‌شود. در این روش آدرس شبکه‌ی کارخواه به عنوان کلید درهم‌سازی برای مشخص کردن کارگزار پاسخ‌دهنده مورد استفاده قرار می‌گیرد.

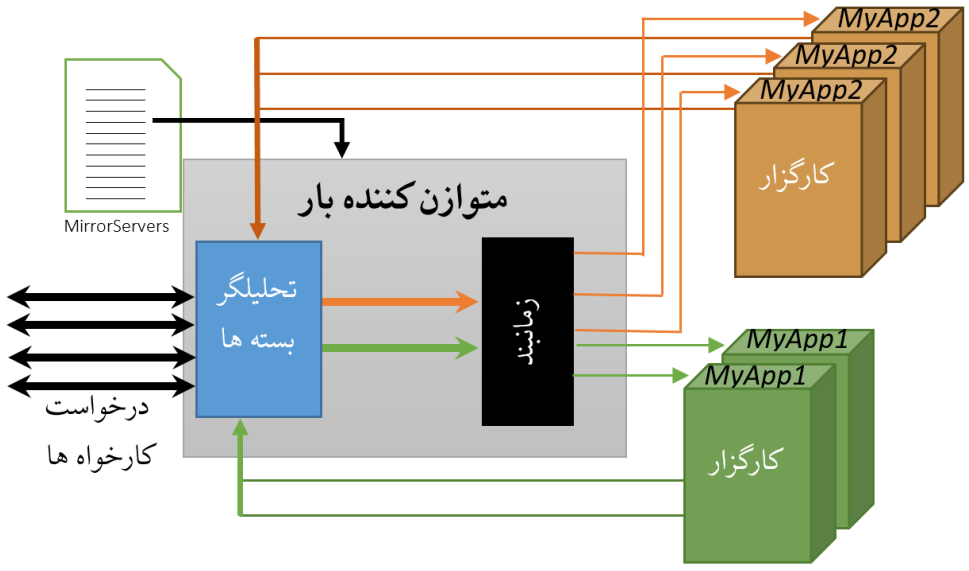
- متوازن‌کننده‌ی بار وزن‌دار[۱۱]: می‌توان برای هر کارگزار، وزنی تعریف کرد تا هر یک از روش‌های ذکر شده بر اساس این وزن‌دهی اجرا شود. کارگزارهایی با بالاترین وزن، به هنگام تصمیم‌گیری برای توزیع درخواست‌ها، بیشترین درخواست را دریافت خواهند کرد.

# ۶. توضیح تمرینِ متوازن‌کننده‌ی بار

در این بخش روند طراحی ماژول متوازن‌کننده‌ی بار توضیح داده شده است.

## ۱.۶ روش کار

در این بخش از تمرین، شما باید ماژولی برای متوازن‌کننده‌ی بار بنویسید که قابلیت دریافت درخواست از کارخواه و توزیع آن بین کارگزارهای وبِ مجازی را داشته باشد و پاسخ را به سمت کارخواه برگرداند. توسعه‌ی کد متوازن‌کننده‌ی بار مانند بخش اول تمرین می‌تواند توسط زبان‌های جاوا، C یا پایتون صورت بگیرد.
در هنگام اجرای ماژول متوزان‌کننده‌ی بار اولین کاری که صورت می‌گیرد، ایجاد سوکت و گوش‌دادن به درخواست‌های ارسالی از کاربران مختلف خواهد بود.




شکل ۳: فرآیندهای مورد انتظار متوازن کننده بار در تمرین دوم

[۱۱] weighted Load balancer

این ماژول یک پرونده‌ی پیکربندی به شکل زیر را می‌خواند تا بتواند آدرس‌های وب و آدرس‌های کارگزارهای مرتبط را شناسایی نماید.

```
loadbalancer {
    upstream myapp1 {
        server 192.168.200.2:13030;
        server 192.168.200.3:13030;
        server 192.168.200.4:13030;
        }

    upstream myapp2 {
        server 192.168.80.10:13030;
        server 192.168.80.11:9999;
        server 192.168.80.12:13030;
        }
}
```

این پرونده با نام `MirrorServers` بدون پسوند در کنار پرونده‌ی اجرایی شما قرار خواهد گرفت.

## ۲.۶. بسته‌های دریافتی کارخواه‌ها

برنامه‌ی شما روی پورتی که از ورودی می‌گیرید، درخواست اتصال سوکت دریافت خواهد کرد. برای مثال شماره سوکت به شکل زیر به برنامه شما داده می‌شود:

```
$ ./loadbalancer 61919
```

پس از پذیرش سوکت و برقرای اتصال، بسته‌هایی با پروتکل HTTP دریافت خواهد کرد. برنامه شما هنگام دریافت درخواست کافیست قابلیت پشتیبانی از تابع GET در پروتکل HTTP نسخه ۱ را داشته باشد و هر درخواست خارج از این حالت را باید با عبارت `Not Implemented` در تابع GET پاسخ دهد و سوکت را ببندد. بسته های دریافتی استاندارد به شکل زیر خواهند بود:

```
GET <appID>/<input> HTTP/1.0
<Headers>
[blank line]
<data>
```

مثال:

```
GET myapp1/exam_score HTTP/1.0
From: saeed@sut.edu
```

```
password:eViL2048
```

## ۳.۶. ارسال بسته‌ها به کارگزار

برنامه شما باید از روی `appID` و پرونده `MirrorServers` جفت‌های آدرس و پورتی که می تواند به آن وصل شود را پیدا کند و طبق الگوریتم چرخش عادلانه و به ترتیب آدرس‌های داده شده، کارگزار مناسب را پیدا کند و اگر سوکت باز با آن کارگزار نداشت درخواست برقرای اتصال سوکت دهد و پس از برقراری اتصال، پیغام دریافتی از کارخواه را بدون تغییر از طریق این سوکت بفرستد. نیاز نیست درستی سرآیندها و داده‌ی بدنه‌ی HTTP را بررسی نمایید.

## ۴.۶. دریافت پاسخ کارگزارها

هر لحظه ممکن است کارگزار پاسخ درخواست‌های مربوط به کارخواه‌ها را بدهد، اگر درخواست خارج از استاندارد بود، عبارت `Bad respond` را چاپ کند و سوکت خود را با کارگزار خاطی ببندد. در هنگام دریافت پاسخ از کارگزار کافی است که برنامه شما قابلیت پشتیبانی از شکل استاندارد زیر را داشته باشد:

```
GET <appID>/<input> HTTP/1.0
<Headers>
[blank line]
<data>
```

سپس از طریق تطبیق مقدار `<input>` با مقدار آن در بسته‌های دریافتی از کارخواه‌ها، کارخواه مناسب را پیدا کند و همین بسته را بدون تغییر از طریق سوکت ایجادشده با این کارخواه ارسال نماید و بعد از ارسال، سوکت خود را تنها با کارخواه ببندد و همچنان سوکت رابرای ارتباط با کارگزار حفظ کند.

تضمین می‌شود تمام بسته‌های دریافتی از کارگزارها و کارخواه‌ها در بخش `<data>` حاوی دقیقاً یک خط داده فقط شامل حروف، اعداد و علائم نگارشی در کدگذاری `ASCII` باشد؛ همچنین تضمین می‌شود `<appID>` در تمام بسته تنها حاوی حروف و اعداد و مقدار `<input>` یک عدد طبیعی قابل ذخیره در حافظه ۳۱ بیتی باشد.

برنامه متوازن کننده بار باید در هر شرایطی بتواند بسته‌های سالم دریافتی از طرف کارگزار و کارخواه‌هایی که پروتکل را رعایت می‌کنند را به درستی انتقال دهد، به این معنی که خطا در ارتباط با یک کارگزار یا کارخواه مثل بسته شدن سوکت از سمت مقابل یا نقض پروتکل روی بقیه ارتباط‌ها تاثیری نداشته باشد.

پاسخ کارگزار به کارخواه حاوی عبارتی است که کارخواد متوجه شود به کدام کارگزار متصل شده است، به همین منظور پاسخ به شکل `appId:{} serverId:{} value:{}` می‌باشد.

## ۷. نکات ضروری

- تمرین‌ها به صورت خودکار کامپایل می‌شوند (در صورت استفاده از زبان جاوا یا زبان سی)، آزمون و نمره‌دهی برای تمرین اول به صورت برون‌خط[۱۲] صورت می‌گیرد.
- نحوه‌ی ارزیابی ممکن است دچار تغییراتی شود و روش‌های ارزیابی دیگری اضافه شوند.
- در صورتی‌که هر مشکل یا پرسشی داشتید که فکر می‌کنید پاسخ آن برای همه مفید خواهد بود، آن را به گروه اینترنتی درس ارسال کنید.
- از فرستادن جواب تمرین به گروه اینترنتی درس خودداری کنید.
- تمام برنامه‌ی شما باید توسط خود شما نوشته شده باشد. فرستادن کل یا قسمتی از برنامه‌تان برای افراد دیگر، یا استفاده از کل یا قسمتی از برنامه‌ی فرد دیگری، حتی با ذکر منبع، تقلب محسوب می‌شود.
- پس از اتمام کارتان لازم است پاسخ خود را به هر دو بخش تمرین فشرده کرده و از طریق وب‌سایت پرتو ارسال نمایید.

[۱۲] Offline