

به نام خدا



درس سیستم‌های عامل

نیم‌سال اول ۰۲-۰۳

دانشکده مهندسی کامپیوتر

دانشگاه صنعتی شریف

مدرس محمدعلی میرزایی، محدثه میربیگی

تمرین یک فردی

موضوع ساخت یک شل

موعد تحویل ساعت ۲۳:۵۹ شنبه ۲۰ آبان ۱۴۰۲

مسئول تمرین مهدی علیزاده

با سپاس از دکتر مهدی خرازی، محمد حدادیان، امیرمهدی کوششی

اقتباس شده از CS162 در بهار ۲۰۲۰ در دانشگاه کالیفرنیا، برکلی

در این تمرین، یک شل^۱ مشابه بش^۲ در سیستم خود را خواهید ساخت. هدف یک شل این است که به کاربر یک واسط جهت ارتباط با سرویس‌های سیستم عامل، که شامل مدیریت پرونده‌ها و پرده‌ها می‌شود، ارائه دهد. شل‌ها می‌توانند تعاملی^۳ یا غیرتعاملی^۴ باشند. به عنوان مثال، هنگامی که یک اسکریپت بش اجرا می‌کنید، به طور غیرتعاملی از بش استفاده می‌کنید. اجرای دستور bash بدون آرگومان یا با پرچم -i بش را در حالت تعاملی اجرا می‌کند. هسته سیستم عامل^۵ مستندات بسیار مفیدی جهت ساخت شل‌ها فراهم نموده است. با ساخت یک شل برای خودتان با این روابط بیشتر آشنا خواهید شد و احتمالاً اطلاعاتی پیرامون سایر شل‌ها نیز کسب خواهید نمود.

۱ راه‌اندازی مقدمات

برای زدن تمرین عملی، شما نیاز به سیستم عامل Linux، Ubuntu ورژن ۲۰ خواهید داشت. برای اینکه تمامی دانشجویان در یک محیط کد بزنند پیشنهاد می‌کنیم همه از همین ورژن استفاده کنید. برای استفاده از یک محیط ایزوله می‌توانید از یک ماشین مجازی با ubuntu ورژن ۲۰ استفاده کنید. برای این کار می‌توانید از ابزارهای موجود مانند virtualbox یا Parallels برای macOS استفاده کنید. همچنین برای سادگی می‌توانید از docker استفاده کنید و تست‌های خود را بر روی یک کانتینر با تصویر ubuntu:۲۰.۰۴ استفاده کنید. ما کدهای اولیه مورد نیاز برای ساخت شل را به همراه یک میک‌فایل^۶ در پوشه hw1 در اختیار شما قرار داده ایم. این کدها، شامل قطعه برنامه‌ای می‌شوند که یک رشته را دریافت می‌کند و آن را به کلمات، تقسیم می‌کند. به منظور اجرای شل می‌بایست دستورهای زیر را اجرا کنید:

```
1 $ make
2 $ ./shell
```

همچنین به منظور خاتمه دادن به اجرای شل پس از شروع آن، می‌توانید quit را تایپ کرده و یا CTRL-D را فشار دهید.

۲ پشتیبانی از فرمان‌های cd و pwd

ساختار کد شل شما یک نگارنده^۷ برای فرمان‌های داخلی^۸ دارد. در واقع هر شل یک مجموعه از فرمان‌های درونی دارد که کارکردهای مربوط به خود شل هستند و نه برنامه‌های خارجی. مثلاً فرمان quit به عنوان یک فرمان داخلی پیاده‌سازی شده است زیرا این فرمان خود شل را از اجرا خارج می‌کند. این شل که هم اکنون در اختیار شماست تنها دو فرمان داخلی دارد. فرمان ؟ که منوی راهنما را نشان می‌دهد و فرمان quit که شل را می‌بندد. در اولین بخش تمرین، شما قرار است فرمان جدید pwd را به مجموعه فرمان‌های داخلی اضافه کنید، که مسیر پوشه فعلی را در خروجی استاندارد چاپ کند. سپس فرمان درونی جدید cd را به فرمان‌های درونی شل اضافه کنید که یک ورودی از جنس مسیر (مثلاً /first/dir) می‌گیرد و مسیر کاری فعلی شل را به آن تغییر می‌دهد. راهنمایی: از chdir و getcwd برای ایجاد تغییرات لازم در فایل shell.c استفاده کنید.

۳ اجرای برنامه

اگر تلاش کنید چیزی در شل تایپ کنید که از فرمان‌های داخلی نباشد، یک پیام مبنی بر اینکه شل نمی‌داند چگونه باید برنامه را اجرا کند مشاهده خواهید کرد. طوری شل خود را تغییر دهید که هر گاه فرمان اجرای برنامه‌ای را به آن بدهید، بتواند

¹ shell

² bash

³ interactive

⁴ non-interactive

⁵ kernel

⁶ makefile

⁷ dispatcher

⁸ built-in commands

آن را اجرا کند. اولین کلمه فرمان، نام برنامه و مابقی کلمات ورودی‌های برنامه خواهند بود. فعلاً می‌توانید اینگونه در نظر بگیرید که کلمه اول، نشانی کامل^۹ برنامه است؛ بنابراین به جای اجرای `wc` باید `/usr/bin/wc` را اجرا کنید. در بخش بعدی تلاش خواهید کرد که به جای پشتیبانی از آدرس کامل برنامه، پشتیبانی از نام ساده آن (`wc`) را پیاده‌سازی کنید. شما باید تنها از توابع تعریف شده در `parse.c` برای جداسازی و شکستن متن ورودی به کلمات بهره ببرید. پس از پیاده‌سازی این گام قادر خواهید بود که برنامه‌های مشابه زیر را اجرا کنید:

```
1 $ ./shell
2 0: /usr/bin/wc shell.c
3    77 262 1843 shell.c
4 1: exit
```

وقتی شل بخواهد یک برنامه را اجرا کند، باید با فراخوانی یکی از توابع خانواده `exec` یک پرده فرزند فورک^{۱۰} کند. همچنین پرده والد باید تا اتمام کار پرده فرزند صبر کرده و سپس منتظر فرمان‌های بعدی باشد.

۴ تفکیک‌پذیری مسیرها

احتمالاً تاکنون متوجه شده‌اید که تست کردن شل در قسمت قبل بسیار سخت بود، زیرا باید مسیر کامل هر برنامه را وارد می‌کردید. خوشبختانه هر برنامه‌ای و از جمله آن‌ها برنامه شل، به یک مجموعه از متغیرهای محلی^{۱۱} دسترسی دارند که به صورت یک جدول هش^{۱۲} از زوج مرتب‌های کلید و مقدار سازماندهی شده‌اند. یکی از این متغیرهای محلی متغیر `PATH` است. شما می‌توانید این متغیر را بر روی ماشین مجازی خود چاپ کنید (توجه کنید که از بش برای این قسمت استفاده کنید نه از شل دست‌ساز خودتان):

```
1 $ echo $PATH
```

که خروجی آن مشابه زیر است:

```
1 $ /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:...
```

وقتی بش یا هر شل دیگری، بخواهد یک برنامه مانند `wc` را اجرا کند، در تمام مسیرهای موجود در متغیر محلی `PATH` به دنبال برنامه‌ای با نام `PATH` می‌گردد و اولین برنامه‌ای که پیدا کند را اجرا می‌کند. هر پوشه در `PATH` با استفاده از علامت : از سایرین جدا می‌شود. حال باید شل دست‌ساز خود را چنان تغییر دهید که از متغیر محلی `PATH` استفاده کرده و برنامه را با نام ساده آن نیز اجرا کند.

توجه ۱: باید کماکان از نوشتن مسیر کامل برنامه نیز پشتیبانی شود.

توجه ۲: به هیچ وجه از `execvp` استفاده نکنید وگرنه نمره‌ای به شما تعلق نخواهد گرفت. در عوض می‌توانید از `execv` استفاده کرده و مسیرهای موردنیاز را خودتان بسازید.

۵ هدایت ورودی/خروجی‌ها

گاهی می‌خواهیم یک برنامه به جای کار کردن با ورودی و خروجی استاندارد، ورودی خود را از یک پرونده بخواند یا خروجی خود را در یک پرونده بنویسد. دستور `[file] < [process]` به شل می‌گوید که خروجی استاندارد پرده باید در یک پرونده نوشته شود. به طور مشابه دستور `[file] > [process]` به شل می‌گوید که محتوای پرونده را به عنوان ورودی استاندارد پرده به کار برد. شما باید شل خود را به گونه‌ای تغییر دهید که از هدایت^{۱۳} کردن ورودی و خروجی استاندارد به پرونده‌ها پشتیبانی کند.

^۹ full path

^{۱۰} fork

^{۱۱} environment variables

^{۱۲} HashTable

^{۱۳} redirect

نیازی به پشتیبانی از هدایت فرمان‌های داخلی شل (مثلاً `pwd`) ندارید. همچنین نیازی به پشتیبانی هدایت از `stderr` یا الحاق کردن^{۱۴} به پرونده‌ها (`[file]`) «`[process]`» نیست. فرض کنید که همواره پیرامون دو نویسه `<` و `>` فضای خالی وجود دارد. توجه کنید که `[file] >` یا `[file] <` به عنوان ورودی (یعنی در `argv`) به برنامه‌ها پاس داده نمی‌شوند.

۶ کار با سیگنال

اکثر شل‌ها به ما اجازه می‌دهند که اجرای پردازش‌ها را با فشردن برخی کلیدهای خاص مانند `CTRL-C` یا `CTRL-Z` با وقفه مواجه کرده یا کلاً متوقف کنیم. این کلیدها با فرستادن سیگنال به زیرپردازش^{۱۵}‌های شل کار می‌کنند. برای مثال با فشردن `CTRL-C` یک سیگنال `SIGINT` ارسال می‌شود که معمولاً برنامه در حال اجرا را متوقف می‌کند و با فشردن `CTRL-Z` یک سیگنال `SIGTSTP` ارسال می‌شود که معمولاً برنامه در حال اجرا را به حالت پس‌زمینه^{۱۶} می‌برد. اگر در حال حاضر این کلیدها را بر روی شل خود به کار ببرید، سیگنال‌ها به پردازش خود شل ارسال می‌شوند و این آن چیزی نیست که ما به دنبالش هستیم. مثلاً وقتی شما با فشردن `CTRL-Z` می‌خواهید یک زیرپردازش از شل را متوقف کنید، خود شل نیز متوقف می‌شود. در حالی که ما می‌خواهیم سیگنال‌ها فقط روی زیرپردازش‌های شل تاثیر بگذارد.

۱.۶ مثال: شل در شل

با استفاده از یک مثال می‌خواهیم بهتر متوجه عملکرد درست سیگنال‌ها در شل شویم. در این مثال از دو دستور `ps` و `jobs` استفاده می‌کنیم. دستور `ps` به ما اطلاعاتی درباره‌ی تمام پردازش‌های در حال اجرا روی سیستم می‌دهد و دستور `jobs` لیستی از برنامه‌هایی که توسط شل فعلی مدیریت می‌شوند را نشان می‌دهد. با وارد کردن دستورات زیر را در ترمینال ماشین مجازی خود، می‌بایست خروجی‌های مشابهی را مشاهده کنید:

```
1 vagrant@development ~ $ ps
2  PID TTY          TIME CMD
3 2672 pts/0    00:00:00 bash
4 13096 pts/0    00:00:00 ps
5 vagrant@development ~ $ sh
6 $ ps
7  PID TTY          TIME CMD
8 2672 pts/0    00:00:00 bash
9 13097 pts/0    00:00:00 sh
10 13098 pts/0    00:00:00 ps
```

در این قسمت ما با استفاده از دستور `sh`، یک شل بورن^{۱۷} درون شل بش خود آغاز کرده‌ایم.

```
1 $ cat
2 hello
3 hello
4 ^Z
5 [1] + Stopped      cat
6 $ ps
7  PID TTY          TIME CMD
8 2672 pts/0    00:00:00 bash
9 13097 pts/0    00:00:00 sh
10 13102 pts/0    00:00:00 cat
11 13103 pts/0    00:00:00 ps
```

^{۱۴}append

^{۱۵}subprocess

^{۱۶}background

^{۱۷}Bourne shell

توجه کنید که با فشردن CTRL-Z ما برنامه‌ی cat را متوقف کردیم در حالی که دو برنامه‌ی sh و bash به اجرا ادامه دادند.

```

1 $ jobs
2 [1] + Stopped      cat
3 $ fg
4 cat
5 world
6 world
7 ^C
8 $ ps
9 PID TTY          TIME CMD
10 2672 pts/0    00:00:00 bash
11 13097 pts/0    00:00:00 sh
12 13237 pts/0    00:00:00 ps
13 $ exit
14 vagrant@development ~ $

```

در این قسمت با دستور fg به اجرای برنامه‌ی متوقف شده‌ی cat ادامه داده‌ایم و با فشردن CTRL-C این برنامه را به طور کامل متوقف کرده‌ایم. همان طور که می‌بینید، این سیگنال نیز تاثیری بر روی دو برنامه‌ی شل دیگر نداشته است. ما نیز می‌خواهیم سیگنال‌هایی داشته باشیم که تنها بر روی زیرپردازه‌هایی که شل ایجاد کرده است اثر بگذارند. قبل از توضیح نحوه انجام این کار، قصد داریم پیرامون چند مفهوم در سیستم‌عامل‌ها بیشتر صحبت کنیم.

۲.۶ گروه‌های پردازش

می‌دانید که هر پردازش یک pid یکتا دارد. اما هر پردازش یک pgid مخصوص گروه خود را هم داراست که یکتا نیست و به صورت پیش فرض همان pgid پردازش والد است. پردازش‌ها می‌توانند شناسه گروه خود را دریافت و مقداری بکنند و این کار به کمک دستورهای getpgid() و setpgid() یا getpgrp() و setpgrp() انجام می‌پذیرد. در نظر داشته باشید که وقتی شل شما یک برنامه جدید را شروع می‌کند، آن برنامه ممکن است نیاز داشته باشد که چند پردازش دیگر درست عمل کنند. تمام این پردازش‌ها pgid مشابه پردازش اصلی را ارث‌بری می‌کنند. بنابراین ایده خوبی به نظر می‌رسد که هر زیرپردازش شل را در گروه مربوط به خودش قرار دهید و با این کار سازماندهی آنها را آسان‌تر کنید. توجه: هر وقت زیرپردازش‌ای را به یک گروه جدید مخصوص به خودش منتقل می‌کنید، pgid آن باید با pid برابر باشد.

۳.۶ ترمینال پیش‌زمینه

هر ترمینال یک pgid مربوط به گروه پردازش‌های پیش‌زمینه^{۱۸} دارد. وقتی CTRL-C را تایپ می‌کنید، ترمینال یک سیگنال به هر پردازش‌ای که در گروه پردازش‌های پیش‌زمینه باشد ارسال می‌کند. می‌توانید گروه پردازش‌هایی که در پیش‌زمینه ترمینال قرار دارند را با دستور زیر تغییر دهید:

```
1 tcsetpgrp(int fd, pid_t pgrp)
```

در حالت ورودی استاندارد، fd باید صفر باشد.

۴.۶ آشنایی با سیگنال‌ها

سیگنال‌ها پیام‌های ناهمگامی^{۱۹} هستند که به پردازش‌ها فرستاده می‌شوند و با شماره سیگنال شناسایی می‌شوند. اسامی سیگنال‌ها با SIG آغاز می‌شود و معمولاً با عملکردشان متناسب است. برخی از سیگنال‌ها عبارتند از:

- SIGINT: با تایپ CTRL-C فرستاده می‌شود و به صورت پیش‌فرض برنامه را متوقف می‌کند.

¹⁸foreground

¹⁹asynchronous

- **SIGTERM** : کلید میانبری برای این سیگنال وجود ندارد و بصورت پیش فرض برنامه را متوقف می‌کند؛ اما برنامه‌ها به صورت جدی تری نسبت به این سیگنال واکنش می‌دهند.
- **SIGQUIT** : با تایپ `\-CTRL` فرستاده می‌شود و مشابه **SIGTERM** رفتار می‌کند. علاوه بر آن این سیگنال تلاش می‌کند که یک برگرفت از حافظه^{۲۰} قبل از خروج تولید کند.
- **SIGKILL** : کلید میانبری برای این سیگنال وجود ندارد. همچنین این سیگنال به اجبار برنامه را متوقف می‌کند و نمی‌تواند توسط برنامه لغو^{۲۱} شود؛ در حالی که بیشتر سیگنال‌ها می‌توانند توسط برنامه نادیده گرفته شوند.
- **SIGTSTP** : با تایپ `Z-CTRL` فرستاده می‌شود و بصورت پیش فرض برنامه را موقتاً متوقف می‌کند. در بش اگر این کار را انجام دهید، برنامه کنونی موقتاً متوقف شده و بش شروع به دریافت فرمان‌های بیشتر می‌کند.
- **SIGCONT** : اگر دستور `fg %NUMBER` یا `fg` را در بش وارد کنید، این سیگنال فرستاده می‌شود. این سیگنال اجرای یک برنامه موقتاً متوقف شده را ادامه می‌دهد.
- **SIGTTIN** : این سیگنال به پردازنده پس زمینه‌ای که تلاش به خواندن ورودی از صفحه کلید می‌کند فرستاده می‌شود. چون پردازنده‌های پس زمینه نمی‌توانند ورودی از صفحه کلید را بخوانند، به صورت پیش فرض این سیگنال برنامه را موقتاً متوقف می‌کند. وقتی شما پردازنده پس زمینه را با **SIGCONT** به حالت ادامه اجرا در می‌آورید و آن را به حالت پیش زمینه می‌برید، می‌تواند مجدداً ورودی را از صفحه کلید بخواند.
- **SIGTTOU** : این سیگنال به پردازنده پس زمینه‌ای که تلاش به نوشتن خروجی در ترمینال می‌کند، فرستاده می‌شود. در حالیکه پردازنده پیش زمینه دیگری وجود دارد که در حال استفاده از ترمینال است. همچنین این سیگنال به صورت پیش فرض مشابه **SIGTTIN** عمل می‌کند.

برای ارسال سیگنال در شل می‌توانید از دستور زیر استفاده کنید:

```
1 kill -XXX PID
```

که XXX در آن پسوند نام سیگنال است.

برای مثال دستور زیر سیگنال **SIGTERM** را به پردازنده با شناسه ۱۰۰ ارسال می‌کند:

```
1 kill -TERM 100
```

در زبان C برای تغییر نحوه برخورد پردازنده کنونی با یک سیگنال، می‌توانید از تابع `sigaction` استفاده کنید. خود شل باید بیشتر سیگنال‌ها را نادیده بگیرد ولی زیرپردازنده‌های آن براساس یک عملکرد پیش فرض نسبت به آنها پاسخ دهند. مثلاً شل باید سیگنال **SIGTTOU** را نادیده بگیرد اما زیرپردازنده‌ها باید پاسخ دهند. توجه کنید که پردازنده‌های فورک شده از گرداننده سیگنال^{۲۲} پردازنده اصلی ارث‌بری می‌کنند. برای کسب اطلاعات بیشتر می‌توانید به `man 2 sigaction` و `man 7 signal` مراجعه کنید.

همچنین اطمینان حاصل کنید که ثابت‌های^{۲۳} `SIG_DFL` و `SIG_IGN` را بررسی کرده باشید. وظیفه اصلی شما در این قسمت این است که مطمئن شوید هر برنامه در گروه پردازنده خودش شروع به کار می‌کند. همچنین وقتی یک پردازنده شروع به کار می‌کند، گروه پردازنده‌اش باید در حالت پیش زمینه قرار بگیرد. علاوه بر این نحوه برخورد شل با سیگنال‌ها باید به درستی پیاده‌سازی شود، به طور خاص سیگنال‌های **SIGQUIT**، **SIGINT** و **SIGTSTP** باید تنها بر روی برنامه پیش زمینه اثر بگذارد نه شل پس زمینه.

²⁰ core dump

²¹ override

²² signal handler

²³ constant

۷ پردازش پس‌زمینه

تاکنون شل به گونه‌ای بوده است که قبل از شروع برنامه بعدی منتظر اتمام برنامه‌های قبلی می‌ماند. بسیاری از شل‌ها امکان اجرای یک دستور در پس‌زمینه را با قرار دادن علامت & در انتهای خط فرمان فراهم می‌سازند. پس از شروع برنامه پس‌زمینه، شل به شما اجازه می‌دهد که پردازش‌های بیشتری را بدون انتظار جهت اتمام پردازش پس‌زمینه، شروع کنید. شل را به گونه‌ای تغییر دهید که فرمان‌هایی که با قالب مذکور وارد می‌شوند را در پس‌زمینه اجرا کند. توجه کنید که تنها باید پشتیبانی از پردازش‌های پس‌زمینه را فراهم کنید و نیازی به پیاده‌سازی فرمان‌های داخلی نیست. پس از پیاده‌سازی این قسمت باید قادر باشید دستوراتی مشابه دستور زیر را اجرا کنید:

```
1 sleep 60 &
```

و بالا‌فاصله پس از اجرای آن با استفاده از دستور ps پردازشی در حال اجرا در پس‌زمینه را ببینید. همچنین باید دستور جدید داخلی wait را اضافه کنید. این دستور صبر می‌کند تا تمام کارهای پس‌زمینه تمام شوند و سپس به حالت عادی باز می‌گردد. یعنی اگر پردازشی در پس‌زمینه در حال اجراست، وقتی این دستور را وارد کنیم، شل مدتی بدون چاپ خروجی می‌ماند و پس از اتمام کار پردازش‌های پس‌زمینه، مجدداً اجازه‌ی وارد کردن دستورات بعدی را به ما می‌دهد. برای درک بهتر این دستور می‌توانید آن را در شل ماشین خود امتحان کنید. می‌توانید فرض کنید که همواره پیرامون نویسه & فاصله وجود دارد. همچنین فرض کنید که این نویسه آخرین نشان ۲۴ در آن خط فرمان است.

۸ Further Reading: Pipe

یکی از عملگرهایی که در بسیاری از شل‌ها وجود دارد، پایپ (|) است. این عملگر بین دو دستور می‌آید و خروجی استاندارد ۲۵ دستور قبلی را به ورودی استاندارد ۲۶ دستور بعدی وصل می‌کند. همچنین می‌توان به صورت زنجیر وار از این عملگر استفاده کرد. به عنوان مثال دستور

```
1 cat hello.txt | grep search | wc -l
```

به صورت خلاصه تعداد خط‌هایی از فایل hello.txt را می‌شمارد که عبارت search را داشته باشند. این کار با انتقال خروجی هر دستور به ورودی دستور بعدی انجام می‌گیرد. یکی از روش‌های پیاده‌سازی پایپ، استفاده از named pipe سیستم عامل است. به عنوان مثال می‌توان دستور فوق را به صورت زیر پیاده‌سازی کرد:

```
1 mkfifo first.pipe second.pipe
2 cat hello.txt > first.pipe &
3 grep search < first.pipe > second.pipe &
4 wc -l < second.pipe
```

اما مشکلی که در این روش وجود دارد این است که برای هر پایپ باید یک named pipe جدا ساخته شود. بدین منظور یکی از روش‌های دیگر که استفاده می‌شود استفاده از anonymous pipe است. در این حالت به کمک فراخوانی سیستمی pipe یک پایپ می‌سازیم. سپس به کمک دستور dup قسمت writer پایپ را به خروجی استاندارد دستور اول، و قسمت reader آنرا به ورودی استاندارد دستور دوم می‌چسبانیم. این کار را برای هر تعداد دستور می‌توان به صورت مشابه انجام داد. بدین ترتیب می‌توان به دو صورت پایپ را پیاده‌سازی کرد.

پیاده‌سازی پایپ در شل‌های مختلف می‌تواند متفاوت باشد. به عنوان مثال دستور زیر را در نظر بگیرید:

```
1 exit | exit
```

²⁴token

²⁵stdout

²⁶stdin

این دستور را یک بار در bash و باری دیگر در zsh امتحان کنید. با توجه به نتیجه‌ی حاصل این دو شل چگونه fork و پایپ کردن پراسس‌ها را پیاده سازی کرده‌اند؟
در ادامه دستور زیر را در ترمینال خود وارد کنید:

```
1 curl "https://www.sharif.edu/" | wc -l
```

خروجی دستور شبیه چنین چیزی است:

| % Total | % Received | % Xferd | Average Speed | | Time | Time | Time | Current |
|---------|------------|---------|---------------|--------|-------|---------|------|---------|
| | | | Dload | Upload | Total | Spent | Left | Speed |
| 100 | 263k | 0 | 263k | 0 | 0 | 238k | 0 | --- |
| 14706 | | | | | | 0:00:01 | --- | 238k |

همان طور که مشخص است curl وضعیتی از دانلود صفحه‌ی داده شده را در ترمینال نشان می‌دهد و زمانی که دانلود تمام می‌شود، تعداد خط‌های صفحه در ترمینال نشان داده می‌شود. این وضعیت دانلود چطور در ترمینال نشان داده می‌شود با اینکه خروجی استاندارد در ورودی استاندارد دستور بعدی pipe شده است؟
حال دستور

```
1 curl "https://www.sharif.edu/"
```

را اجرا کنید. متوجه می‌شوید که این بار اصلاً وضعیت دانلود نشان داده نمی‌شود. تحقیق کنید که چه طور curl متوجه می‌شود که خروجی استاندارد در جایی pipe یا redirect شده است.

۹ داوری

به نکات زیر دقت کنید:

- توجه کنید که تعدادی تست به همراه خروجی مورد انتظار در اختیار شما قرار گرفته است که شما می‌توانید از آنها برای بررسی عملکرد صحیح کد خود استفاده کنید. برای نمره‌دهی نهایی تعدادی تست دیگر نیز در هنگام تحویل به برنامه‌ی شما داده خواهد شد که برای دریافت نمره‌ی کامل باید کد شما به آن‌ها پاسخ درست دهد.
- قسمت مربوط به کار با سیگنال‌ها داوری خودکار نداشته و پس از پایان تمرین نمره‌دهی خواهد شد.

۱۰ تحویل‌دادنی‌ها

برای ارسال این تمرین باید کدهایی که در اختیار شما قرار داده‌ایم را به پوشه‌ای به نام `hw1-std-id` انتقال داده و پس از ایجاد تغییرات مورد نظر تمامی پرونده‌ها به خصوص پرونده‌های `h` و `c` و `Makefile` را در `quera` آپلود کنید.