به نام خداوند بخشنده مهربان

دانشگاه صنعتی شریف
مدرس: مهدی خرازی

تمرین برنامه نویسی صفرم *
شبکه‌های رایانه‌ای

دانشکده‌ی مهندسی کامپیوتر
زمستان ۱۳۹۷

اهداف تمرین

- آشنایی با نحوه‌ی کار گیت
- آشنایی با ارتباطات کارگزار ـ کارخواه
- آشنایی با Socket

۱. مقدمه

در این تمرین شما ابتدا با نحوه‌ی کار با سامانه‌ی گیت آشنا می‌شوید. تمرین اصلی مربوط به برنامه‌نویسی سوکت [۱] می‌باشد. پس از انجام این تمرین شما باید یک Makefile برای تمرین خود ایجاد کنید و سپس تمرین خود را در سامانه‌ی گیت بارگذاری نمایید.

* با سپاس از تیم دستیاران آموزشی

[۱] Socket Programming

۲. راه‌اندازی مقدمات

۱.۲ Gogs

تمامی تمارین فردی و گروهی شما از طریق این سامانه دریافت می گردد. بنابراین شما نیازمند یک حساب کاربری هستید. تیم دستیاران تمرین برای شما مخازن خصوصی می‌سازند و اطلاعات آن را در اختیار شما قرار می‌دهند.

۱.۱.۲ Git Config

شما برای انجام همه‌ی تمارین نیازمند استفاده از لینوکس هستید. بنابراین اگر از یک ماشین ویندوزی استفاده می‌کنید این عملیات را در یک ماشین مجازی انجام دهید. دستورهای زیر را در ترمینال ماشین لینوکسیِ خود اجرا کنید تا تنظیماتی که برای کامیت‌های خود [۲] استفاده می‌کنید برقرار گردد.

```
git config --global user.name "Your Name"
git config --global user.email "Your Email"
```

۲.۱.۲ ssh-key

در این مرحله نیاز دارید که کلیدهای ssh خود را به منظور احراز هویت از درون ماشین مجازی خود تنظیم کنید. برای این کار دو دستور زیر را به ترتیب اجرا کنید :

```
ssh-keygen -N "" -f ~/.ssh/id_rsa
cat ~/.ssh/id_rsa.pub
```

دستور اول یک جفت کلید ssh برای شما تولید می‌کند. دستور دوم کلید عمومی‌تان را در ترمینال نمایش می‌دهد. شما می‌بایست به این سامانه ورود کرده و سپس از این قسمت کلید عمومی خود را به حساب خود اضافه کنید. کلید شما باید با عبارت ssh-rsa شروع شده باشد.

۳.۱.۲ Repos

برای دسترسی به پرونده‌های مورد نیاز هر تمرینِ درس، یک مخزن عمومی ساخته شده است. این مخزن از آدرس زیر در دسترس است:

```
git@tarasht.ce.sharif.ir:ce443-972-students/ce443-972-handouts.git
```

همه‌ی پرونده‌های مورد نیاز شما در پوشه‌ی مربوط به تمرین است. برای دریافت این مخزن می‌توانید از دستورات زیر استفاده کنید:

[۲] Commit

```
$ git clone git@tarasht.ce.sharif.ir:ce443-972-students/ce443-972-handouts.git
```

برای اتصال به مخازنی که از قبل برای شما به صورت خصوصی تعریف شده کافی است به مسیر دلخواه (در این مستند فرض کرده‌ایم در مسیر ~/code/ مخزن‌های گیت را بارگیری می‌کنید) خود رفته و دستورات زیر را اجرا کنید:

```
git clone PERSONAL-REPO-URL
```

که PERSONAL-REPO-URL ساختاری شبیه به:

```
git@tarasht.ce.sharif.ir:ce443-972-students/ce443-972-"student-id".git
```

خواهد داشت.

۳. برنامه‌نویسی سوکت

۱.۳. مقدمه

همان‌طور که در درس می‌خوانید، برای انتقال داده‌ها می‌توانیم از واسط برنامه‌نویسی سوکت استفاده کنیم. در این تمرین، هدف این است که شما با برنامه‌نویسی سوکت آشنا شوید و یک ارتباط ساده برقرار کنید.

۲.۳. توضیح تمرین

به صورت ساده، شما باید کد همتاهایی [۳] را بزنید که هر کدام قابلیت این را داشته باشند که به دیگری وصل شوند و به یکدیگر پیام ارسال نمایند. هر همتا، در واقع یک پردازه در سامانه است که هنگام اجرا، درگاهی[۴] که قرار است روی آن گوش کند را دریافت می‌کند. در صورتی که همتایی به شما متصل شد، دیگر نیازی نیست روی این درگاه گوش کنید و منتظر ارتباط‌های بعدی باشید و تنها کافیست با این همتا ارتباط برقرار کنید. پس به طور کلی رفتار برنامه‌ی شما اینگونه است:

ابتدا با دستور

```
./start PORT
```

برنامه شما شروع به اجرا می‌کند و روی درگاه وارد شده گوش می‌کند.

نکته: عدد وارد شده بزرگتر از ۱۰۰۰۰ است، پس نیازی نیست نگران اشغال درگاه‌های رزرو شده باشید.

هنگامی که پردازه دیگری به شما متصل شد، باید عبارت

[۳] peer
[۴] port

`connected to` $PORT$

را در هر دو طرف اتصال چاپ کنید که هر کدام، شماره‌ی درگاهی که سر دیگر سوکت آن‌ها به آن متصل شده را چاپ می‌کنند.

همچنین در صورتی که در خط فرمان دستور

`connect` $PORT$

وارد شد، باید از طریق ارتباط سوکت به این پردازه با پورت مبداء خودتان به‌علاوه‌ی یک، وصل شوید. بنابراین سر دیگر ارتباط شما را با یک عدد بیشتر می‌بیند.

در صورتی که به همتای دیگر متصل شدید، با دستور

`exit`

ارتباط خود را با سر دیگر قطع کنید. و دوباره طرفین روی درگاهی که گوش می‌کردند، منتظر می‌مانند.

هنگامی که دستور

`send` $Message$

را دیدید، برای همتایی که به آن متصل هستید، پیامی که به جای قسمت Message آمده را به ترتیب کاراکتر از چپ به راست ارسال کنید و چیز بیشتر یا کمتری ارسال نکنید. سپس در طرف دیگر این پیام را اینگونه چاپ کنید:

`recv` $Message$

تنها هنگامی که دستور

`listen`

در ترمینال شما وارد شد، دستور

`accept`

را اجرا کنید و آنقدر منتظر بمانید که همتای دیگری به شما متصل شود. سپس دوباره منتظر دستورات ترمینال شوید.

تنها هنگامی که دستور recv در ترمینال شما وارد شد و سوکت شما در حال اتصال بود، دستور

`recv`

را اجرا کنید و به حالت blocking بروید تا مادامی که بسته‌ای از طرف مقابل برای شما بیاید و پس از آن دوباره به منتظر دستورات بعدی ترمینال باشید. دقت کنید که شما باید دستور recv را در کد خود داشته باشید.

نکته: اگر در ارتباطی قرار داشتید، با دیدن دستور connect یا اگر متصل نبودید با دیدن دستور exit و send پیام

```
invalid command
```

را چاپ کنید.

نکته: دقت کنید که با دو ساده‌سازی مربوط به دستور listen و recv دیگر شما نیاز به برنامه نویسی multi-thread نخواهید داشت. اگر از برنامه‌نویسی multi-thread استفاده می‌کنید، دستورات listen و recv را در دو همتا نادیده بگیرید و توجه کنید که داوری به صورت single-thread انجام می‌شود و لازم نیست پیچیدگی زیادی در کد خود داشته باشید.

نکته: تضمین می‌شود اندازه‌ی پیام‌ها کوچک است و نیازی به استفاده از چند بار دستور recv نیست.

۳.۳. نکات پیاده سازی

برای پیاده سازی می‌توانید از دو زبان c++ و java استفاده کنید. استفاده از ابزار telnet یا nc می‌تواند برای آزمون برنامه به شما کمک کند.

آموزش بسیار خوبی برای زبان c++ در مورد socket وجود دارد با نام beejs که در وب‌گاه درس موجود است. خواندن بخشی از آن شما را برای تمام قسمت‌های این تمرین آماده می‌کند.

کار با زبان java راحت‌تر است و پیچیدگی‌های کم‌تری دارد، ولی فراموش نکنید که واسط برنامه‌نویسی سوکت برای ارتباط با سامانه‌عامل طراحی شده و سامانه‌های لینوکسی با زبان C توسعه پیدا کرده و به همین دلیل برنامه‌های سی ارتباط بهتری با سامانه دارند، اگرچه پیچیدگی‌های بیش‌تری هم دارند.

در هر صورت، چه با زبان java و چه با زبان c++ پرونده‌ی شما قابلیت این را داشته باشد که با دستور make کامپایل شده و سپس با دستوراتی که پیش‌تر گفته شد، اجرا شود در غیر اینصورت شما هیچ نمره ای از این تمرین نخواهید گرفت!

make ابزاری است که به صورت خودکار برنامه های اجرایی و کتابخانه‌ها را از کد منبع تولید می‌کند و این کار را به کمک خواندن پرونده‌ی Makefile انجام می‌دهد. Makefile تعیین می‌کند که چگونه به برنامه هدف دسترسی پیدا کند. به این صورت که فهرست تمامی وابستگی‌ها[۵] را در آن قرار می‌دهید و make با پیمایش آن‌ها برنامه اجرایی شما را تولید می‌کند. متاسفانه make ساختار بسیار پیچیده‌ای دارد که اگر به صورت درست از آن‌ها استفاده نکنید برای فهم آنها دچار مشکل خواهید شد. برای یادگیری make می‌توانید اینجا و اینجا را مشاهده کنید. هم‌چنین مستندسازی رسمی GNU هم از اینجا قابل دسترس است که مسلماً حجم بیشتری دارد.

۴. تحویل دادنی‌ها

برای این تمرین نیاز است یک پرونده شامل کد تمرین و یک پرونده‌ی Makefile ارسال نمایید. به ازای هر تمرین در مخزن شخصی خود یک پوشه با حروف بزرگ و با شماره‌ی تمرین بسازید، سوالات تمرین را پاسخ داده و همه‌ی پرونده‌های لازم را با همان نامی که در مستند تمرین ذکر شده است جهت نمره دهی با دستورهای زیر ارسال کنید:

[۵] dependency

```
git status
git add PA0
git commit -m "Finished PA0"
git push origin master
```