

به نام خداوند بخشنده مهربان



دانشکده‌ی مهندسی کامپیوتر

پاییز ۱۳۹۷

تمرین یکم\*  
سیستم‌های عامل

دانشگاه صنعتی شریف

مدرس: مهدی خرازی

## اهداف تمرین

- ساخت یک shell
- فراهم نمودن قابلیت مدیریت و اجرای برنامه‌ها

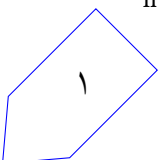
### ۱. مقدمه

این تمرین شامل شش بخش اصلی است. در این تمرین یک shell خواهید ساخت و هدف از این کار آن است که کاربر بتواند برنامه‌های خود را مدیریت و اجرا کند. هسته سیستم‌عامل<sup>۱</sup> مستندات بسیار مفیدی جهت ساخت shell فراهم نموده است.

با ساخت یک shell برای خودتان با این واسطه<sup>۲</sup>ها بیشتر آشنا خواهید شد و احتمالاً اطلاعاتی را پیرامون سایر shellها نیز کسب خواهید نمود.

\* این تمرین برگرفته از تمرین ارائه شده در دانشگاه برکلی، مربوط به درس CS۱۶۲ می‌باشد.  
با سپاس از زحمات تیم دستیاران تمرین

<sup>۱</sup>kernel  
<sup>۲</sup>interface



## ۲. راه‌اندازی مقدمات

به ماشین مجازی خود در Vagrant وارد شده و همان‌طور که در تمرین صفر گفته شد، یک پوشه متناسب با این تمرین ایجاد کنید و در آن پوشه موارد مرتبط با این تمرین را قرار دهید.

```
cd ~/code/personal/hw1
```

تیم دستیاران تمرین کد شروع کار بر روی shell و یک Makefile ساده را برای شما در قسمت skeleton قرار داده‌اند. در این قسمت، قطعه برنامه‌ای را مشاهده خواهید کرد که یک‌رشته<sup>۳</sup> را دریافت می‌کند و آن را به کلمات می‌شکند<sup>۴</sup>. (شما می‌توانید skeleton را از اینجا دریافت کنید).  
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 exit را تایپ کرده و یا CTRL-D را فشار دهید.

---

<sup>۳</sup>string  
<sup>۴</sup>split

## ۳. پشتیبانی از دستور cd و pwd

ساختار کد shell شما یک dispatcher برای دستورهای<sup>۵</sup> از پیش تعبیه شده<sup>۶</sup> دارد. در واقع هر shell یک سری دستورات از پیش تعبیه شده دارد که عملگرهای داخلی و مربوط به خود shell هستند نه برنامه‌های خارجی. برای مثال دستور exit می‌بایست به عنوان یک دستور از پیش تعبیه شده، پیاده‌سازی شده باشد زیرا این دستور خود shell را exit می‌کند.

این shell که هم اکنون در اختیار شماست تنها دو دستور از پیش تعبیه شده دارد. دستور ؟ که منوی راهنما را نشان می‌دهد و دستور exit که shell را exit می‌کند. اولین تمرین شما آن است که دستور جدید pwd را اضافه کنید که مسیر کنونی را با قالب خروجی استاندارد چاپ کند. سپس دستور جدید cd را اضافه کنید که یک آرگومان دریافت می‌کند. این دستور یک مسیر را دریافت کرده و مسیر کنونی را به آن مسیر تغییر می‌دهد. پس از افزودن این دو دستور 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: در تمامی مراحل موظف هستید کد خود را به صورت مرتب commit کنید، زیرا با این کار می‌توانید در صورت نیاز به نسخه‌های قبلی کد خود بازگردید.

---

<sup>۵</sup>command

<sup>۶</sup>built-in

## ۴. اجرای برنامه

اگر تلاش کنید که چیزی در shell تایپ کنید که جزو دستورهای از پیش تعبیه شده نباشد، یک پیام را مشاهده خواهید نمود که shell چگونگی اجرای برنامه را نمی‌داند. طوری shell خود را تغییر دهید که وقتی برنامه‌ای را وارد می‌کنید آن را اجرا کند. اولین کلمه لزهر فرمان، نام برنامه و مابقی کلمات آرگومان‌های برنامه خواهند بود. فعلا می‌توانید این‌گونه در نظر بگیرید که کلمه‌ی اول همان آدرس کامل<sup>۷</sup> برنامه است. بنابراین به جای اجرای `wc` بایستی `/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 پیاده سازی کنید.

می‌بایست تنها از توابع تعریف شده در پوشه مربوط به تمرین برای جداسازی و شکستن متن ورودی به کلمات بهره ببرید. پس از پیاده سازی این گام قادر خواهید بود که برنامه‌هایی مشابه زیر را اجرا کنید:

```
./shell
```

```
0: /usr/bin/wc shell.c
```

```
77 262 1843 shell.c
```

```
1: exit
```

وقتی shell نیاز دارد که یک برنامه را اجرا کند، می‌بایست یک پردازهی فرزند را `fork` کند، که در واقع یکی از توابع `exec` را برای اجرای برنامه جدید فراخوانی می‌کند. پردازهی والد می‌بایست صبر کند تا پردازهی فرزند به اتمام برسد و سپس به فرمان‌های جدید گوش دهد.

---

<sup>۷</sup>full path

## ۵. نام‌گذاری با استفاده از Path

احتمالا تاکنون متوجه شده‌اید که تست کردن shell در قسمت قبل بسیار سخت بود، زیرا می‌بایست مسیر کامل 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 hash table 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 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، در هر مسیر از متغیر محلی PATH به دنبال برنامه‌ای با نام «wc» می‌گردد و اولین برنامه‌ای که پیدا می‌کند را اجرا می‌کند. هر مسیر در PATH با استفاده از علامت «:» از سایرین جدا می‌شود.

حال می‌بایست shell خود را چنان تغییر دهید که از متغیر محلی PATH استفاده کند و برنامه را با نام ساده‌ی آن نیز اجرا کند.

نکته: می‌بایست کماکان از تایپ مسیر کامل برنامه نیز پشتیبانی شود.

نکته: به هیچ وجه از «execvp» استفاده نکنید، زیرا در این صورت نمره‌ای به شما تعلق نخواهد گرفت. به جای آن می‌توانید از «execv» استفاده کنید و PATH resolution خودتان را پیاده‌سازی کنید.

## ۶. خواندن ورودی و نوشتن خروجی با استفاده از فایل

هنگام اجرای برنامه‌ها گاهی اوقات مفید است که ورودی از فایل خوانده شود یا خروجی در یک فایل نوشته شود. دستور «[process] > [file]» به shell می‌گوید که خروجی استاندارد پردازش می‌بایست در یک فایل نوشته شود. به طور مشابه دستور «[process] < [file]» به shell می‌گوید که محتوای فایل را به عنوان ورودی استاندارد پردازش 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 دستورهای از پیش تعبیه شده shell ندارید. همچنین نیازی به پشتیبانی از redirect کردن از stderr یا append کردن به فایل‌ها ( [process] >> [file] ) نیست. فرض کنید که همواره پیرامون دو کاراکتر خاص > و < کاراکتر space وجود دارد. توجه کنید که «[file] <» یا «[file] >» به عنوان آرگومان به برنامه پاس داده نمی‌شوند.

## ۷. کار با سیگنال

بیشتر shellها این امکان را فراهم می‌کنند که با فشردن یک کلید خاص مانند Ctrl-C یا Ctrl-Z، پردازنده‌ها را pause و یا stop کنید. این keystrokeها با فرستادن سیگنال به زیرپردازنده<sup>۸</sup>های shell کار می‌کنند. برای مثال با فشردن CTRL-C یک سیگنال SIGINIT ارسال می‌شود که معمولاً برنامه‌ی در حال اجرا را stop می‌کند و با فشردن CTRL-Z یک سیگنال SIGTSTP ارسال می‌شود که معمولاً برنامه‌ی در حال اجرا را به حالت پس‌زمینه<sup>۹</sup> می‌برد. اگر در حال حاضر این keystrokeها را بر روی shell خود به کار ببرید، سیگنال‌ها به پردازنده خود shell ارسال می‌شوند و این آن چیزی نیست که ما به دنبالش هستیم. مثلاً وقتی شما با فشردن CTRL-Z می‌خواهید یک زیرپردازنده از shell خود را متوقف کنید، خود shell نیز متوقف می‌شود. ما می‌خواهیم سیگنال‌هایی داشته باشیم که تنها بر روی زیرپردازنده‌هایی که shell ایجاد کرده است اثر بگذارند. قبل از توضیح نحوه انجام این کار قصد داریم پیرامون چند مفهوم در سیستم‌های عامل بیشتر صحبت کنیم.

### ۱.۷. گروه‌های پردازنده

می‌دانید که هر پردازنده ای یک pid یکتا دارد. اما هر پردازنده یک pgid مخصوص گروه خود را داراست که یکتا نیست و به صورت پیش‌فرض همان pgid پردازنده والد است. پردازنده‌ها می‌توانند شناسه گروه خود را دریافت و مقداردهی کنند و این کار به کمک دستورهای getpgid() و setpgid() یا getpgrp() و setpgrp() انجام می‌شود. در نظر داشته باشید که وقتی شما یک برنامه جدید را شروع می‌کنید، می‌بایست سایر پردازنده‌های در حال اجرا به صورت درست و بدون آنکه در عملکردشان تداخلی ایجاد شود به کار خود ادامه دهند. تمام این پردازنده‌ها pgid مشابه پردازنده اصلی را ارث‌بری می‌کنند. بنابراین ایده‌ی خوبی به نظر می‌رسد که هر زیرپردازنده shell را در گروه مربوط به خودش قرار دهید و با این کار سازماندهی آن‌ها را آسان‌تر کنید.

نکته: وقتی هر زیرپردازنده‌ای را در گروه پردازنده خودش قرار دهید، pgid می‌بایست با pid برابر باشد.

### ۲.۷. Foreground terminal

هر ترمینال یک pgid مربوط به گروه پردازنده‌های پیش‌زمینه<sup>۱۰</sup> دارد. وقتی CTRL-C را تایپ می‌کنید، ترمینال یک سیگنال به هر پردازنده‌ای که در گروه پردازنده‌های پیش‌زمینه باشد ارسال می‌کند. می‌توانید گروه پردازنده‌هایی که در پیش‌زمینه ترمینال قرار دارند را به کمک تابع زیر تغییر دهید:

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

در حالت ورودی استاندارد، fd می‌بایست صفر باشد.

<sup>۸</sup>subprocess

<sup>۹</sup>background

<sup>۱۰</sup>foreground

## ۳.۷. آشنایی با سیگنال‌ها

سیگنال‌ها پیام‌های آسنکرونی هستند که به پردازنده‌ها فرستاده می‌شوند و با شماره سیگنال شناسایی می‌شوند. گاهی اوقات نیز اسامی سیگنال‌ها کاملاً با عملکردشان متناسب است و با SIG آغاز می‌شوند. برخی از سیگنال‌ها عبارتند از:

- SIGINT : با تایپ CTRL-C فرستاده می‌شود و به صورت پیش فرض برنامه را متوقف می‌کند.
- SIGQUIT : با تایپ CTRL-\ فرستاده می‌شود و به صورت پیش فرض برنامه را متوقف می‌کند، اما برنامه‌ها به صورت جدی‌تری نسبت به این سیگنال واکنش نشان می‌دهند. همچنین این سیگنال تلاش می‌کند که یک core dump از برنامه، قبل از exit کردن آن تولید کند.
- SIGKILL : کلید میانبری برای این سیگنال وجود ندارد. همچنین این سیگنال به اجبار برنامه را متوقف می‌کند و نمی‌تواند توسط برنامه لغو شود، در حالی که 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 به صورت پیش فرض برنامه را pause می‌کند. در bash اگر این کار را انجام دهید، برنامه کنونی pause شده و bash شروع به دریافت دستورهای بیشتر می‌کند.
- SIGCONT : اگر دستور fg یا fg %NUMBER را در bash وارد کنید، سیگنال فرستاده می‌شود. این سیگنال اجرای برنامه pause شده را ادامه می‌دهد.
- SIGTTIN : این سیگنال به پردازنده پس‌زمینه‌ای که تلاش به خواندن ورودی از صفحه‌کلید می‌کند فرستاده می‌شود. چون پردازنده‌های پس‌زمینه نمی‌توانند ورودی از صفحه‌کلید را بخوانند به صورت پیش فرض این سیگنال برنامه را pause می‌کند. وقتی شما پردازنده پس‌زمینه را با SIGCONT به حالت ادامه اجرا در می‌آورید و آن را به حالت پیش‌زمینه می‌برید، می‌تواند مجدداً ورودی را از صفحه‌کلید بخواند.
- SIGTTOU : این سیگنال به پردازنده پس‌زمینه‌ای که تلاش به نوشتن خروجی در ترمینال می‌کند، فرستاده می‌شود در حالی که پردازنده پیش‌زمینه 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 shell خود می‌توانید از دستور زیر استفاده کنید:

```
kill -XXX PID
```

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



```
kill -TERM PID
```

در زبان C می‌توانید از تابع `signal` استفاده کنید که نحوه برخورد با پردازش کنونی را تغییر دهید. `shell` می‌بایست بیشتر این سیگنال‌ها را نادیده بگیرد، در حالی که زیرپردازهای آن براساس یک عملکرد پیش فرض نسبت به آن‌ها پاسخ دهند. مثلاً `shell` می‌بایست سیگنال `SIGTTOU` را نادیده بگیرد، اما زیرپردازها می‌بایست پاسخ دهند.

نکته: پردازش‌های `fork` شده از `signal handler` های پردازش اصلی ارث‌بری می‌کنند. برای کسب اطلاعات بیشتر می‌توانید موارد زیر را مطالعه کنید:

```
man 2 signal
```

```
man 7 signal
```

همچنین اطمینان حاصل کنید از اینکه ثوابت `SIG_IGN` و `SIG_DFL` را بررسی کرده باشید. برای مطالعه کامل توصیه می‌کنیم به [این پیوند](#) مراجعه نمایید.

و اما وظیفه اصلی شما این است که اطمینان حاصل کنید از اینکه هر برنامه در گروه پردازش خودش شروع می‌شود. وقتی شما یک پردازش را شروع می‌کنید، گروه پردازش می‌بایست به حالت پیش‌زمینه برود. همچنین سیگنال‌های متوقف‌کننده می‌بایست تنها بر روی برنامه پیش‌زمینه اثر بگذارد نه `shell` پس‌زمینه.

## ۸. پردازش پس‌زمینه

تا به حال shell به گونه‌ای بوده است که قبل از شروع برنامه بعدی منتظر اتمام برنامه‌های قبلی می‌ماند. بسیاری از shellها امکان اجرای یک دستور در پس‌زمینه را با قراردادن علامت «&» در انتهای خط فرمان فراهم می‌سازند. پس از شروع برنامه پس‌زمینه، shell به شما اجازه می‌دهد که پردازش‌های بیشتری را بدون انتظار جهت اتمام پردازش پس‌زمینه، شروع کنید.

shell را به گونه‌ای تغییر دهید که دستورهایی را که با فرمت مذکور وارد می‌شوند در پس‌زمینه اجرا کند. توجه کنید که تنها می‌بایست پشتیبانی از پردازش پس‌زمینه را فراهم کنید و نیازی به پیاده‌سازی دستور از پیش تعبیه شده نیست.

همچنین می‌بایست، دستور جدید از پیش تعبیه شده‌ی wait را اضافه کنید. این دستور این‌گونه است که صبر می‌کند تا تمام کارهای پس‌زمینه terminate شوند و سپس به حالت عادی بازمی‌گردد. می‌توانید فرض کنید که همواره پیرامون کاراکتر & فاصله وجود دارد. همچنین فرض کنید که این کاراکتر آخرین کاراکتر در آن خط فرمان است.

## ۹. تحویل دادنی‌ها

تنها تحویل دادنی این تمرین یک shell است که می‌بایست قابلیت‌های مطرح شده در قسمت‌های قبل را فراهم کرده باشد.

نکته: پس از انجام هر یک از قسمت‌ها و فراهم نمودن یک قابلیت جدید برای shell خود، شما باید تغییرات را commit کنید، در غیر این صورت از شما نمره کسر خواهد شد.  
شما می‌توانید تمرین خود را جهت نمره‌دهی با دستور زیر ارسال کنید:

```
git commit -m 'Finished HW1'
```

```
git push origin master
```