هو الحق

**دانشکده‌ی مهندسی کامپیوتر**

CE-40443 پاییز ۱۳۹۶

# شبکه‌های کامپیوتری ـ تمرین دوم

*استاد: دکتر جعفری* *موعد تحویل: ساعت ۲۳:۵۹ ـ ۱۷ آذر*

## ۱ مقدمه

هدف از این تمرین، آشنایی با پروتکل TCP و پیاده‌سازی نمونه‌ای نسبتا ساده از ویژگی‌ها و سازوکارهای مورد استفاده در این پروتکل است. این تمرین از دو بخش عمده تشکیل شده است.

بخش اول مربوط به پیاده‌سازی ساختار برقراری ارتباط [۱] ، تعامل و قطع ارتباط توسط دو سیستم (میزبان) [۲] در دو سوی ارتباط است. بخش دوم به نحوه ارسال و پیغام‌های کنترلی، و از همه مهم‌تر بر روی کنترل ازدحام [۳] تمرکز دارد.

## ۲ پیش‌زمینه

در این بخش درباره پروتکل‌های لایه‌ی انتقال [۴] به‌ویژه TCP/IP و مفاهیم ازدحام توضیح داده خواهد شد. در صورتی که این اطلاعات برای شما بدیهی است، می‌توانید این قسمت را رد کرده و از بخش بعد ادامه دهید.

### ۱.۲ پروتکل TCP

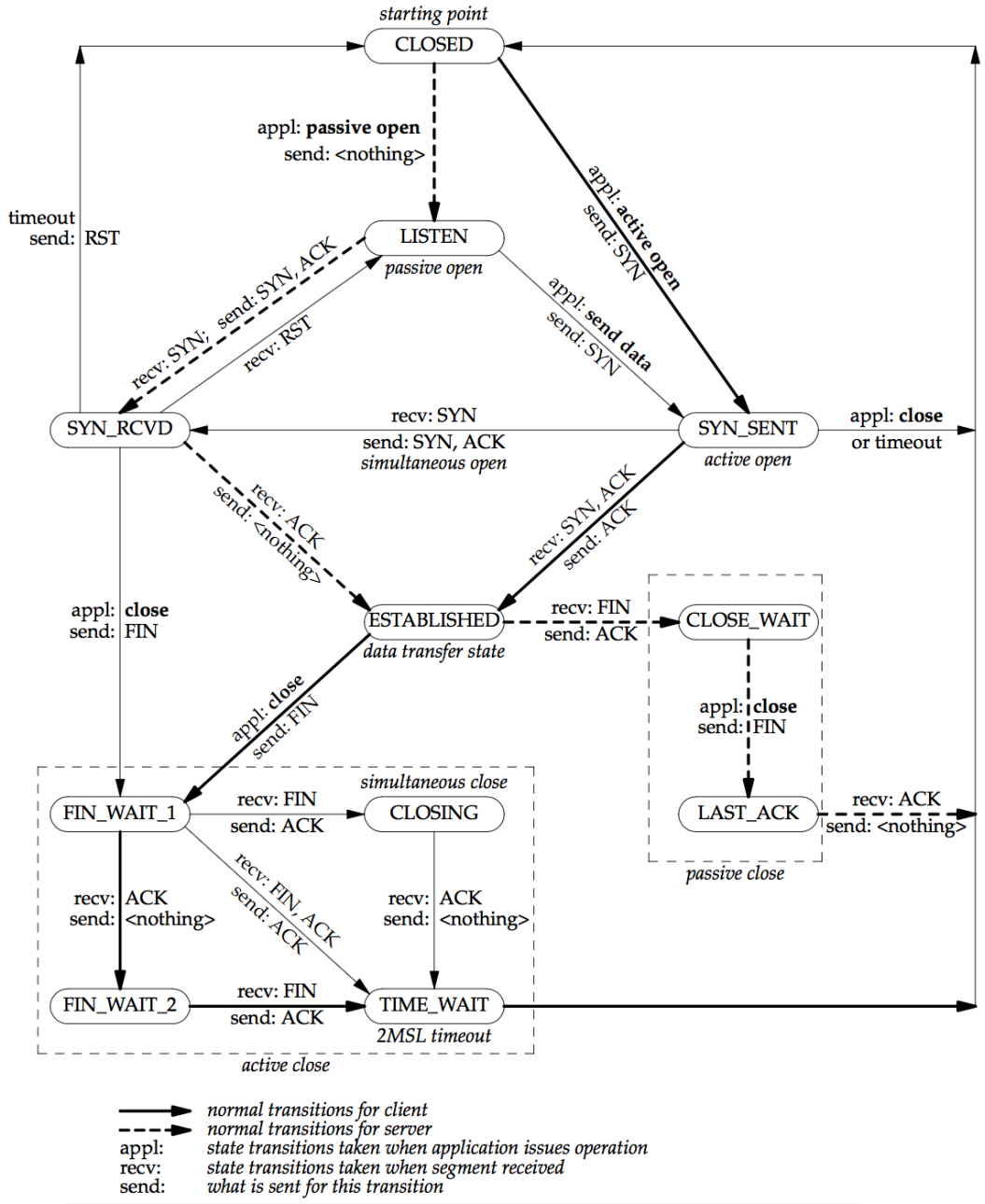
پروتکل کنترل انتقال [۵] یا به اختصار TCP ، یکی از پروتکل‌های مطرح لایه انتقال به حساب می‌آید که بر روی پروتکل IP [۶] در لایه سوم یعنی شبکه [۷] قرار میگیرد. [۸] IP تنها با پکت‌ها سر و کار دارد. این در حالی است که TCP سازوکاری فراهم آورده تا دو میزبان یک ارتباط را شروع کرده و جریانی از داده را با اطمینان از رسیدن، صحت و ترتیب درست پکت‌ها مبادله

[۱] Connection
[۲] End-host
[۳] Congestion Control
[۴] Transform Layer
[۵] Transmission Control Protocol
[۶] Internet Protocol
[۷] Network Layer
[۸] مدل هفت لایه OSI شبکه، مدلی مفهومی از لایه‌هایی است که دو یا چند سیستم از طریق آن به یکدیگر متصل می‌شوند.

کنند. برنامه‌های بسیاری مانند شبکه جهانی وب، پست‌های الکترونیکی یا برنامه‌های انتقال فایل از ترکیب IP/TCP به دلیل همین اطمینان استفاده می‌کنند. از طرفی دیگر، چون این سازکار بررسی موجب تاخیر می‌شود، دسته دیگری از برنامه‌ها که بر درستی کامل داده پافشاری نمی‌کنند از پروتکل دیگری به نام UDP بهره برده که سرعت بر درستی اولویت دارد. در ادامه به برخی از ویژگی‌های اصلی TCP می‌پردازیم.

## ۲.۲ شروع و پایان ارتباط

به طور کلی، ارتباطات TCP از ۳ فاز اصلی تشکیل شده‌اند: ***برقراری ارتباط***، ***انتقال داده*** و ***پایان ارتباط***



این قسمت با تبادل ۳ پیغام به صوت 3-way Handshake صورت می‌گیرد. این ۳ پیغام عبارت است از SYN، SYN/ACK ، ACK که دو پیام از شروع‌کننده ارتباط به دیگری ارسال می‌شود و یک پیام را دیگری برای شروع کننده ارسال می‌کند.برای اتمام ارتباط، یک handshake ۴ مرحله ای بین دو طرف ارتباط مبادله می‌شود ( FIN، ACK ، FIN و ACK ) در بالا تصویری از نمودار حالت [۹] یک ارتباط TCP آمده‌است،

## ۳.۲ انتقال داده و مدیریت ازدحام

اطلاعات مربوط به پروتکل و این بخش در کلاس درس توضیح داده شده است. همچنین برای اطلاعات بیشتر، میتوانید به اسلایدها، پیوند ۱ و پیوند ۲ مراجعه کنید. به علاوه در طی کلاس حل تمرین توضیحات بیشتر خواهد شد.

## ۴.۲ بسته‌های TCP

در این تمرین از شما خواسته شده قطعه مربوط به لایه انتقال بسته [۱۰] ها را پیاده‌سازی کنید. این بدین معناست که باید بارداده [۱۱] و سرآیند TCP را به درستی پر کنید.

سرآیند پروتکل TCP به شکل زیر است:

**TCP Header**

| *Offsets* | Octet | 0 | | | | | | | | 1 | | | | | | | | 2 | | | | | | | | 3 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Octet | Bit | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| 0 | 0 | Source port | | | | | | | | | | | | | | | | Destination port | | | | | | | | | | | | | | | |
| 4 | 32 | Sequence number | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 8 | 64 | Acknowledgment number (if ACK set) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 12 | 96 | Data offset | | | | Reserved 0 0 0 | | | | NS | CWR | ECE | URG | ACK | PSH | RST | SYN | FIN | Window Size | | | | | | | | | | | | | | |
| 16 | 128 | Checksum | | | | | | | | | | | | | | | | Urgent pointer (if URG set) | | | | | | | | | | | | | | | |
| 20 ... | 160 ... | Options (if *data offset* > 5. Padded at the end with "0" bytes if necessary.) ... | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

معنای هر فیلد از این سرآیند را می‌توانید از این پیوند مطالعه کنید. همچنین نحوه پر کردن فیلدهای این سرآیند در این تمرین، در قسمت **سناریو** آمده است.

---

[۹] State diagram
[۱۰] Packet
[۱۱] Payload

# ۳ سناریو

به طور خلاصه شما باید در این تمرین TCP و به ویژه ازدحام را از دو منظر فرستنده ( `sender` ) و گیرنده ( `receiver` ) پیاده‌سازی کنید. هر دو نیز در ارتباط خود سه فاز مختلف را می‌گذرانند.

فرستنده، شروع‌کننده و پایان‌دهنده ارتباط است. فرستنده درخواست برقراری ارتباط را می‌دهد و پس از مبادله سه بسته[۱۲]ی اولیه، سپس پیام را به بسته‌هایی تقسیم کرده و شروع به ارسال بسته‌های داده و دریافت بسته‌ی `ACK` مربوط به هر یک می‌کند. در انتها وقتی از رسیدن آخرین بسته‌ی داده اطمینان حاصل کرد (یعنی `ACK` آن را دریافت کرد)، فاز سوم پایان دادن به ارتباط را آغاز می‌کند و به ارسال و دریافت بسته‌های `FIN` و `ACK` مربوط به آن می‌کند و در آخر پایان می‌یابد.

گیرنده پس از دریافت درخواست برقراری ارتباط (بسته `SYN` ) باقی مراحل Handshake را انجام می‌دهد. پس از اتمام موفقیت آمیز این فاز، به ازای دریافت بسته‌های داده، بسته `ACK` مناسب را برای فرستنده موردنظر ارسال می‌کند. در انتها، منتظر فرستنده مانده تا فاز انتهایی بستن را آغاز کند و چهار بسته آن مبادله شود و برنامه پایان یابد.

در ادامه به برشمردن نکات مربوط به ساختار بسته‌ها می‌پردازیم:

## ۱.۳ ساختار بسته‌ها

تمامی بسته‌ها حاوی یک بخش سرآیند هستند که طول پیش‌فرض سرآیند TCP را دارد.[۱۳] بسته‌هایی که حاوی بخشی از داده مورد نظر برای انتقال هستند، بارداده[۱۴]ای به طول حداکثر یک MSS[۱۵] دارند. بسته‌هایی که داده‌ای ندارند و تنها نقش بسته‌های کنترلی `ACK` ، `SYN` ، `FIN` یا ... را دارند نیز تنها از یک سرآیند تشکیل شده‌اند و بارداده‌ای ندارند.

***نکته:*** *این امکان وجود دارد که در آخرین بسته حاوی داده، بارداده کمتر یک MSS باشد.*

## ۲.۳ سرآیند بسته‌ها

- همانطور که پیش‌تر گفته شد، سرآیند بسته‌ها بخش اختیاری ( `Options` ) را ندارند.
- مقدار مربوط به شماره پورت مبدا و مقصد، همان‌طور که واضح است باید حاوی عدد پورت ارسال‌کننده و دریافت‌کننده‌ی بسته باشد. این مقدار یک عدد ۱۶ بیتی بدون علامت است.

[۱۲] Packet
[۱۳] یعنی سرآیند بسته‌های ما بخش Options را ندارند.
[۱۴] Payload
[۱۵] Maximum Segment Size

- مقدار `Sequence number` از 0 تا $2^{32} - 1$ می‌تواند باشد و سه حالت دارد:

  ۱. در صورتی که بسته از سمت گیرنده ارسال شده باشد، **صفر** است. در غیر این صورت:

  ۲. در صورتی که پرچم[۱۶] `SYN` ۱ مقداردهی شده باشد، مقدار این فیلد، یک مقدار اولیه از تصادفی است.

  ۳. در صورتی که پرچم `SYN` صفر باشد، این فیلد حاوی یک مقدار افزایشی[۱۷] نسبت به مقدار اولیه (حالت قبلی) است که به شماره **هر سگمنت** زیاد می‌شود. [۱۸]

- مقدار `Acknowledgement number` نیز از 0 تا $2^{32} - 1$ می‌تواند باشد. در بسته‌هایی که پرچم `ACK` آن‌ها مقدار یک داده شده `Sequence number` بسته **بعدی** که منتظرش هستند، می‌باشد و برای باقی بسته‌ها مقدار **صفر** را دارد.
- در میان پرچم‌ها، پرچم `SYN` ، `FIN` ، `ACK` ، `CWR` را درست مقداردهی کنید و باقی پرچم‌ها را صفر بگذارید.
- مقدار `Window size` را اندازه پنجره ارسال **فرستنده** (۱۶بیت بدون علامت) مقداردهی کنید. [۱۹]
- فیلد `Checksum` و `Data offset` را طبق روشی که در TCP محاسبه می‌شود، مقداردهی کنید.
- قسمت `Urgent pointer` صفر بگذارید.

### Checksum

در RFC793 آمده است که:

```
The checksum field is the 16 bit one's complement of the one's complement sum of
all 16-bit words in the header and text. If a segment contains an odd number of
header and text octets to be checksummed, the last octet is padded on the right
with zeros to form a 16-bit word for checksum purposes. The pad is not transmitted
as part of the segment. While computing the checksum, the checksum field itself is
replaced with zeros.
```

شما نیز باید طبق آن checksum را محاسبه و **چک** کنید. تنها در نظر داشته باشید، از آن جا که در تمرین سرآیند IP نداریم، نیاز نیست سرآیند pseudo-TCP حاوی اطلاعات IP بسازید. (تنها سرآیند TCP و بارداده را درنظر بگیرید.)

## ۳.۳ ارسال داده

در روند ارسال داده، در ابتدا نرخ ارسال داده و اندازه پنجره ازدحام[۲۰] یک MSS است و تا زمانی که ارسال داده و دریافت `ACK` بدون خلل و به طور درست انجام گیرد، این اندازه به طور نمایی تا حداکثر اندازه پنجره افزایش می‌یابد. به این فاز

[۱۶] Flag
[۱۷] Incremental
[۱۸] پس از $2^{32} - 1$ عدد 0 می‌آید.
[۱۹] در رابطه با این Sliding window و نحوه مقداردهی این فیلد، در کلاس حل تمرین توضیحات لازم داده خواهد شد.
[۲۰] Congestion window

Slow Start می‌گویند. *برای اطلاعات به این پیوند مراجعه کنید.*

در حین ارسال ممکن است دو مشکل رخ دهد: Timeout یا Triple Duplicate Acknowledgement

اگر بسته‌ای timeout شود، یعنی `ACK` آن نرسد یا آنقدر دیر برسد که از زمان timeout که دو برابر Round Trip Time است، بگذرد، بسته موردنظر بازارسال[۲۱] می‌شود و اندازه پنجره ارسال به یک MSS بازنشانی می‌شود. سپس تا نصف اندازه پنجره قبل از رخ دادن timeout به طور نمایی افزایش می‌یابد و بعد به طور خطی زیاد می‌شود.

در صورتی که Triple duplicate acknowledgement رخ دهد، یعنی سه بار بسته `ACK` تکراری توسط گیرنده ارسال شود (در کل ۴ بسته با شماره `ACK` یکسان)، بسته مورد نظر باز ارسال می‌شود و همچنین نرخ ارسال نصف مقدار کنونی شده و از این به بعد به صورت خطی زیاد می‌شود.

در این تمرین، حداقل اندازه پنجره ۱MMS و حداکثر اندازه آن نیز به عنوان آرگومان ورودی در ابتدای برنامه مشخص می‌شود. در تقسیمات برای اندازه پنجره، اعداد را رو به پایین گرد کنید. (طبیعتا اندازه از ۱ کمتر نخواهد شد.)

همچنین در تمرین، مقدار Maximum Segment Size برای بارداده را **۱۴۸۰ بایت** در نظر بگیرید.

نکات زیر را برای دریافت بسته‌ها و ارسال `ACK` از سمت گیرنده را رعایت کنید:

- اگر گیرنده بسته‌ای جلوتر از آنچه انتظارش را می‌کشید را دریافت کرد، باید آن را ذخیره و بافر کرده تا در زمان مناسب از آن استفاده کند.
- همچنین نیاز نیست در صورت دریافت بسته‌های قدیمی‌تر گیرنده `ACK` تکراری[۲۲] به گیرنده ارسال کند.
- پنجره ارسال در صورت دریافت `ACK` درست جلو می‌رود و بسته جدید ارسال می‌کند.
- در صورتی که فرستنده Duplicate ACK را تشخیص داد و عملیات بازارسال پنجره را انجام داد، تا زمانی که `ACK` درست را دریافت نکرده یا timeout رخ نداده نباید بسته جدیدی ارسال کرده یا چندباره Duplicate ACK تشخیص بدهد.
- بازارسال پنجره به یک‌باره انجام می‌شود. پس زمان ارسال بسته‌های آن و مقدار فیلد Window Size یکسان است.

## ۴.۳ Timeout

مقدار timeout دو برابر RTT است. برای محاسبه زمان RTT در ابتدا به شما عددی پیش‌فرض داده می‌شود. پس از آن برای محاسبه RTT از فرمول زیر استفاده کنید و عدد محاسبه‌شده را به سمت بالا گرد کنید. $(\alpha = 0.125)$

$$EstimatedRTT = (1 - \alpha).PreviousEstimatedRTT + \alpha.SampleRTT$$

برای بروز رسانی مقدار RTT تنها `ACK` هایی را در نظر بگیرید که به درستی و بدون خللی دریافت شده‌اند.

---

[۲۱] Retransmit

[۲۲] Duplicate

# ۴ پیاده‌سازی

تمامی ارسال و دریافت بسته‌ها و سیگنال‌ها در این تمرین از طریق فایل‌های `fifo` صورت می‌گیرد. به این شکل که هر قسمت برای دریافت، یک فایل `fifo` داشته و از آن می‌خواند و بخش دیگر در آن فایل می‌نویسد.[۲۳]

با دستور زیر (تحت bash) می‌توان یک فایل `fifo` ساخت.[۲۴]

```
makefifo <NAME>
```

## ۱.۴ مقدمه

به طور طبیعی، کافیست که فرستنده و گیرنده در فایل‌های یکدیگر بنویسند. همچنین فرستنده از طریق یک فایل `fifo` دیگر سیگنال زمان را دریافت می‌کند (مشابه `STDIN` ) ولی به دلیل آنکه ارسال و دریافت بسته‌ها به طور عادی کاملا خودکار و غیرهمزمان[۲۵] اتفاق می‌افتد، نمی‌توان آن را به راحتی مورد آزمایش قرار داد. به همین دلیل، در این تمرین، عامل سومی را درنظر می‌گیریم که به طور خلاصه، وظیفه‌ی کنترل جریان داده را بر عهده دارد. در ادامه، این عامل میانی را، **میانجی** می‌نامیم.

## ۲.۴ فرستنده

فرستنده مسئولیت ارسال یک پیام را دارد_ که در این تمرین یک **فایل** دلخواه است. یک فرستنده با دستور زیر اجرا می‌شود:

```
./send <SENDER_PORT> <RECEIVER_PORT> <INIT_RTT> <MAX_WINDOW> <FILE_PATH>
```

- آرگومان `SENDER_PORT` شماره پورت فرستنده و آرگومان `RECEIVER_PORT` شماره پورت گیرنده
- آرگومان `FILE_PATH` آدرس مطلق[۲۶] فایل ارسالی
- آرگومان `INIT_RTT` ، همان زمان RTT اولیه پیش‌فرضی است که برای محاسبه Timeout مورد استفاده قرار می‌گیرد.
- آرگومان `MAX_WINDOW` حداکثر سایز پنجره ارسال است.

با اجرای فرستنده، دو فایل `fifo` با نام‌های زیر ساخته می‌شود. فرستنده از این دو فایل به ترتیب برای دریافت بسته و کنترل زمان استفاده می‌کند.

```
sender_<SENDER_PORT>_data.pipe
sender_<SENDER_PORT>_time.pipe
```

[۲۳] فایل‌های fifo در واقع یک خط لوله مشخص اسم‌دار(Named Pipeline) هستند.
[۲۴] https://linux.die.net/man/3/mkfifo
[۲۵] asynchronous
[۲۶] Absolute path

### زمان

پس از اجرا، فرستنده باید همواره آماده دریافت دستور زیر درفایل `sender_<SENDER_PORT>_time.pipe` خود باشد:

```
tick
```

این دستور، حکم سیگنال کلاک ساعت برای بروز رسانی زمان را دارد. با هر بار وارد شدن آن، زمان فرستنده، **یک واحد** جلو می‌رود. در نتیجه، زمان به طور قطعی[۲۷]، نه توسط زمان محلی سیستم میزبان، بلکه توسط تست‌ها کنترل می‌شود.

### بسته‌ها

فایل فرستنده، که از آن می‌خواند، `sender_<SENDER_PORT>_data.pipe` و فایلی که در آن می‌نویسد (در واقع فایل بخش میانجی مسیر رفت) نیز `forwardnet_data.pipe` نام دارد.

خواندن و نوشتن بسته‌ها در فایل‌ها تماما به صورت **network-endian** و بایتی (rb, wb) صورت می‌گیرد. همچنین از آن‌جایی که طول بسته‌ها مشخص نیست، هر فرآیند خواندن/نوشتن در دو مرحله انجام می‌گیرد:

۱. ابتدا طول بسته مورد نظر در قالب عددی **۴ بایتی** به قالب network-endian نوشته/خوانده می‌شود.

۲. سپس، خود بسته نیز در قالب تعدادی بایت نوشته/خوانده می‌شود.

### خروجی

در صورتی که گیرنده با شماره پورت مورد نظر در حال اجرا نبود، فرستنده باید در خروجی `STDOUT` خود پیغام زیر را چاپ کند:

```
sending host <SENDER_PORT>: no receiving host <RECEIVER_PORT> is available.
```

با آغاز شدن ارتباط (ارسال اولین بسته‌ی `SYN` ) پیام زیر پیغام زیر در خروجی `STDOUT` فرستنده چاپ شود.

```
sending host <SENDER_PORT>: is running...
```

پس از پایان یافتن اجرای فرستنده نیز پیام زیر پیغام زیر در خروجی `STDOUT` فرستنده چاپ شود.

```
sending host <SENDER_PORT>: is terminated.
```

مشخص است که برای برقراری ارتباط، گیرنده پیش از فرستنده اجرا می‌شود و به نوعی کارگزار به حساب می‌آید.

[۲۷] Deterministic

## ۳.۴ گیرنده

گیرنده داده‌ای که فرستنده ارسال می‌کند را دریافت می‌کند. یک گیرنده با دستور زیر اجرا می‌شود:

```
./receive <RECEIVER_PORT>
```

### بسته‌ها

گیرنده بسته‌ها را از فایل `receiver_<RECEIVER_PORT>_data.pipe` خوانده و در فایل `netbackward_data.pipe` می‌نویسد. باقی قواعد خواندن و نوشتن، مانند حالت فرستنده است.

### خروجی

با اجرای گیرنده، این بخش آماده برقراری ارتباط می‌شود و پیغام زیر را در خروجی چاپ می‌کند.

```
receiving host <RECEIVER_PORT>: is running...
```

پس از بسته شدن ارتباط، گیرنده به کار خود پایان می‌دهد. پس از پایان یافتن اجرای گیرنده نیز پیغام زیر در خروجی `STDOUT` فرستنده چاپ شود.

```
receiving host <RECEIVER_PORT>: is terminated.
```

## ۴.۴ میانجی

به طور کلی میانجی ، مسئولیت تاخیر انداختن یا دراپ کردن بسته‌ها و یا تغییر بسته‌ها را دارد. **دو** بخش میانجی در این تمرین استفاده شده است: یکی در مسیر رفت داده‌ها و دیگری در مسیر برگشت `ACK` ها. این دو میانجی،در واقع، دو مسیر یا لینک Half-duplex را شبیه‌سازی کرده‌اند.

### مسیر رفت داده

در مسیر رفت، قسمت میانجی از فایل `netforward_data.pipe` بسته را خوانده و در `STDIN` می‌پرسد که با آن چه‌کار کند. اگر `s<NUM>` وارد شده بود (مثلا `s5` )، ابتدا به تعداد تعیین شده `tick` به فرستنده ارسال کرده (از طریق فایل `sender_<SENDER_PORT>_time.pipe` ) سپس بسته مورد نظر را برای گیرنده می‌فرستد.(مانند بخش‌های قبل، در فایل `receiver_<RECEIVER_PORT>_data.pipe` می‌نویسد) و اگر `d<NUM>` وارد شده بود (مثلا `d5` )، ابتدا به تعداد تعیین شده `tick` به فرستنده ارسال کرده و سپس بسته مورد نظر را drop می‌کند.

### مسیر برگشت ACK

در مسیر برگشت، قسمت میانجی از فایل `netbackward_data.pipe` بسته را خوانده و در `STDIN` می‌پرسد که با آن چه‌کار کند. اگر `s` وارد شده بود، بسته مورد نظر را برای فرستنده ارسال می‌کند.(در فایل `sender_<SENDER_PORT>_data.pipe`

مانند بخش‌های قبل می‌نویسد) و اگر `d` وارد شده بود، بسته `ACK` مورد نظر را drop می‌کند.

**نمونه‌ای** ساده از کد پایتون بخش میانی در اختیار شما قرار داده شده است. خواندن و نوشتن در فایل‌های `fifo` برای باقی قسمت‌ها، مشابه این قسمت است.

شما می‌توانید (باید) کد میانجی را مطابق با میلتان (برای تست تمامی حالت‌ها و لاگ گرفتن) تغییر دهید. در انتها، این بخش از کد شما تصحیح نخواهد شد و تاثیری در نمره شما ندارد.

### ۵.۴ مثال اجرا

در نتیجه اگر فرض کنیم، فرستنده و گیرنده به ترتیب با پورت‌های ۸۸۰۰ و ۹۴۶۴، قرار است فایلی با آدرس `/tmp/somefile.jpeg` را انتقال دهند، شما باید به ترتیب در ۴ ترمینال مجزا دستورات زیر را اجرا کنید:

گیرنده:

```
./renew.sh
./receive 9464
```

فرستنده:

```
./sender 8800 9464 10 50 "/tmp/somefile.jpeg"
```

میانجی رفت:

```
./pipe.py netforward_data.pipe receiver_9464_data.pipe sender_8800_time.pipe
```

میانجی برگشت:

```
./pipe.py netbackward_data.pipe sender_8800_data.pipe
```

## ۵ سوال نظری

همانطور که در زیرقسمت ۲.۲ دیدید، نمودار حالت برای یک ارتباط TCP کشیده شده‌است. تمامی حالاتی که در برقراری و همچنین پایان یافتن ارتباط TCP ممکن است پیش آید را بررسی کرده و به ازای دریافت یا عدم دریافت بسته‌های کنترلی مربوط، توضیح دهید هر یک از میزبان‌ها در چه شرایطی، در چه حالتی قرار می‌گیرند.

پاسخ این سوال را به فارسی در قالب Markdown به انتهای فایل `theory.md` مخزن `git` خود اضافه کنید.

# ۶ نکات پایانی

- این تمرین مانند تمرین‌های گذشته در قالب `git` تحویل گرفته می‌شود. پوشه‌ی `hw2` مربوط به این تمرین است. کدهای خود را در زیرپوشه `codes` قرار داده و فایل‌های `*.pipe` نیز در زیرپوشه `pipes` بسازید.
- پیش از هر سری اجرا از پاک شدن تمامی فایل‌های `*.pipe` در اجراهای پیشین، اطمینان حاصل کنید. بدین منظور، هر سری `renew.sh` را اجرا کنید.
- فایل‌های `.pipe` مربوط به فرستنده و گیرنده باید توسط خودشان در ابتدای اجرا ساخته و در انتهای اجرا پاک شوند. (همان‌طور که در مثال اجرا گفته شده‌است)
- می‌توانید از زبان‌های پایتون، C/C++ و جاوا استفاده کنید.ضمنا از کتابخانه‌های پیشرفته حق استفاده ندارید.
- این تمرین با سیستم‌عامل‌های Unix-based تصحیح می‌شود. در نتیجه دانشجویانی که سیستم‌عامل ویندوز دارند، موظفند تمرین خود را در ماشین مجازی[۲۸] لینوکس بزنند.
- در این تمرین و سایر تمرینهای درس، با هرگونه تقلب شدیداً برخورد خواهدشد.
- کد ها باید توسط خود شما نوشته شوند و کپی کردن از منابع اینترنتی تقلب محسوب می‌شود.
- در صورت داشتن هرگونه سوال ابتدا کمی تحقیق کرده و سپس سوال خود را از راه piazza بپرسید.
- توصیه می‌شود که در کلاس‌های حل تمرین مربوط به این تمرین شرکت کنید.
- تاخیر تا پنج روز بعد از موعد ارسال تمرین مجاز بوده و هر روز ٪۱۰ کسر نمره را شامل می‌شود.
- به ازای هر روز ارسال زودتر از موعد تمرین، ٪۵ نمره اضافی دریافت می‌کنید. (حداکثر ٪۳۵)

موفق باشید.

[۲۸] Virtual Machine