به نام خدا

درس سیستم‌های عامل

نیم‌سال دوم ۰۲-۰۱

دانشکدهٔ مهندسی کامپیوتر

دانشگاه صنعتی شریف

مدرس **مهدی خرازی**

تمرین **یک فردی**

موضوع **ساخت یک شل**

موعد تحویل **ساعت ۲۳:۵۹ پنج‌شنبه ۴ اسفند ۱۴۰۱**

با سپاس از دستیاران آموزشی **محمد حدادیان، مریم ابراهیم‌زاده، صبا هاشمی و هیربد بهنام**

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

در این تمرین، یک شل[1] مشابه بش[2] در ماشین مجازی 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 پردازه‌ها می‌شود، ارائه دهد. شل‌ها می‌توانند تعاملی[3] یا غیرتعاملی[4] 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.
هستهٔ سیستم عامل[5] مستندات بسیارمفیدی جهت ساخت شل‌ها فراهم نموده است. با ساخت یک شل برای خودتان با این رابط‌ها بیشتر آشنا خواهید شد و احتمالاً اطلاعاتی پیرامون سایر شل‌ها نیز کسب خواهید نمود.

۱ راه‌اندازی مقدمات

به ماشین مجازی خود در Vagrant وارد شوید و دستورهای زیر را اجرا کنید:

```
$ cd /home/vagrant/code/handouts
$ git pull origin
```

هم چنین می‌توانید در پوشه‌ی تمرین‌های فردی خود از دستور

```
$ git pull handouts master
```

برای بارگیری پرونده‌های مربوط به این تمرین از مخزن handouts استفاده کنید.
ما کدهای اولیه مورد نیاز برای ساخت شل را به همراه یک میک‌فایل[6] در پوشهٔ `hw1` در اختیار شما قرار داده‌ایم. 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:

```
$ make
$ ./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[7] برای فرمان‌های داخلی[8] 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 پوشهٔ 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.
پس از افزودن این دو فرمان کد شل خود را پوش کنید:

```
$ git add shell.c
$ git commit -m "Add basic functionality into the shell"
$ git push origin master
```

توجه: به عنوان یک قانون کلی، بهتر است در تمام مراحل کد خود را به صورت مرتّب کامیت کنید، زیرا با این کار می‌توانید در صورت نیاز، به نسخه‌های قبلی از کد خود بازگردید.

[1] shell
[2] bash
[3] interactive
[4] non-interactive
[5] kernel
[6] makefile
[7] dispatcher
[8] built-in commands

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 آن را اجرا کند. اولین کلمهٔ فرمان، نام برنامه و مابقی کلمات ورودی‌های برنامه خواهند بود. فعلاً می‌توانید اینگونه درنظر بگیرید که کلمهٔ اول، نشانی کامل[9] 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 سادهٔ 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:

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

وقتی شل بخواهد یک برنامه را اجرا کند، باید با فراخوانی یکی از توابع خانوادهٔ `exec` یک پردازهٔ فرزند فورک[10] کند. همچنین پردازهٔ والد باید تا اتمام کار پردازهٔ 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 جملهٔ آن‌ها برنامهٔ شل، به یک مجموعه از متغیرهای محلی[11] دسترسی دارند که به صورت یک جدول هش[12] 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):

```
$ echo $PATH
```

که خروجی آن مشابه زیر است:

```
$ /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 سادهٔ آن نیز اجرا کند.
توجه ۱: باید کماکان از نوشتن مسیرکامل 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 **`[process] > [file]`** به شل می‌گوید که خروجی استاندارد پردازه باید در یک پرونده نوشته شود. به طور مشابه دستور **`[process] < [file]`** به شل می‌گوید که محتوای پرونده را به عنوان ورودی استاندارد پردازه به کار ببرد. **شما باید شل خود را به گونه‌ای تغییر دهید که از هدایت[13] کردن ورودی و خروجی استاندارد به پرونده‌ها پشتیبانی کند.**
نیازی به پشتیبانی از هدایت فرمان‌های داخلی شل (مثلاً `pwd`) ندارید. همچنین نیازی به پشتیبانی هدایت از `stderr` یا الحاق‌کردن[14] به پرونده‌ها (**`[process] >> [file]`**) نیست.

[9] full path
[10] fork
[11] environment variables
[12] HashTable
[13] redirect
[14] append

فرض کنید که همواره پیرامون دو نویسهٔ < و > فضای خالی وجود دارد.
توجّه 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 پردازه‌ها 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 زیرپردازه[15]های شل کار می‌کنند. برای مثال با فشردن **CTRL-C** یک سیگنال SIGINT ارسال می‌شود که معمولاً برنامهٔ درحال اجرا را متوقف می‌کند و با فشردن **CTRL-Z** یک سیگنال SIGTSTP ارسال می‌شود که معمولاً برنامهٔ درحال اجرا را به حالت پس‌زمینه[16] می‌برد. اگر درحال حاضر این کلیدها را بر روی شل خود به کار ببرید، سیگنال‌ها به پردازهٔ خود شل ارسال می‌شوند و این آن چیزی نیست که ما به دنبالش هستیم. مثلاً وقتی شما با فشردن **CTRL-Z** می‌خواهید یک زیرپردازه از شل را متوقف کنید، خود شل نیز متوقف می‌شود. در حالی که ما می‌خواهیم سیگنال‌ها فقط روی زیرپردازه‌های 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 پردازه‌های 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:

```
vagrant@development  ~ $ ps
  PID TTY          TIME CMD
2672 pts/0    00:00:00 bash
13096 pts/0    00:00:00 ps
vagrant@development ~  $ sh
$ ps
PID TTY          TIME CMD
2672 pts/0    00:00:00 bash
13097 pts/0    00:00:00 sh
13098 pts/0    00:00:00 ps
```

در این قسمت ما با استفاده از دستور sh، یک شل بورن [17] درون شل بش خود آغاز کرده‌ایم.

```
$ cat
hello
hello
^Z
[1] + Stopped      cat
$ ps
PID TTY          TIME CMD
2672 pts/0    00:00:00 bash
13097 pts/0    00:00:00 sh
13102 pts/0    00:00:00 cat
13103 pts/0    00:00:00 ps
```

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

```
$ jobs
[1] + Stopped      cat
$ fg
cat
```

[15] subprocess
[16] background
[17] Bourne shell

```
world
world
^C
$ ps
PID TTY          TIME CMD
2672 pts/0    00:00:00 bash
13097 pts/0    00:00:00 sh
13237 pts/0    00:00:00 ps
$ exit
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. همان‌طور 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 نحوهٔ انجام این کار، قصد داریم پیرامون چند مفهوم در سیستم‌عامل‌ها بیشتر صحبت کنیم.

۲.۶ گروه‌های پردازه

می‌دانید که هر پردازه یک pid یکتا دارد. اما هر پردازه یک pgid مخصوص گروه خود را هم داراست که یکتا نیست و به صورت پیش فرض همان pgid پردازهٔ والد است. پردازه‌ها می‌توانند شناسهٔ گروه خود را دریافت و مقداردهی کنند و این کار به کمک دستورهای **getpgid()** و **setpgid()** یا **getpgrp()** و **setpgrp()** انجام می‌پذیرد. درنظر داشته باشید که وقتی شل شما یک برنامهٔ جدید را شروع می‌کند، آن برنامه ممکن است نیاز داشته باشد که چند پردازهٔ دیگر درست عمل کنند. تمام این پردازه‌ها pgid مشابه پردازهٔ اصلی را ارث‌بری می‌کنند. بنابراین ایدهٔ خوبی به نظر می‌رسد که هر زیرپردازهٔ شل را در گروه مربوط به خودش قرار دهید و با این کار سازماندهی آنها را آسان‌تر کنید.
توجه: هروقت زیرپردازه‌ای 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 پردازه‌های پیش‌زمینه[۱۸] دارد. وقتی **CTRL-C** را تایپ می‌کنید، ترمینال یک سیگنال به هر پردازه‌ای که در گروه پردازه‌های پیش‌زمینه باشد ارسال می‌کند. می‌توانید گروه پردازه‌هایی که در پیش‌زمینهٔ ترمینال قرار دارند را با دستور زیر تغییر دهید:

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

درحالت ورودی استاندارد، fd باید صفر باشد.

۴.۶ آشنایی با سیگنال‌ها

سیگنال‌ها پیام‌های ناهمگامی[۱۹] هستند که به پردازه‌ها فرستاده می‌شوند و با شمارهٔ سیگنال شناسایی می‌شوند. اسامی سیگنال‌ها با SIG آغاز می‌شود و معمولا با عملکردشان متناسب است. برخی از سیگنال‌ها عبارتند از:

- SIGINT : با تایپ **CTRL-C** فرستاده می‌شود و به‌صورت پیش‌فرض برنامه را متوقف می‌کند.
- SIGTERM : کلید میانبری برای این سیگنال وجود ندارد و بصورت پیش‌فرض برنامه را متوقف می‌کند؛ اما برنامه‌ها به‌صورت جدی‌تری 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؛ درحالی که بیشتر سیگنال‌ها می‌توانند توسط برنامه نادیده گرفته شوند.

[18] foreground
[19] asynchronous
[20] core dump
[21] override

- SIGTSTP : با تایپ **CTRL-Z** فرستاده می‌شود و بصورت پیش‌فرض برنامه را موقتاً متوقف می‌کند. در بش اگر این کار را انجام دهید، برنامهٔ کنونی موقتاً متوقف شده و بش شروع به دریافت فرمان‌های بیشتر می‌کند.
- SIGCONT : اگر دستور **fg** یا **fg %NUMBER** را در بش وارد کنید، این سیگنال فرستاده می‌شود. این سیگنال اجرای یک برنامهٔ موقتاً متوقف‌شده را ادامه می دهد.
- SIGTTIN : این سیگنال به پردازهٔ پس‌زمینه‌ای که تلاش به خواندن ورودی از صفحه کلید می‌کند فرستاده می‌شود. چون پردازه‌های پس‌زمینه نمی‌توانند ورودی از صفحه کلید را بخوانند، به صورت پیش‌فرض این سیگنال برنامه را موقتاً متوقف می‌کند. وقتی شما پردازه پس‌زمینه را با SIGCONT به حالت ادامهٔ اجرا در می‌آورید و آن را به حالت پیش‌زمینه می‌برید، می‌تواند مجدداً ورودی را از صفحه کلید بخواند.
- SIGTTOU : این سیگنال به پردازهٔ پس زمینه‌ای که تلاش به نوشتن خروجی در ترمینال می‌کند، فرستاده می‌شود درحالیکه پردازهٔ پیش‌زمینه دیگری وجود دارد که درحال استفاده از ترمینال است. همچنین این سیگنال به صورت پیش‌فرض مشابه SIGTTIN عمل می‌کند.

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

```
kill -XXX PID
```

که XXX در آن پسوند نام سیگنال است.
برای مثال دستور زیر سیگنال SIGTERM را به پردازهٔ با شناسهٔ ۱۰۰ ارسال می‌کند:

```
kill -TERM 100
```

در زبان C برای تغییر نحوه برخورد پردازه‌ی کنونی با یک سیگنال، می‌توانید از تابع sigaction استفاده کنید. خود شل باید بیشتر سیگنال‌ها را نادیده بگیرد ولی زیرپردازه‌های آن براساس یک عملکرد پیش‌فرض نسبت به آنها پاسخ دهند. مثلاً شل باید سیگنال SIGTTOU را نادیده بگیرد اما زیرپردازه‌ها باید پاسخ دهند.
توجه کنید که پردازه‌های فورک‌شده از گردانندهٔ سیگنال[۲۲] پردازهٔ 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_IGN و SIG_DFL را بررسی کرده باشید.
وظیفهٔ اصلی شما در این قسمت این است که مطمئن شوید هر برنامه در گروه پردازهٔ خودش شروع به کار می‌کند. هم‌چنین وقتی یک پردازه شروع به کار می‌کند، گروه پردازه‌اش باید در حالت پیش‌زمینه قرار بگیرد. علاوه بر این نحوه‌ی برخورد شل با سیگنال‌ها باید به درستی پیاده‌سازی شود، به طور خاص سیگنال‌های SIGINT ، SIGQUIT و SIGTSTP باید تنها بر روی برنامهٔ پیش‌زمینه اثر بگذارد نه شل پس‌زمینه.

۷ پردازش پس‌زمینه

تاکنون شل به‌گونه‌ای بوده است که قبل از شروع برنامهٔ 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 برنامهٔ پس‌زمینه، شل به شما اجازه می‌دهد که پردازه‌های بیشتری را بدون انتظار جهت اتمام پردازهٔ پس‌زمینه، شروع کنید.
شل را به‌گونه‌ای تغییر دهید که فرمان‌هایی که با قالب مذکور وارد می‌شوند را در پس‌زمینه اجرا کند. توجه کنید که تنها باید پشتیبانی از پردازه‌های 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:

```
sleep 60 &
```

و بالافاصله پس از اجرای آن با استفاده از دستور ps پردازه‌ی 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 پردازه‌ای در پس‌زمینه در حال اجراست، وقتی این دستور را وارد کنیم، شل مدتی بدون چاپ خروجی می‌ماند و پس از اتمام کار پردازه‌های 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.

[22] signal handler
[23] constant

می‌توانید فرض کنید که همواره پیرامون نویسهٔ & 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

```
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 پیاده سازی پایپ، استفاده از named pipeهای سیستم عامل است. به عنوان مثال می‌توان دستور فوق را به صورت زیر پیاده سازی کرد:

```
mkfifo first.pipe second.pipe
cat hello.txt > first.pipe &
grep search < first.pipe > second.pipe &
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 پیاده سازی کرد.
پیاده‌سازی پایپ در شل‌های مختلف می‌تواند متفاوت باشد. به عنوان مثال دستور زیر را در نظر بگیرید:

```
exit | exit
```

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:

```
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 --:--:--  0:00:01 --:--:--  238k
14706
```

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 ترمیال نشان داده می‌شود با اینکه خروجی استاندارد در ورودی استاندارد دستور بعدی pipe شده است؟
حال دستور

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

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

[24] token
[25] stdout
[26] stdin

۹ داوری

در استفاده از سیستم داوری به نکات زیر دقت کنید:

- سیستم داوری به صورت خودکار کدهای موجود برروی انشعاب[۲۷] `master` از مخزن‌های شما را بررسی خواهد کرد و این کار حتی اگر تمرین را با تاخیر می‌فرستید هم انجام خواهد شد. در واقع سیستم داوری با هر پوش جدید برروی مخزن شما، کار داوری را شروع می‌کند.
- اگر تمرین خود را با تاخیر می‌فرستید، سامانهٔ داوری نمرهٔ بدون تاخیر را اعلام می‌کند. برای اطلاع از قوانین ارسال با تاخیر تمرینات به قوانین درس مراجعه کنید.
- نمره‌ای که در نهایت سامانهٔ داوری به شما می‌دهد، بیشینهٔ نمرهٔ شما در کامیت‌های مختلف نیست بلکه نمرهٔ آخرین کامیت شماست. هرگونه تحویل‌دادنی‌های دیگر مانند مستند گزارش که توسط سامانهٔ داوری نمره‌دهی نمی‌شوند مبتنی بر آخرین کامیت شما نمره‌دهی خواهند شد.
- از پوش کردن کامیت‌های متعدد در زمان کوتاه پرهیز کنید و در استفاده از سامانهٔ داوری دقت کافی را مبذول فرمایید؛ در غیر این‌صورت هرگونه مشکل در سامانهٔ داوری که منجر به کم‌شدن زمان مفید شما تا ضرب‌الاجل ارسال تمرین شود، بر عهدهٔ خودتان خواهد بود.
- دقت بفرمایید که سامانهٔ داوری ابزاری برای آزمودن عملکردها و قابلیت‌های کد شما نیست و صرفاً ابزاری برای نمره‌دهی است. بنابراین لطفاً از سامانهٔ داوری به عنوان آزمونگر صحت عملکرد کد خود استفاده نکنید و روی سیستم خودتان آزمون‌های لازم را انجام داده و سپس پوش کنید. بدین ترتیب، نیازی به پوش کردن متوالی هم پیدا نخواهید کرد.
- 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` در مخزن شخصی خود انتقال داده و پس از ایجاد تغییرات مورد نظر تنها پرونده‌های `.h` و `.c` و `Makefile` را پوش کنید. لطفاً به هیچ عنوان پرونده‌های `.o` و پرونده‌های اجرایی را پوش نکنید. (برای این موضوع می‌توانید از یک فایل `.gitignore` استفاده کنید) تنها تحویل‌دادنی این تمرین، کد مربوط به یک شل است که می‌بایست قابلیت‌های مطرح‌شده در قسمت‌های قبل را داشته باشد. توجه کنید که نام پوشه‌ی حاوی پرونده‌ها باید حتما `hw1` (با حروف کوچک) باشد و در غیر این صورت کد شما داوری نخواهد شد.

توجه: شما موظف هستید که هر مرحله از تغییرات کدتان را کامیت کنید تا در صورت بروز مشکل، بتوانید به نسخه‌های قبلی بازگردید. می‌توانید تمرین خود را جهت نمره‌دهی با دستور زیر ارسال کنید:

```
git push origin master
```

در صورت داشتن هرگونه سوال در رابطه با درس و تمرینات، سوال خود را در سرور دیسکورد درس و در کانال مرتبط با سوال خودتان مطرح کنید. همچنین اگر در استفاده از سامانهٔ طرشت به مشکلی برخوردید، آن را از طریق این ایمیل مطرح کنید.

[27] branch