به نام خدا

# درس سیستم‌های عامل

نیم‌سال دوم ۰۱-۰۰

دانشکدهٔ مهندسی کامپیوتر

دانشگاه صنعتی شریف

---

مدرس **مهدی خرازی**

تمرین **صفر**

موضوع **آشنایی و راه‌اندازی پیش‌نیازهای تمرین‌ها**

موعد تحویل **ساعت ۲۳:۵۹ سه‌شنبه ۳ اسفند ۱۴۰۰**

با سپاس از دستیاران آموزشی **محمد حدادیان، مجید گروسی و امیرمهدی نامجو**

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

در طول این ترم، شما از ابزارهای مختلفی برای توسعه، رفع‌خطا [1] و تحویل کدهای خود استفاده خواهید کرد. در این تمرین که از سه بخش راه‌اندازی پیش‌نیازهای تمرین، آشنایی با چند ابزار و انجام یک تمرین ساده تشکیل شده است خود را برای ادامه‌ی تمرین‌ها آماده می‌کنید.

# ۱ راه‌اندازی مقدمات

## ۱.۱ Gogs

در تمام تمرین‌های این درس شما باید کدها و مستندات خود را در سامانه‌ی طرشت ارسال کنید. به این منظور مخزن‌هایی خصوصی توسط تیم دستیاران آموزشی برای شما ساخته و در اختیارتان قرار خواهد گرفت. **از استفاده از مخازن عمومی برای کدهای خود اکیداً خودداری کنید. مطالعه‌ی کدهای موجود در بستر اینترنت برای فهم بهتر مسائل توصیه می‌شود اما استفاده از آنها اولاً باید همراه با ذکر منبع بوده و دوماً دانشجو خود باید کدها را نوشته و از این منابع فقط برای درک مطالب استفاده کند.**

## ۲.۱ Vagrant

تصویری [2] از یک ماشین مجازی Vagrant در اختیار شما قرار خواهد گرفت که تمام ابزارها و تنظیمات موردنیاز در طول این درس برای توسعه و اجرای کدهایتان در آن تعبیه شده است. Vagrant ابزاری برای مدیریت ماشین‌های مجازی است. شما می‌توانید از Vagrant برای بارگیری و اجرای ماشین‌های مجازی آماده استفاده کنید (در ترم جاری تغییراتی در ماشین مجازی اعمال کرده‌ایم، از نسخه‌های مربوط به ترم‌های گذشته استفاده نکنید).

**اگر از سیستم‌عامل Windows استفاده می‌کنید، اقدامات زیر ممکن است که برای شما کار نکند. اگر این اقدامات انجام شدند یعنی این بخش به‌درستی انجام شده است اما اگر انجام نشدند، به بخش پایین‌تر از این مستند و زیر بخش Windows مراجعه کنید.**

۱. Vagrant مبتنی بر مجازی‌ساز VirtualBox است بنابراین نیاز است تا آخرین نسخه‌ی این مجازی‌ساز را از وبسایت آن بارگیری و نصب کنید. توجه کنید که در فرآیند بارگیری VirtualBox ممکن است به استفاده از ابزارهای رفع‌تحریم نیاز پیدا کنید. دیده شده که برروی تعدادی از نسخه‌های VirtualBox مشکلاتی برای اجرای تصویر به‌وجود می‌آید، از این رو حتماً آخرین نسخه‌ی آن را بارگیری و نصب کنید. همچنین توجه کنید که هرچند از لحاظ تئوری امکان کار کردن با VMWare هم وجود دارد، ولی احتمالاً برای راه‌اندازی آن نیاز به عوض کردن بسیاری از تنظیمات خواهید داشت. در نتیجه توصیه می‌شود تنها از VirtualBox استفاده کنید.

۲. آخرین نسخه‌ی Vagrant را از وبسایت آن بارگیری و نصب کنید. (برای بارگیری این ابزار هم ممکن است به استفاده از ابزارهای رفع‌تحریم نیاز پیدا کنید)

۳. بعد از آن که Vagrant نصب شد چنان چه از سیستم‌عامل لینوکس استفاده می‌کنید ابتدا بسته‌ی libarchive-tools را با دستور زیر نصب کنید. توجه داشته باشید که بسته به توزیع لینوکسی که استفاده می‌کنید، ممکن است برای نصب بسته‌ها نیاز به دستور دیگری به جز apt داشته باشید.

```
$ sudo apt install libarchive-tools
```

سپس دستورات زیر را در ترمینال خود اجرا کنید:

[1] Debug
[2] Image

```
$ mkdir ce424-vm
$ cd ce424-vm
$ vagrant init ce424/spring2020
$ vagrant up
$ vagrant ssh
```

این دستورات تصویر ماشین مجازی ما برای این درس را بارگیری کرده و یک نشست ssh برقرار می‌کند. دستور **up** احتمالا مدتی زمان خواهد برد و به اتصال اینترنت هم نیاز دارد.

۴. پس از اتصال به ماشین مجازی، فایل vm_patch.sh را از آدرسی که در ادامه آمده بارگیری نموده و سپس آن را اجرا کنید. برای این کار می‌توانید از دستورات زیر استفاده کنید. در صورتی که اجرای آن موفقیت‌آمیز بود می‌توانید این فایل را حذف نمایید. از آن‌جایی که این اسکریپت مخازنی را از سامانه طرشت دریافت می‌کند، توصیه می‌شود ابتدا قسمت مربوط به ssh-keys را انجام دهید تا در اتصال به گیت و سامانه طرشت مشکل نداشته باشید.

```
$ wget 'http://sharif.edu/~kharrazi/courses/40424-002/vm_patch.sh'
$ bash vm_patch.sh
$ rm vm_patch.sh
```

۵. شما از این پس باید تمام دستورات مربوط به Vagrant را از پوشه‌ی **ce424-vm** اجرا کنید. این پوشه را پاک نکنید در غیر این‌صورت Vagrant نمی‌تواند ماشین‌های مجازی شما را مدیریت کند.

۶. برای توقف اجرای Vagrant از دستور **vagrant halt** استفاده کنید. اگر این دستور کار نکرد، مطمئن شوید که آن را در ماشین میزبان اجرا می‌کنید و نه در ssh. برای دفعات بعدی شما فقط باید دستورات **vagrant up** و **vagrant ssh** را اجرا کنید و نیازی به اجرای دستورات قبل از آن نیست.

### Windows ۱.۲.۱

نسخه‌ی ویندوز شما ممکن است از SSH در سطح خط فرمان پشتیبانی نکند خصوصاً اگر آخرین نسخه‌ی ویندوز را نداشته باشید. در این حالت دستور **vagrant ssh** خطایی مبنی بر دانلود Cygwin یا موارد مشابهی که از ssh پیشتیبانی می‌کنند نمایش خواهد داد. اینجا توضیحات خوبی برای چگونگی نصب Cygwin آمده است. پس از اتمام نصب، دستورالعمل‌های بخش قبل را از طریق ترمینال Cygwin اجرا کنید (قبل از اجرای دستورات، سیستم‌عامل خود را مجدداً راه‌اندازی کنید).
امکان استفاده از PuTTY به‌جای Cygwin هم وجود دارد اما این‌کار به اقدامات بیشتری برای راه‌اندازی نیاز دارد.
اگر شما خطایی مبنی بر رسیدن به محدودیت زمانی هنگام بالا آمدن ماشین مجازی خود برخوردید، احتمالاً باید ویژگی VT-x مربوط به CPU خود را از طریق BIOS فعال کنید.

### Troubleshooting Vagrant ۲.۲.۱

اگر دستور **vagrant up** با خطا مواجه شد، دستور **vagrant provision** را اجرا کنید. این دستور احتمالاً مشکل را برطرف خواهد کرد. اگر این دستور هم مشکل را برطرف نکرد، با اجرای دستور **vagrant destroy** می‌توانید ماشین مجازی خود را از بین برده و مجدداً با دستور **vagrant up** مراحل را تکرار کنید.

### Git Config ۳.۲.۱

دستورات زیر را با ایمیل ثبت‌نامی خود اجرا کنید تا تنظیمات مربوط به نام و ایمیلی که برای کامیت‌های خود استفاده می‌کنید اعمال شوند.

```
$ git config --global user.name  "Your Name"
$ git config --global user.email "Your Registered Email"
```

### ssh-keys ۴.۲.۱

برای اتصال به سامانه‌ی طرشت، نیاز دارید تا کلیدهای عمومی و خصوصی خود را برای برقراری ارتباط ssh با سرور تنظیم کنید. برای این کار، از درون VM خود دستورات زیر را به‌ترتیب اجرا کنید:

```
$ ssh-keygen -N "" -f ~/.ssh/id_rsa
$ cat ~/.ssh/id_rsa.pub
```

دستور اول یک جفت کلید خصوصی و عمومی برای شما ایجاد و آن را در آدرس **~/.ssh/** ذخیره خواهد کرد. دستور دوم هم مقدار کلید عمومی ساخته شده را نمایش خواهد داد. حال شما باید در سامانه‌ی طرشت به این تنظیمات از حساب خود رفته و کلید نمایش داده‌شده را اضافه کنید. کلید شما باید با عبارت **ssh-rsa** آغاز و با عبارت **vagrant@development** پایان یافته باشد.

### Repos ۵.۲.۱

تمام پرونده‌های موردنیاز برای تمرین‌های فردی درس، در یک مخزن عمومی از سامانه طرشت قرار دارند. این مخزن در نشانی زیر در دسترس است:

**https://tarasht.ce.sharif.edu/ce424-002-students/ce424-002-handouts**

چنین مخزنی برای تمرین‌های گروهی هم وجود خواهد داشت که در مستندات مربوط به تمرین‌های گروهی در مورد آن توضیحات لازم ذکر خواهد شد. محتویات این مخزن در آدرس **~/code/handouts** از ماشین مجازی شما قرار دارد. برای به‌روزرسانی این مخزن می‌توانید از دستور زیر استفاده کنید. دقت کنید که برای بارگیری پرونده‌های این مخزن، از عبارت handouts در دستور pull استفاده شده است.

```
$ cd ~/code/handouts
$ git pull origin master
```

همچنین هر دانشجو به دو مخزن خصوصی دسترسی دارد که شامل یک مخزن خصوصی برای تمرین‌های فردی و یک مخزن گروهی برای پروژه‌های گروهی است. این مخزن‌ها به ترتیب در آدرس‌های **~/code/personal** و **~/code/group** از ماشین مجازی شما قرار دارند. پس از آن که تمرین‌های این درس را در پرونده‌های مربوط به تمرین‌های فردی انجام دادید، با استفاده از دستور زیر می‌توانید تغییراتتان را به مخزن مربوط به خودتان در سامانه‌ی طرشت منتقل کنید.

```
$ git push origin master
```

همچنین می توانید در پوشه تمرین‌های فردی خود از دستور

```
$ git pull handouts master
```

هم برای بارگیری پرونده‌های مخزن **handouts** استفاده کنید.
مخازن مربوط به تمرین‌های فردی ساختاری شبیه به

**git@tarasht.ce.sharif.ir:ce424-002-students/ce424-002-"student-id".git**

و مخازن تمرین‌های گروهی ساختاری شبیه به

**git@tarasht.ce.sharif.ir:ce424-002-students/ce424-002-"group-id".git**

خواهند داشت(در زمان انتشار این مستند مخزن‌های گروهی هنوز ساخته نشده‌اند).
در این درس انشعاب [۳] **master** به عنوان انشعاب پیش‌فرض برای ارسال کدهای شما درنظر گرفته شده است. شما می‌توانید انشعاب‌های دیگری برای کدهای خود بسازید اما درنهایت این انشعاب است که توسط سیستم داوری بررسی می‌شود.

[3] Branch

### Issues ۶.۲.۱

بخش Issues در سامانه‌ی طرشت این امکان را به گروه‌ها می‌دهد که برای مشکلات مختلف موجود در کدهای خود، Issue هایی نوشته و هرکدام را به یکی از اعضای تیم اختصاص دهند.

## Judge ۳.۱

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 تمرین‌ها به قوانین درس مراجعه کنید.
- نمره‌ی نهایی‌ای که 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 ضرب‌العجل ارسال تمرین است بر عهده‌ی خودتان خواهد بود.
- تمرین شماره صفر داوری خودکار نداشته و بعد از اتمام تحویل‌ها نمره‌دهی خواهد شد.

## ویرایش کد در ماشین مجازی ۴.۱

ماشین مجازی شما دارای یک سرور SMB [۴] است که اجازه‌ی ویرایش پرونده‌های موجود در پوشه‌ی **home** از کاربر Vagrant را می‌دهد. با این سرور SMB شما می‌توانید با استفاده از هر ویرایش‌گر دلخواه در ماشین میزبان خود پرونده‌ها را ویرایش کنید. **ما در طول این درس این روش را توصیه می‌کنیم**، اما شما می‌توانید از هر روشی که ترجیح می‌دهید استفاده کنید. یک روش استفاده از رابط‌کاربری غیرگرافیکی ویرایش‌گر متن در نشست ssh است. به عنوان مثال VSCode امکاناتی را برای اتصال از طریق ssh به ماشین‌مجازی را در اختیار کاربران قرار می‌دهد.

### Windows ۱.۴.۱

۱. **File Browser** را باز کرده و با زدن کلیدهای **Ctrl+L** به محل وارد کردن آدرس بروید.

۲. آدرس **\\192.168.162.162\vagrant** را وارد کرده و **Enter** بزنید.

۳. نام‌کاربری و رمزعبور در این قسمت هردو **vagrant** است.

حال شما می‌توانید محتویات پوشه‌ی **home** کاربر خود در Vagrant را ببینید.

### Mac OS X ۲.۴.۱

برای استفاده از پوشه‌های اشتراکی در این سیستم‌عامل، ویژگی **File Sharing** باید فعال باشد. اگر این ویژگی فعال نیست، با مراحل زیر آن را فعال کنید:

۱. **System Preferences** را باز کنید.

۲. بر روی گزینه **Sharing** کلیک کنید.

۳. تیک مربوط به گزینه **On** را در کنار **File Sharing** بزنید.

۴. بر روی گزینه **Options** کلید کنید.

[4] Server Message Block

۵. تیک **Share files and folders using SMB** را بزنید.

۶. تیک گزینه **On** در کنار نام کاربری خود و عبارت **Windows File Sharing** را بزنید.

۷. **Done** را بزنید.

پس از آن مراحل زیر را انجام دهید:

۱. **Finder** را باز کنید.

۲. در نوار منو، گزینه **Go** → **Connect to Server...** را انتخاب کنید.

۳. آدرس سرور **smb://192.168.162.162/vagrant** است.

۴. نام‌کاربری و رمزعبور در این قسمت هردو **vagrant** است.

### Linux ۳.۴.۱

از هر کارخواه [۵] SMB برای اتصال به **/vagrant** با نام کاربری و رمزعبور **vagrant** استفاده کنید. توزیع مرورگر پرونده شما ممکن است که از SMB پشتیبانی کند پس دستورات مربوط به استفاده از آن را جست‌وجو کنید.

### Shared Folders ۴.۴.۱

پوشه‌ی **/vagrant** از ماشین مجازی شما به پوشه‌ی **home** از ماشین میزبان شما متصل است. شما می‌توانید از این اتصال استفاده کنید اما روش پیشنهادی ما روش SMB است که در بخش قبل توضیح داده شد.

### برخی مشکلات رایج در هنگام راه‌اندازی ۵.۴.۱

- در صورتی که در ویندوز بعد از اجرای دستور **vagrant up** در ویندوز با خطایی از نوع **E_FAIL** یا **E_INVALIDARG** **(0x80004005)** مواجه شدید که بیان می‌کرد امکان ساخت **Host-only Adapter** در VirtualBox وجود ندارد، به **Device Manager** ویندوز رفته و تمامی دیوایس هایی که نامشان به صورت **Virtual Box Host-only Ethernet Adapter** است را Uninstall کنید و دوباره دستور **vagrant up** را اجرا کنید. بعد از یک یا دو بار انجام این کار احتمالا مشکل شما برطرف خواهد شد. توجه کنید که با این کار ممکن است اتصال به اینترنت سایر ماشین‌مجازی‌هایی که داشتید دچار مشکلاتی بشود و نیاز به تنظیم مجدد آن‌ها داشته باشید.
- در صورتی که در حین فرآیند **vagrant up** برای راه‌اندازی SSH دچار vagrant دچار timeout شد، بعد از اجرای **vagrant halt** و متوقف کردن ماشین مجازی، عبارت

```
config.vm.boot\_timeout = 240
```

را به انتهای **Vagrantfile** اضافه کنید. در صورتی که باز هم مشکل برطرف نشد، با دستور **vagrant destroy** این ماشین‌مجازی را به طور کامل حذف کنید و سپس به انتهای **Vagrantfile** عبارت

```
config.vm.provider "virtualbox" do |vb|
    vb.customize ["modifyvm", :id, "--cableconnected1", "on"]
    vb.customize ["modifyvm", :id, "--uart1", "0x3F8", "4"]
    vb.customize ["modifyvm", :id, "--uartmode1", "file", File::NULL]
end
```

را اضافه کرده و دوباره آن را راه‌اندازی کنید.

[۵] Client

# ۲ معرفی چند ابزار مفید

در این بخش نگاه کوتاهی به تعدادی ابزار خواهیم انداخت که در جعبه‌ابزار هر کاربر پیشرفته‌ای یافت می‌شوند! از میان این ابزارها یادگیری git و make برای ترجمه و ارسال کدهایتان اجباری هستند. سایر ابزارها برای رفع‌خطا یا چندوظیفگی استفاده می‌شوند. همه‌ی این ابزارها برروی ماشین مجازی شما از قبل نصب شده‌اند.
**توجه:** ما در این مستند در هر یک از این ابزارها دقیق نمی‌شویم بلکه لینک‌هایی به منابع مناسب برای یادگیری آن‌ها قرار می‌دهیم که می‌توانید هریک را بیشتر خوانده و درک کنید. ما اکیداً خواندن این نوشته‌ها را توصیه می‌کنیم اگرچه همه‌ی آنها برای این تمرین نیاز نیستند اما تضمین می‌کنیم که در طول ترم به استفاده از آنها احتیاج پیدا خواهید کرد. اگر در استفاده یا درک هریک از این ابزارها به مشکل برخوردید، به دستیاران آموزشی مراجعه کنید.

## ۱.۲ Git

یک برنامه مدیریت نسخه [۶] است که به کمک آن می‌توانید روند توسعه کدها را دنبال کنید. GitHub یکی از سامانه‌های تحت وب است که امکان مدیریت کدهای شما را برایتان فراهم می‌کند. اگرچه شما تاکنون تنظیمات مقدماتی را انجام داده‌اید، اما تسلط شما به قابلیت‌های git می‌تواند در طول این درس خصوصاً در تمرین‌های گروهی به کمک شما بیاید.
اگر تاکنون از git استفاده نکرده‌اید یا دنبال یک شروع خوب هستید، ما پیشنهاد می‌کنیم که از اینجا شروع کنید. اگر درکی از git دارید، این وبسایت برای درک اجزای درونی گیت مفید خواهد بود.

## ۲.۲ make

make ابزاری است که به صورت خودکار، برنامه‌های قابل اجرا و کتابخانه‌ها را از کدمنبع [۷] با خواندن پرونده‌هایی به نام Makefile می‌سازد. پرونده‌های Makefile چگونگی ساخته‌شدن برنامه نهایی را طی دستوراتی و با ساختار خود برای make مشخص می‌کنند. نحوه‌ی کار make خیلی ساده است: شما وابستگی‌های برنامه‌ی نهایی خود را در Makefile مشخص می‌کنید و make یک گراف وابستگی از روی آن ساخته و سپس با پیمایش این گراف، برنامه‌ی نهایی را ایجاد می‌کند. متاسفانه، make ساختار پیچیده‌ای دارد که اگر شما نحوه‌ی کار آن را به‌درستی متوجه نشوید به مشکلاتی در استفاده از آن بر خواهید خورد.
تعدادی آموزش بسیار خوب برای make در اینجا و اینجا وجود دارد. مطالعه‌ی مستندات رسمی GNU در این باره و از اینجا هم در عین فشردگی مفید خواهد بود.
در این بخش ما از ساده‌ترین نوع make استفاده می‌کنیم: بدون Makefile (البته شما به زودی نیاز خواهید داشت تا ساخت Makefile های زیاد را یاد بگیرید)! شما می‌توانید پرونده `limits.c` (که در مخزن عمومی مربوط به این تمرین قرار دارد) را به سادگی اجرای دستور زیر ترجمه و لینک کنید:

```
$ make limits.c
```

این دستور یک پرونده اجرایی تولید می‌کند که می‌توانید با زدن دستور زیر آن را اجرا کنید:

```
$ ./limits
```

(البته کارکرد مربوط به این پرونده هنوز کامل نیست و شما باید در بخش‌های بعد این برنامه را کامل کنید.)

## ۳.۲ man

man یا کتابچه‌ی راهنما ابزار بسیار مهمی است. راهنمایی‌های زیادی در اینترنت وجود دارد اما مستندات موجود در man اصل کار است! صفحه‌ی man از طریق ترمینال سیستم در دسترسی شماست. به عنوان مثال اگر می‌خواهید در رابطه با دستور `ls` بیشتر بدانید، به راحتی دستور `man ls` را در ترمینال خود اجرا کنید. با این کار لیستی از پرچم [۸]های مورد پشتیبانی توسط `ls` و نحوه‌ی کار با آن نمایش داده خواهد شد. شما حتی می‌توانید نحوه‌ی کار با تابعی مانند `fork` را هم با زدن دستور `man fork` مشاهده کنید.

[6] Control Version
[7] Code Source
[8] Flag

## gdb ۴.۲

رفع خطا در برنامه‌های به زبان C سخت است! اگر خطایی در حین اجرای برنامه رخ دهد، پیغام متناسبی با آن یا پیمایش مناسبی از پشته [۹] به‌صورت پیش‌فرض نمایش داده نخواهد شد. خوش‌بختانه، ابزار رفع‌خطا GNU یا همان gdb وجود دارد که به ما امکان رفع‌خطای مناسب از برنامه‌های C را می‌دهد. اگر شما برنامه‌ی خود را با استفاده از پرچم خاص `-g` ترجمه کنید، برنامه‌ی خروجی نمادهای موردنیاز برای رفع‌خطا را خواهد داشت که به gdb اجازه‌ی معجزه را خواهد داد! اگر شما برنامه‌ی خودتان را از طریق gdb اجرا کنید، به شما اجازه‌ی پیمایش پشته، دیدن مقادیر متغیرها، تغییر متغیرها، توقف کد و بسیاری امکانات دیگر را می‌دهد. علاوه بر این، gdb امکان ایجاد پردازه‌های جدید و اتصال آنها به پردازه‌های موجود را هم فراهم می‌کند که این کار برای رفع‌خطا سیستم‌عامل PintOS که در این درس استفاده می‌کنیم مفید خواهد بود.
gdb عادی رابط‌کاربری کاملاً خامی دارد از این رو ما cdgb را برروی ماشین مجازی شما نصب کرده‌ایم که امکان رنگ‌آمیزی دستورات و چند ویژگی خوب دیگر دارد. در cdgb شما می‌توانید با دستورات `i` و `ESC` بین پنجره [۱۰] های بالایی و پایینی جابه‌جا شوید.
این یک مستند بسیار خوب برای درک استفاده از gdb است. مستند رسمی gdb هم مناسب اما اندکی طولانی است.

## tmux ۵.۲

تیماکس یک multiplexer مربوط به ترمینال است که چندین tab مربوط به ترمینال را شبیه‌سازی می‌کند اما آنها را در یک نشست [۱۱] از ترمینال نمایش می‌دهد. البته این چند tab را هنگام ssh به ماشین مجازی حفظ می‌کند.
شما می‌توانید یک نشست جدید با استفاده از دستور

```
$ tmux new -s <session_name>
```

ایجاد کنید.
هنگامی که یک نشست جدید ایجاد کردید شما فقط یک ترمینال عادی مشاهده می‌کنید. با فشار دادن کلیدهای `Ctrl-B + C` یک پنجره‌ی جدید ایجاد می‌کند. `Ctrl-B + n` به پنجره‌ی شماره n می‌رود. توجه کنید که منظور از `Ctrl-B + C` این است که ابتدا کلید `Ctrl` و `B` را با هم فشار بدهید تا tmux وارد حالت گرفتن دستور جدید بشود. سپس کلید `C` را فشار دهید.
`Ctrl-B + D` شما را از نشستی که ایجاد کرده بودید خارج می‌کند. با این کار نشست شما و برنامه‌هایی که در آن نشست در حال اجرا بودند، همچنان در حالت اجرا حفظ می‌شوند و فقط شما از آن نشست خارج می‌شود. شما می‌توانید با زدن دستور زیر مجدداً کار در یک نشست را ادامه دهید:

```
$ tmux attach -t <session_name>
```

بهترین بخش tmux این است که این کار حتی اگر شما نشست ssh خود را ببندید و یک نشست جدید ایجاد کنید هم کار می‌کنند.
یک مستند مناسب برای شروع کار با tmux در اینجا قرار دارد.

## vim ۶.۲

vim یک ویرایش‌گر متنی مناسب برای استفاده از طریق ترمینال است که یادگیری کار کردن با آن ارزش زیادی دارد. در این مستند مجموعه‌ی مناسبی برای بهتر کارکردن با vim وجود دارد. هر ویرایش‌گری که انتخاب می‌کنید، مهم آن است که بتوانید در نوشتن کدها با آن راحت و حرفه‌ای شوید.
برای استفاده از ویرایش‌گرهای دیگری که واسط گرافیکی دارند، به بخش ۴.۱ از این مستند مراجعه کنید.

## ctags ۷.۲

ctags ابزاری است که جابه‌جایی در منابع بزرگ کد را برای شما آسان می‌کند. با توجه به این که در این درس (و در آینده) شما نیاز به بررسی و خواندن کدهای زیادی را دارید، استفاده از این ابزار در صرفه‌جویی زمان تاثیر به‌سزایی خواهد داشت. علاوه بر دیگر ویژگی‌ها، این ابزار به شما اجازه‌ی پرش به محل تعریف یک متغیر را می‌دهد. این ویژگی به همراه ویژگی `go-back-to-last-location` ویرایش‌گر شما بسیار قدرت‌مند خواهند بود.
دستورالعمل‌های مربوط به نصب این ابزار برای vim در اینجا و برای sublime در اینجا قرار دارند. اگر شما از هیچ یک از این دو ویرایش‌گر استفاده نمی‌کنید، ctags احتمالاً از ویرایش‌گر شما هم پشتیبانی می‌کند. در این‌صورت نحوه‌ی نصب آن را جست‌وجو کنید.

[9] Stacktrace
[10] Pane
[11] Session

# ۳ تمرین‌های مقدماتی

## ۱.۳ words

برنامه‌نویسی در زبان C در این درس بسیار مهم است. این تمرین برای اطمینان از آن است که شما با این زبان به اندازه کافی آشنا هستید. به طور خاص، شما باید با **structs** ، داده‌ساختارهای پیوندی [۱۲] مانند **list** ها، اشاره‌گرها، آرایه‌ها، **typedef** و مانند آنها آشنا باشید.
شما باید در این بخش یک برنامه به نام **words** بنویسید که ۱) تعداد تمام کلمات و ۲) تکرار هر کلمه در پرونده(ها) را بشمرد و سپس خروجی را در **stdout** چاپ کنید. مانند بسیاری از دیگر ابزارهای لینوکس در دنیای واقعی، برنامه‌ی شما باید پرونده‌های ورودی خود را از طریق ورودی‌های خط فرمان [۱۳] گرفته و خروجی را به‌صورت تجمعی چاپ کند. اگر هیچ پرونده‌ای به عنوان ورودی داده نشده بود، ورودی را از **stdin** بخواند.

در زبان C پرونده‌های سرآیند [۱۴] (که با پسوند **h.** نمایش داده می‌شوند) چگونگی مدیریت سطوح تجرید[۱۵] را مشخص می‌کنند. سرآیندها اشیا [۱۶]، نوع‌ها [۱۷] و توابع را تعریف کرده و مهم‌تر از همه مستندات کد معمولا در آن‌ها قرار می‌گیرد . پرونده‌هایی که با پسوند **c.** وجود دارند، نحوه‌ی پیاده‌سازی موارد تعریف شده در فایل‌های سرآیند را مشخص می‌کنند. شما می‌توانید کدهایی بنویسید که از پرونده‌های سرآیند استفاده کنند بدون آن که از نحوه‌ی پیاده‌سازی آن با خبر باشید.
در این مورد، پرونده **words/word_conut.h** نحوه‌ی تعریف ساختار **word_count** را مشخص کرده است که از آن به عنوان یک لیست پیوندی [۱۸] برای نگه‌داری مشخصات کلمات و تعداد تکرار آنها استفاده خواهیم کرد. این ساختار به عنوان یک نوع با نام **WordCount** تعریف شده است که به شما اجازه‌ی استفاده از نوع‌داده **WordCount** را می‌دهد. این پرونده سرآیند همچنین لیستی از توابع مورد استفاده در **words/word_count.c** در خود دارد که به عنوان بخشی از این تمرین شما باید کد این توابع را در **words/word_count.c** بنویسید.

ما برای شما یک نسخه‌ی ترجمه‌شده از **sort_words** تهیه کرده‌ایم که در نتیجه شما نیازی ندارید تا تابع **wordcount_sort** را پیاده‌سازی کنید. با این وجود شما نیاز دارید که تابع مقایسه‌کننده خود (مثلاً تابع **wordcount_less** ) را بنویسید. Makefile این پرونده ( **wc_sort.o** ) را با دو آبجکت فایل [۱۹] شما، **words.o** و **word_count.o** لینک می‌کند.
توجه کنید که **words.o** یک پرونده باینری در قالب ELF [۲۰] است بنابرین شما نیاز دارید که از سیستمی که قابلیت اجرای پرونده‌های ELF را دارد (مانند ماشین مجازی درس) استفاده کنید. دقت کنید که سیستم‌عامل‌های Windows و OS X از این قالب پشتیبانی نمی‌کنند و از این رو نمی‌توان از آنها برای این تمرین استفاده کرد.

برای این بخش شما باید تغییرات لازم را در پرونده‌های **words/main.c** و **words/word_count.c** ایجاد کنید. بعد از ایجاد تغییرات لازم در این پرونده‌ها، به محل پوشه‌ی **words** رفته و دستور **make** را در ترمینال خود اجرا کنید. با این کار یک پرونده اجرایی از برنامه‌ی شما ایجاد می‌شود. سپس این پرونده اجرایی را برای خودتان تست کنید. سامانه داوری به طور خودکار برنامه‌ی شما را با تعدادی نمونه آزمایش می‌سنجد.
برای مثال زیر، فرض کنید که ما یک پرونده به نام **words.txt** داریم که شامل محتویات زیر است:

```
abc def AaA
bbb zzz aaa
```

### ۱.۱.۳ Total Word Count

وظیفه اول شما این است که تعداد تمام کلمات را بشمرید. وقتی که این برنامه اجرا شود، باید تعداد تمام کلمات را برروی **stdout** بنویسد. در این بخش نیازی نیست که در **word_count.c** تغییراتی ایجاد کنید. اعمال تغییرات در تابع **num_words()** کافی

[12] Structures Data Linked
[13] Line Command
[14] Header
[15] Abstraction
[16] Objects
[17] Types
[18] List Linked
[19] File Object
[20] Format Linkable and Executable

است.
یک کلمه به عنوان دنباله‌ی پیوسته‌ای از شناسه [۲۱]های الفبایی با طول بیش‌تر از یک تعریف می‌شود. تمام کلمات باید به شکل کوچک [۲۲] تبدیل شده و برنامه‌ی شما نسبت به بزرگی و کوچکی شناسه‌ها حساس نباشد. بیشترین طول یک کلمه در بالای پرونده **main.c** تعریف شده است.

بعد از کامل شدن این بخش، اجرای برنامه‌ی شما برای پرونده نمونه‌ی بالا باید خروجی زیر را دربر داشته باشد:

```
The total number of words is: 6
```

### Word Frequency Count ۲.۱.۳

وظیفه دوم شما این است که تعداد تکرار کلمات را بشمرید. برنامه شما باید هر کلمه را به همراه تعداد تکرار آن، به ترتیب تعداد تکرار (کم‌ترین تکرار اول) و در مواقعی که تعداد تکرار دو کلمه یکسان می‌شود، به ترتیب الفبایی چاپ کند. تابع **wordcount_sort** برای شما در پرونده **wc_sort.o** تعریف شده است. اما شما نیاز دارید تا تابع **wordcount_less** را در پرونده **main.c** پیاده‌سازی کنید.
شما باید توابع موجود در پرونده **WordCount** را به‌گونه‌ای پیاده‌سازی کنید که از لیست‌های پیوندی مانند **word_count.c** پشتیبانی کنند. یک پیاده‌سازی کامل **word_counts.c** برای پیاده‌سازی تابع **count_words()** در پرونده **main.c** بسیار راه‌گشا خواهد بود.

بعد از کامل کردن این بخش، برنامه‌ی شما باید خروجی زیر را برای پرونده نمونه بالا چاپ کند:

```
1 abc
1 bbb
1 def
1 zzz
2 aaa
```

**راهنمایی:** شما می‌توانید دستور زیر را برای اطمینان از درستی برنامه خود اجرا کنید:

```
$ cat <filename> \
    | tr " " "\n" \
    | tr -s "\n" \
    | tr "[:upper:]" "[:lower:]" \
    | tr -d -C "[:lower:]\n" \
    | sort \
    | uniq -c \
    | sort -n
```

### make ۳.۱.۳

همان‌طور که پیش‌تر گفته شد، make ابزار مهمی‌ست که هر کاربر حرفه‌ای باید به آن تسلط داشته باشد. در این تمرین ما برای راحتی کار شما، Makefile مناسب برای ترجمه کردن پرونده‌ها را نوشته‌ایم که در آدرس **words/Makefile** قرار دارد. اما از شما می‌خواهیم به‌صورت خلاصه توضیح دهید که هر خط از این پرونده چه چیزی را مشخص می‌کند. توضیحات خود را در یک پرونده جدید به نام **Makefile.txt** نوشته و در مخزن خصوصی خود قرار دهید.

## user limits ۲.۳

اکنون که با زبان C و ابزارهای طول ترم آشنایی پیدا کردید، نگاهی به نحوه‌ی اجرای یک برنامه در سیستم‌عامل و مواردی که یک سیستم‌عامل باید آنها را کنترل کند می‌اندازیم.

[21] Character
[22] Lower-case

سیستم‌عامل باید حافظه‌های Stack و Heap را - که به صورت پویا [۲۳] تخصیص داده می‌شوند - مدیریت کند. این حافظه‌ها تا چه حدی می‌توانند بزرگ باشند؟ جست‌وجو کنید که چگونه می‌توانیم این مقادیر را گرفته و تغییر دهیم. پرونده **limits.c** را به‌گونه‌ای تغییر دهید که بیشینه سایز Stack ، بیشینه تعداد پردازه‌ها و بیشینه تعداد توصیف‌گرهای پرونده‌ها [۲۴] را چاپ کند. اگر پرونده **limits.c** را اجرا کنید، تعداد زیادی از محدودیت‌های منابع سیستم را چاپ می‌کند که متاسفانه مقدار آنها صفر است. وظیفه‌ی شما این است که مقادیر واقعی این محدودیت‌ها را چاپ کنید.
راهنمایی: دستور **man getrlimit** را اجرا کنید.
خروجی برنامه شما باید مانند زیر باشد:

```
stack size: 8388608
process limit: 2782
max file descriptors: 1024
```

شما می‌توانید با اجرای دستور **make limits** کد خود را ترجمه کنید.

## ۳.۳ الف تا ض gdb

در این بخش می‌خواهیم از یک برنامه‌ی ساده به نام **map** برای انجام دادن تعدادی تمرین با gdb استفاده کنیم. قبل از شروع، نگاهی به پرونده‌های **map.c** و **recurse.c** که برنامه **map** را تشکیل می‌دهند بیاندازید. بعد از آنکه با چگونگی کارکرد برنامه آشنا شدید، با دستور **make map** آن را ترجمه کنید.
دستوراتی که برای کامل کردن هریک از قدم‌های زیر انجام می‌دهید را نوشته و فراموش نکنید که **جواب خود به سوالاتی که پررنگ شده‌اند** را در پرونده **gdb.txt** قرار دهید. توجه کنید که باید دستورات لازم برای اجرای گام‌هایی که پررنگ نشده‌اند را هم بنویسید وگرنه نمره کامل را از این سوال دریافت نخواهید کرد.

(آ) GDB را اجرا کنید.

(ب) یک وقفه [۲۵] در ابتدای اجرای برنامه قرار دهید.

(ج) برنامه را تا رسیدن به وقفه اجرا کنید.

(د) **argv به کجا اشاره می‌کند؟**

(ه) **در آدرس argv چه چیزی قرار دارد؟**

(و) برنامه را تا رسیدن به اولین فراخوانی **recur** ادامه دهید.

(ز) **آدرس حافظه تابع recur چیست؟**

(ح) برنامه را تا رسیدن به اولین فراخوانی **recur** ادامه دهید.

(ط) برنامه را تا رسیدن به یک عبارت شرطی if ادامه دهید.

(ی) نحوه‌ی نمایش را به حالت assembly تغییر دهید.

(ک) دستورات را تا رسیدن به دستور **callq** ادامه دهید.

(ل) **چه مقادیری در تمام رجیسترها وجود دارند؟**

(م) درون دستور فراخوانی وارد شوید.

(ن) به حالت نمایشی C تغییر حالت دهید.

(س) پشته‌ی فراخوانی [۲۶] را در این لحظه چاپ کنید. (راهنمایی: دستور **backtrace** چه کاری انجام می‌دهد؟)

(ع) حال یک وقفه بر تابع **recur** در حالتی که به‌ازای **0** بودن ورودی اجرا می‌شود قرار دهید.

(ف) ادامه دهید تا زمانی که به وقفه برسید.

[23] Dynamic
[24] Descriptors File
[25] Breakpoint
[26] Call Stack

(ص) پشته فراخوانی را در این لحظه چاپ کنید.

(ق) حال در پشته فراخوانی به بالا رفته تا جایی که به **main** برسید. مقدار **argc** چیست؟

(ر) تا رسیدن به دستور **return** پیش بروید.

(ش) به حالت نمایشی assembly جابه‌جا شوید.

(ت) **چه دستوری معادل "return 0" در زبان C است؟**

(ث) حال به نمایش کد منبع جابه‌جا شوید.

(خ) سه فراخوانی بعدی را اجرا کنید.

(ذ) برنامه را تا آخر اجرا کنید.

(ض) از gdb خارج شوید.

## ۴.۳ Compiling, Assembling, and Linking

حالا که با نحوه‌ی کارکرد **map** آشنا شدید، چه چیزهایی برای رفتن از C به یک پرونده اجرایی می‌خواهیم.

۱۰ سوال در این بخش وجود دارند که باید پاسخ‌های خود به آنها را در پرونده **call.txt** نوشته و در مخزن خود قرار دهید.

ابتدا اجازه دهید که با تعدادی از پرچم‌های مربوط به مترجم آشنا شویم:

- **-Wall** تمام هشدارهای مترجم را فعال می‌کند.
- **-m32** کد را برای معماری **i386** ترجمه می‌کند
- **-S** فقط مترجم [۲۷] را صدا می‌زند.
- **-c** هم مترجم و هم اسمبلر [۲۸] را صدا می‌زند.

حال با اجرای مترجم شروع می‌کنیم. مترجم یک پرونده C گرفته و آن را به پرونده‌های اسمبلی برای معماری‌های **8086** یا **i386** تبدیل می‌کند.
برای ترجمه پرونده **map.c** دستور زیر را اجرا کنید:

```
$ gcc -m32 -S -o map.S map.c
```

این دستور مترجم را فراخوانی می‌کند و به او دستور می‌دهد تا پرونده‌ی **map.c** را به زبان اسمبلی ترجمه کند و نتیجه را در پرونده **map.S** نوشته و ذخیره کند.

۱. **پرونده اسمبلی recurse.S را ساخته و بنویسید کدام دستورالعمل(ها) معادل فراخوانی بازگشتی recur(i-1) هستند.**

حال ما کد اسمبل‌شده خود را تبدیل به یک برنامه اجرایی می‌کنیم. به این منظور، دستور زیر را اجرا کنید:

```
$ gcc -m32 -c map.S -o map.obj
```

[27] Compiler
[28] Assembler

این دستور پرونده اسمبلی ما را تبدیل به یک پرونده اجرایی می‌کند. بدیهتاً می‌توانستیم این دو دستور را با استفاده از پرچم `-c` با هم ترکیب کرده و مستقیماً این پرونده اجرایی را بسازیم.

اسمبلر پرونده اسمبلی خام را تبدیل به کدهای ماشین و دیگر شبه‌داده [29]ها می‌کند که برای اجرای پرونده نیاز هستند. سیستم‌عامل‌های مختلف از پرونده‌های اجرایی مختلفی پشتیبانی می‌کنند. در این کلاس ما از پرونده‌های اجرایی ELF استفاده می‌کنیم که توسط Linux استفاده می‌شود.
حال نگاهی به درون پرونده‌های **`map.out`** و **`recurse.out`** می‌اندازیم. این‌ها پرونده‌ها، باینری هستند در نتیجه از ابزار **`objdump`** برای خواندن محتویات آنها استفاده می‌کنیم.

۲. **بخش‌های `.section` و `.text` چه مواردی را شامل می‌شوند؟**

اسمبلر یک جدول نمادها [30] می‌سازد که بخشی از آبجکت فایل است. جدول نمادها شامل تمام نمادهایی است که می‌توانند به صورت عمومی توسط دیگر آبجکت فایل‌ها استفاده شوند.

۳. **چه دستوری جدول نمادهای یک پرونده ELF را نمایش می‌دهد؟**

خلاصه‌ای از جدول نمادهای پرونده **`map.obj`** در زیر نمایش داده شده است:

```
00000000 g O .data 00000004 stuff
00000000 g F .text 00000060 main
...
00000000 *UND* 00000000 malloc
00000000 *UND* 00000000 recur
```

۴. **پرچم‌های `g` ، `O` و `*UND*` چه معنایی دارند؟**

۵. **توضیح دهید که چگونه می‌توان آدرس دیگری برای نماد `malloc` پیدا کرد.**

۶. **چه جای دیگری می‌توانیم نماد `recur` را پیدا کنیم؟ در چه پرونده‌ای؟ جدول نماد مربوط به این بخش را در جواب خود قرار دهید.**

حال دو پرونده را با دستور زیر به یک‌دیگر پیوند می‌دهیم تا یک پرونده اجرایی تولید شود:

```
$ gcc -m32 map.obj recurse.obj -o map
```

۷. **جدول نمادها را برای برنامه‌ی `map` در نظر بگیرید. چه چیزهایی تغییر کرده است؟**

# ۴ تحویل‌دادنی‌ها

شما باید مستندات و پرونده‌های مربوط به هر یک از بخش‌های زیر را در مخزن خصوصی مربوط به خود و زیر پوشه‌ی **`hw0`** قرار دهید:

- گزارش کلی از روند انجام تمرین بدون توضیحات اضافه
- پرونده‌های تغییریافته مربوط به تمرین **`words`** شامل **`main.c`** و **`word_count.c`**
- توضیحات مربوط به **`Makefile`** در پرونده **`Makefile.txt`**
- پرونده تغییریافته **`limits.c`**
- توضیحات و جواب‌های مربوط به بخش **`gdb`** در پرونده **`gdb.txt`**

[29] Metadata
[30] Symbol Table

- توضیحات و جواب‌های مربوط به بخش **Compiling, Assembling, and Linking** در پرونده **call.txt**

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