به نام خدا

درس سیستم‌های عامل

نیم‌سال دوم ۰۰-۹۹

دانشکدهٔ مهندسی کامپیوتر

دانشگاه صنعتی شریف

مدرس **مهدی خرازی**

تمرین **یک فردی**

موضوع **ساخت یک شل**

موعد تحویل **ساعت ۲۳:۵۹ سه‌شنبه ۱۲ اسفند ۱۳۹۹**

با سپاس از دستیاران آموزشی **محمد حدادیان و مریم ابراهیم‌زاده**

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 و هدف از این کار آن است که کاربر بتواند برنامه‌های خود را مدیریت و اجرا کند. هستهٔ سیستم عامل[3] مستندات بسیارمفیدی جهت ساخت شل‌ها فراهم نموده است. شل‌ها می‌توانند تعاملی[4] یا غیرتعاملی[5] باشند. به عنوان مثال، هنگامی که اسکریپت بش را اجرا می‌کنید، از تعامل متقابل بش استفاده می‌کنید. توجه به این نکته ضروری است که بش به طور پیش‌فرض غیرتعاملی است. **`bash -i`** را برای یک شل تعاملی اجرا کنید. 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 ورود کنید و دستورهای زیر را اجرا کنید:

```
$ cd /home/vagrant/code/ce424-992-handouts
$ git pull origin
```

ما کدهای اولیه مورد نیاز برای ساخت شل را به همراه یک میک‌فایل[6] در پوشهٔ `hw1` در اختیار شما قرار داده‌ایم. کدهای در دسترس، شامل قطعه برنامه‌ای هستند که یک رشته[7] 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[8] برای فرمان‌های[9] داخلی[10] 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[11] 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) kernel
4) interactive
5) non-interactive
6) makefile
7) string
8) dispatcher
9) command
10) built-in
11) push

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 همان نشانی کامل[۱۲] 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 `tokenizer.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[۱۳] 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 مسیرکامل هر برنامه را وارد می‌کردید. خوشبختانه هر برنامه‌ای و از جملهٔ آن‌ها برنامهٔ شل، به یک مجموعه از متغیرهای محلی دسترسی دارند که به صورت یک جدول هش[۱۴] 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 `wc` 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[۱۵] 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[۱۶] به پرونده‌ها (**`[process] >> [file]`**) نیست.
فرض کنید که همواره پیرامون دو نویسهٔ < و > فضای خالی وجود دارد.
توجّه کنید که " **`[file] >`** " یا " **`> [file]`** " به عنوان ورودی به برنامه پاس داده نمی‌شوند.

12) full path
13) fork
14) HashTable
15) redirect
16) append

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[۱۷]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[۱۸] 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. ما 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.

۱.۶ 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.

۲.۶ ترمینال پیش‌زمینه

هر ترمینال یک pgid مربوط به گروه پردازه‌های پیش‌زمینه[۱۹] 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.

۳.۶ آشنایی با سیگنال‌ها

سیگنال‌ها پیام‌های ناهمگامی[۲۰] هستند که به پردازه‌ها فرستاده می‌شوند و با شمارهٔ سیگنال شناسایی می‌شوند. گاهی اوقات نیز اسامی سیگنال‌ها کاملاً با عملکردشان متناسب است و با SIG آغاز می‌شود. برخی از سیگنال‌ها عبارتند از:

- SIGINT : با تایپ **CTRL-C** فرستاده می‌شود و به‌صورت پیش فرض برنامه را متوقف می‌کند.
- SIGQUIT : با تایپ **CTRL-** فرستاده می‌شود و بصورت پیش‌فرض برنامه را متوقف می‌کند؛ اما برنامه‌ها به‌صورت جدی‌تری نسبت به این سیگنال واکنش می‌دهند. همچنین این سیگنال تلاش می کند که یک برگرفت از حافظه[۲۱] 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[۲۲]شود؛ درحالی‌که بیشتر سیگنال‌ها می‌توانند توسط برنامه نادیده گرفته شوند.
- SIGTERM : کلید میانبری برای این سیگنال وجود ندارد و مشابه SIGQUIT رفتار می‌کند.
- 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.

17) subprocess
18) background
19) foreground
20) asynchronous
21) core dump
22) override

- 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 می‌توانید از تابع signal استفاده کنید که نحوهٔ برخورد با پردازهٔ کنونی را تغییر دهید. 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[۲۳] 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[۲۴] SIG_IGN و SIG_DFL را بررسی کرده باشید.
وظیفهٔ اصلی شما این است که مطمئن شوید هر برنامه در گروه پردازهٔ خودش شروع به کار می‌کند. وقتی شما یک پردازه را شروع به کار می‌کنید، گروه پردازه‌اش باید به حالت پیش‌زمینه برود. همچنین سیگنال توقف باید تنها بر روی برنامهٔ پیش‌زمینه اثر بگذارد نه شل پس‌زمینه.

۷ پردازش پس‌زمنیه

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.
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.
می‌توانید فرض کنید که همواره پیرامون نویسهٔ & فاصله وجود دارد. همچنین فرض کنید که این نویسه آخرین نشان[۲۵] در آن خط فرمان است.

۸ داوری

در استفاده از سیستم داوری به نکات زیر دقت کنید:

- سیستم داوری به صورت خودکار کدهای موجود برروی انشعاب 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.

23) signal handler
24) constant
25) token

- اگر که 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.
- دقت بفرمایید که این سیستم جهت نمره‌دهی به تمرین است، بنابراین ابتدا از صحت کد خود مطمئن شوید سپس برای تصحیح کد، آن را ارسال کنید.

۹ 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. تنها تحویل‌دادنی این تمرین، کد مربوط به یک شل است که می‌بایست قابلیت‌های مطرح‌شده در قسمت‌های قبل را داشته باشد.
توجه: شما موظف هستید که هر مرحله از تغییرات کدتان را کامیت کنید تا در صورت بروز مشکل، بتوانید به نسخه‌های قبلی بازگردید.
می توانید تمرین خود را جهت نمره دهی 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 بر خوردید، آن را از طریق این ایمیل مطرح کنید.