به نام خدا

درس سیستم‌های عامل

نیم‌سال دوم ۹۹-۹۸

دانشکدهٔ مهندسی کامپیوتر

دانشگاه صنعتی شریف

مدرس **مهدی خرازی**

تمرین **یک**

موضوع **ساخت یک shell**

موعد تحویل **ساعت ۲۳:۵۹ ۹۸/۱۲/۵**

در این تمرین، یک shell مشابه bash در ماشین مجازی خود را خواهید ساخت و هدف از این کار آن است که کاربر بتواند برنامه‌های خود را مدیریت و اجرا کند. هسته‌ی سیستم عامل[۱] مستندات بسیارمفیدی جهت ساخت shell ها فراهم نموده است. shell ها می‌توانند تعاملی[۲] یا غیرتعاملی[۳] باشند.به عنوان مثال، هنگامی که اسکریپت bash را اجرا می‌کنید، از تعامل متقابل bash استفاده می‌کنید. توجه به این نکته ضروری است که bash به طور پیش‌فرض غیر تعاملی است. **bash -i** را برای یک shell تعاملی اجرا کنید. با ساخت یک shell برای خودتان با این رابط‌ها بیشتر آشنا خواهید شد و احتمالا اطلاعاتی پیرامون سایر shell ها 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/ce424-982-handouts
$ git pull origin
```

ما کدهای اولیه مورد نیاز برای ساخت shell را به همراه یک makefile در پوشه hw۱ در اختیار شما قرار داده‌ایم. کدهای در دسترس، شامل قطعه برنامه‌ای هستند که یک رشته[۴] را دریافت می‌کند و آن‌را به کلمات، تقسیم می‌کند. به منظور اجرای shell می‌بایست دستورهای زیر را اجرا کنید:

```
$ make
$ ./shell
```

همچنین به منظور خاتمه دادن به اجرای shell 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 shell شما یک dispatcher برای فرمان‌های[۵] داخلی[۶] دارد. در واقع هر shell یک مجموعه از فرمان‌های درونی دارد که کارکردهای مربوط به خود shell هستند و نه برنامه‌های خارجی. برای مثال فرمان quit به عنوان یک فرمان داخلی پیاده سازی شده است زیرا این فرمان خود shell را exit می‌کند.

این shell که هم اکنون در اختیار شماست تنها دو فرمان داخلی دارد. فرمان "؟"،که منوی راهنما را نشان می‌دهد و فرمان "quit"، که shell را exit می‌کند. در اولین بخش تمرین، شما قرار است که فرمان جدید pwd را به مجموعه فرمان‌های داخلی اضافه کنید، که پوشه فعلی را در خروجی استاندارد چاپ کند. سپس فرمانِ درونیِ جدیدِ cd را که یک ورودی از جنس مسیر (مثلا /path/to/directory) می‌گیرد و مسیرِ کاریِ فعلیِ shell را به آن آرگومان تغییر می‌دهد، به فرمان‌های درونی shell اضافه کنید.

راهنمایی: از chdir و getcwd برای ایجاد تغییرات لازم در فایل shell.c استفاده کنید.

پس از افزودن این دو فرمان کد shell خود را push کنید:

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

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

۳ اجرای برنامه

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

[۱] kernel
[۲] interactive
[۳] non-interactive
[۴] string
[۵] command
[۶] built-in

کند. اولین کلمه دستور نام برنامه و مابقی کلمات آرگومان‌های برنامه خواهند بود. فعلا می‌توانید اینگونه درنظر بگیرید که کلمه اول همان نشانی کامل [۷]برنامه است. بنابراین به جای اجرای wc می‌بایست /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 را(wc) پیاده‌سازی کنید. می بایست 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
```

راهنمایی : وقتی shell نیاز دارد که یک برنامه را اجرا کند، می بایست یک پردازه فرزند fork کند، که درواقع یکی از توابع exec را برای اجرای برنامه جدید فراخوانی می‌کند. پردازه والد می‌بایست صبر کند تا پردازه فرزند به اتمام برسد و سپس منتظر فرمان‌های بعدی باشد.

۴ تفکیک‌پذیری مسیرها به کمک PATH

احتمالا تاکنون متوجه شده‌اید که تست کردن shell درقسمت قبل بسیار سخت بود، زیرا می‌بایست مسیرکامل هر برنامه را تایپ می‌کردید. خوشبختانه هر برنامه‌ای و از جمله آن‌ها برنامه shell ، به یک مجموعه از متغیرهای محلی دسترسی دارد که به صورت یک **HashTable** از رشته‌های key و value سازماندهی شده‌اند. یکی از این متغیرهای محلی متغیر PATH می‌باشد. شما می‌توانید این متغیر را بر روی ماشین مجازی خود چاپ کنید.(توجه کنید که از bash برای این قسمت استفاده کنید نه از shell دست‌ساز خودتان).

```
$ echo $PATH
```

که خروجی آن مشابه زیر است:

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

وقتی bash یا هر shell دیگری، یک برنامه مانند wc را اجرا می‌کند، می رود در هر مسیر از متغیر محلی PATH به دنبال برنامه ای با نام "wc" می‌گردد و اولین برنامه‌ای که پیدا می کند را اجرا می‌کند. هر پوشه در PATH با استفاده از علامت ":" از سایرین جدا می‌شود. حال می بایست shell دست ساز خود را چنان تغییر دهید که از متغیرمحلی PATH استفاده کند و برنامه را با نام ساده آن نیز اجرا کند.

نکته: می‌بایست کماکان از نوشتن مسیرکامل برنامه نیز پشتیبانی شود.

نکته: به هیچ وجه از "execvp" استفاده نکنید، زیرا در این صورت نمره ای به شما تعلق نخواهد گرفت. به جای آن می‌توانید از "execv" استفاده کنید و مسیرهای موردنیاز را خودتان بسازید.

[۷] path full

۵ خواندن ورودی و نوشتن خروجی با استفاده از فایل

گاهی اوقات بهتر است که یک برنامه به جای کار کردن با ورودی و خروجی استاندارد، ورودی خود را از یک فایل بخواند یا خروجی خود را در یک فایل بنویسد. دستور " [file] < [process] " به shell میگوید که خروجی استاندارد پردازه می‌بایست در یک فایل نوشته شود. به طور مشابه دستور " [file] > [process] " به shell میگوید که محتوای فایل را به عنوان ورودی استاندارد پردازه به کار ببرد. **شما می‌بایست shell خود را به گونه‌ای تغییر دهید که از redirect کردن stdin و stdout به فایل‌ها پشتیبانی کند.**

نیازی به پشتیبانی از redirect کردن فرمان‌های داخلی shell (مثلا pwd)ندارید. همچنین نیازی به پشتیبانی از redirect کردن از stderr یا append کردن به فایلها ([file] « [process]) نیست.

فرض کنید که همواره پیرامون دو کاراکتر خاص < و > کاراکتر space وجود دارد. توجّه کنید که " [file] <" یا " [file] >" به عنوان آرگومان به برنامه پاس داده نمی‌شوند.

۶ کار با سیگنال

اکثر shell ها این امکان را فراهم می‌کنند که با فشردن یک کلید خاص مانند Ctrl-C یا Ctrl-Z ، پردازه‌ها را pause ویا stop کنید. این keystroke ها با فرستادن سیگنال به زیرپردازه [۸] های shell کار می‌کنند. برای مثال با فشردن CTRL-C یک سیگنال SIGINT ارسال می‌شود که معمولاً برنامه درحال اجرا را stop می‌کند و با فشردن CTRL-Z یک سیگنال SIGTSTP ارسال می‌شود که معمولا برنامه درحال اجرا را به حالت پس‌زمینه [۹] می‌برد. اگر درحال حاضر این keystroke ها را برروی shell خود به کار ببرید، سیگنال‌ها به پردازه خود shell ارسال می‌شوند و این آن چیزی نیست که ما به دنبالش هستیم. مثلا وقتی شما با فشردن CTRL-Z میخواهید یک زیرپردازه از shell خود را متوقف کنید، خود shell نیز متوقف می‌شود. ما میخواهیم سیگنال‌هایی داشته باشیم که تنها برروی زیرپردازه هایی که shell ایجاد کرده است اثر بگذارند. قبل از توضیح نحوه انجام این کار، قصد داریم پیرامون چند مفهوم در سیستم‌عامل‌ها بیشتر صحبت کنیم.

۱.۶ گروه های پردازه

میدانید که هر پردازه‌ای یک pid یکتا دارد. اما هرپردازه‌ای یک pgid مخصوص گروه خود را داراست که یکتا نیست و به صورت پیش فرض همان pgid پردازه والد است. پردازه‌ها می‌توانند شناسه 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 shell شما یک برنامه جدید را شروع می‌کند، آن برنامه ممکن است نیاز داشته باشد که چند پردازه دیگر درست عمل کنند. تمام این پردازه‌ها pgid مشابه پردازه اصلی را ارث بری می‌کنند. بنابراین ایده خوبی به نظر می‌رسد که هر زیرپردازه shell را در گروه مربوط به خودش قرار دهید و با این کار سازماندهی آنها را آسان‌تر کنید.

توجه: وقتی هر زیرپردازه‌ای را در گروه پردازه خودش قرار دهید، pgid می‌بایست با pid برابر باشد.

۲.۶ Foreground terminal

هر ترمینال یک pgid مربوط به گروه پردازه‌های پیش‌زمینه [۱۰] دارد. وقتی CTRL-C را تایپ می‌کنید، ترمینال یک سیگنال به هر پردازه‌ای که در گروه پردازه‌های پیش زمینه باشد ارسال می‌کند. می‌توانید گروه پردازه‌هایی که در پیش‌زمینه ترمینال قرار دارند را با دستور زیر تغییر دهید:

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

درحالت ورودی استاندارد، fd می‌بایست صفر باشد.

۳.۶ آشنایی با سیگنالها

سیگنال‌ها پیام‌های غیرهم‌زمان [۱۱] هستند که به پردازه‌ها فرستاده می‌شوند و با شماره سیگنال شناسایی می‌شوند. گاهی اوقات نیز اسامی سیگنال‌ها کاملا با عملکردشان متناسب است و با SIG آغاز می‌شوند. برخی از سیگنال‌ها عبارتند از:

- SIGINT : با تایپ CTRL-C فرستاده می‌شود و بصورت پیش فرض برنامه را متوقف می‌کند.

[۸] subprocess
[۹] background
[۱۰] foreground
[۱۱] asynchronous

- SIGQUIT : با تایپ CTRL-\ فرستاده می‌شود و بصورت پیش فرض برنامه را متوقف می کند، اما برنامه‌ها بصورت جدی تری نسبت به این سیگنال واکنش نشان می‌دهند. همچنین این سیگنال تلاش می کند که یک core dump (پرونده‌ای از حافظه مستند شده در هنگام crash کردن یک برنامه)از برنامه، قبل از خروج آن تولید کند.
- SIGKILL : کلید میانبری برای این سیگنال وجود ندارد. همچنین این سیگنال به اجبار برنامه را متوقف می کند و نمی‌تواند توسط برنامه لغو [۱۲]شود، درحالی‌که بیشتر سیگنال‌ها می‌توانند توسط برنامه نادیده گرفته شوند.
- SIGTERM : کلید میانبری برای این سیگنال وجود ندارد و مشابه SIGQUIT رفتار می کند.
- SIGTSTP : با تایپ CTRL-Z فرستاده می‌شود و بصورت پیش‌فرض برنامه را موقتا متوقف می‌کند. در bash اگر این کار را انجام دهید، برنامه کنونی موقتا متوقف شده و bash شروع به دریافت فرمان‌های بیشتر می کند.
- SIGCONT : اگر دستور fg یا fg %NUMBER را در bash وارد کنید، سیگنال فرستاده می‌شود. این سیگنال اجرای برنامه‌ی موقتا متوقف شده را ادامه می دهد.
- SIGTTIN : این سیگنال به پردازه‌ی پس‌زمینه ای که تلاش به خواندن ورودی از صفحه کلید می‌کند فرستاده می‌شود. چون پردازه‌های پس‌زمینه نمی‌توانند ورودی از صفحه کلید را بخوانند ،به صورت پیش فرض این سیگنال برنامه را موقتا متوقف می‌کند. وقتی شما پردازه پس‌زمینه را با SIGCONT به حالت ادامه اجرا در می‌آورید و آن‌را به حالت پیش زمینه می‌برید، می‌تواند مجددا ورودی را از صفحه کلید بخواند.
- SIGTTOU : این سیگنال به پردازه‌ی پس زمینه‌ای که تلاش به نوشتن خروجی در ترمینال می‌کند، فرستاده می‌شود درحالیکه پردازه پیش‌زمینه دیگری وجود دارد که درحال استفاده از ترمینال است. همچنین این سیگنال به صورت پیش فرض مشابه SIGTTIN عمل می‌کند.

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

```
kill -XXX PID
```

برای مثال دستور زیر سیگنال SIGTERM را به پردازه با شناسه PID ارسال می‌کند:

```
kill -TERM PID
```

در C می‌توانید از تابع signal استفاده کنید که نحوه برخورد با پردازه کنونی را تغییر دهید. shell می‌بایست بیشتر این سیگنالها را نادیده بگیرد درحالی‌که زیرپردازه‌های آن براساس یک عملکرد پیش فرض نسبت به آن‌ها پاسخ دهند. مثلا shell می‌بایست سیگنال SIGTTOU را نادیده بگیرد امّا زیرپردازه‌ها می‌بایست پاسخ دهند. توجه: پردازه‌های fork شده از signal handler های پردازه اصلی ارث بری می‌کنند. برای کسب اطلاعات بیشتر می‌توانید از دستورهای زیر استفاده کنید:

```
man 2 signal

man 7 signal
```

همچنین اطمینان حاصل کنید از اینکه ثوابت [۱۳]SIG_IGN و SIG_DFL را بررسی کرده باشید. برای مطالعه کامل توصیه می‌کنیم به این لینک مراجعه نمایید.

و اما وظیفه اصلی شما این است که اطمینان حاصل کنید از اینکه هر برنامه در گروه پردازه خودش start می‌شود. وقتی شما یک پردازه را start می کنید، گروه پردازه‌اش می‌بایست به حالت پیش زمینه برود. همچنین سیگنال توقف می‌بایست تنها برروی برنامه پیش زمینه اثر بگذارد نه shell پس زمینه.

۷ پردازش پس‌زمنیه

تابه حال shell به گونه‌ای بوده است که قبل از شروع برنامه بعدی منتظر اتمام برنامه‌های قبلی می‌ماند. بسیاری از shell ها امکان اجرای یک دستور در پس زمینه را با قرار دادن علامت "&" در انتهای خط فرمان فراهم می‌سازند. پس از شروع برنامه پس‌زمینه، shell به شما اجازه می دهد که پردازه‌های بیشتری را بدون انتظار جهت اتمام پردازه پس زمینه، شروع کنید.

[۱۲] override
[۱۳] constant

shell را به گونه‌ای تغییر دهید که فرمان‌هایی که با قالب مذکور وارد می‌شوند را در پس زمینه اجرا کند. توجه کنید که تنها می‌بایست پشتیبانی از پردازه‌های پس زمینه را فراهم کنید نیازی به پیاده سازی فرمان‌های داخلی نیست.
همچنین می‌بایست، دستورجدید داخلی wait را اضافه کنید. این دستور این گونه است که صبر می‌کند تا تمام کارهای پس‌زمینه تمام شوند و سپس به حالت عادی باز می‌گردد.
می‌توانید فرض کنید که همواره پیرامون کاراکتر & فاصله وجود دارد. همچنین فرض کنید که این کاراکتر آخرین token در آن خط فرمان است.

۸ Judge

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 **push** جدید برروی مخزن شما، کار داوری را شروع می‌کند.
- اگر که تمرین خود را با تاخیر می‌فرستید، سامانه داوری نمره‌ی بدون احتساب تاخیر را اعلام می‌کند. برای اطلاع از قوانین ارسال با تاخیر تمرینات به قوانین درس مراجعه کنید.
- نمره‌ای که در نهایت سامانه‌ی داوری به شما می‌دهد، بیشینه نمره‌ی شما در کامیت‌های مختلف نیست بلکه نمره‌ی آخرین کامیت شماست. هرگونه تحویل‌دادنی‌های دیگر مانند مستند گزارش که توسط سامانه داوری نمره‌دهی نمی‌شوند مبتنی بر آخرین کامیت شما نمره‌دهی خواهند شد.
- از **push** کردن کامیت‌های متعدد در زمان کوتاه پرهیز کنید و در استفاده از سامانه داوری دقت کافی را مبذول فرمایید در غیر این‌صورت هرگونه مشکل در سامانه داوری که منجر به کم‌شدن زمان مفید شما تا ضرب‌الاجل ارسال تمرین است بر عهده‌ی خودتان خواهد بود.
- دقت بفرمایید که سامانه ی داوری ابزاری برای آزمودن عملکردها و قابلیت‌های کد شما نیست و صرفا ابزاری برای نمره‌دهی است. بنابراین لطفا از سامانه‌ی داوری به عنوان آزمونگر صحت عملکردهای کد استفاده نکنید و روی سیستم خودتان آزمون‌های لازم را انجام دهید و سپس push کنید. بدین ترتیب، نیازی به push کردن متوالی هم پیدا نخواهید کرد.
- دقت بفرمایید که این سیستم جهت نمره دهی به تمرین است، بنابراین ابتدا از صحت کد خود مطمئن شوید سپس برای تصحیح کد، آن‌را ارسال کنید

۹ تحویل‌دادنی‌ها

به منظور ارسال این تمرین باید کدهای موجود در handout را به پوشه‌ای به نام hw۱ در مسیری که مخزن

git@tarasht.ce.sharif.ir:ce424-982-students/ce424-982-"student-id".git

را در تمرین قبل clone کرده‌اید، انتقال دهید و پس از ایجاد تغییرات مورد نظر تنها فایل‌های h. و c. و Makefile را push کنید. لطفا به هیچ عنوان فایل‌های o. و فایل‌های اجرایی را push نکنید. تنها تحویل‌دادنی این تمرین، کد مربوط به یک shell است که می‌بایست قابلیت‌های مطرح‌شده در قسمتهای قبل را داشته باشد.
توجه: شما موظف هستید که هر مرحله از تغییرات کدتان را commit کنید تا در صورت بروز مشکل، بتوانید به نسخه‌های قبلی بازگردید.
شما می توانید تمرین خود را جهت نمره دهی با دستور زیر ارسال کنید:

```
1    git push origin master
```

در صورت داشتن هرگونه سوال در رابطه با درس و تمرینات، سوال خود را به لیست ایمیلی درس به آدرس **ce424@lists.sharif.ir** ارسال کنید. همچنین اگر در استفاده از سامانه طرشت به مشکلی برخوردید، آن را از طریق ایمیل **hossein.moghaddas26@student.sharif.edu** مطرح کنید.