به نام خداوند بخشنده مهربان

دانشگاه صنعتی شریف
مدرس: مهدی خرازی

تمرین صفرم *
سیستم‌های عامل

دانشکده‌ی مهندسی کامپیوتر
پاییز ۱۳۹۷

اهداف تمرین

- راه اندازی پیش‌نیازهای انجام تمارین
- آشنایی با چند ابزار مفید برای تولید کد و رفع باگ
- آشنایی با نحوه ارسال تمارین

۱. مقدمه

این تمرین شامل سه بخش اصلی می‌شود که عبارتند از: نصب پیش‌نیازهای انجام تمرین‌ها، آشنایی با چند ابزار مفید و چند تمرین ابتدایی و ساده.

* با سپاس از زحمات تیم دستیاران تمرین

۲. راه‌اندازی مقدمات

۱.۲ Gogs

تمامی تمارین فردی و گروهی شما از طریق این سامانه دریافت می گردد. بنابراین شما نیازمند یک حساب کاربری هستید. تیم دستیاران تمرین برای شما مخازن خصوصی می‌سازند و اطلاعات آن را در اختیار شما قرار می‌دهند.

۲.۲ Vagrant

تیم دستیاران تمرین تصویری [۱] از ماشین مجازی لازم جهت تست و اجرای تمامی کدها فراهم کرده است. Vagrant ابزاری جهت مدیریت ماشین‌های مجازی است. شما می‌توانید از Vagrant برای بارگیری و اجرای ماشین‌های مجازیِ آماده، استفاده کنید.

نکته: اگر از ویندوز استفاده می کنید، می‌توانید از shell provisioning استفاده کنید یا به زیربخش ۱.۲.۲ مراجعه کنید. در ادامه در خصوص راه‌اندازی برروی ویندوز توضیحاتی بیان خواهد گردید.

- راه‌اندازی Vagrant نیازمند نصب و راه ندازی VirtualBox است. بنابراین شما نیاز دارید که مناسب ترین نسخه‌ی آن را از اینجا بارگیری و نصب کنید. در کلاس درس بیشتر در مورد ماشین‌های مجازی توضیح داده خواهد شد، اما فعلاً می‌توانید تصور کنید که منظور از یک ماشین مجازی، نسخه نرم‌افزاری یک سخت افزار واقعی است.
- نسخه مناسب Vagrant را از اینجا نصب کنید.
- طبق دستورات گفته شده در اینجا عمل کنید.
- می‌بایست تمامی دستورهای Vagrant را از مخزن vagrant اجرا کنید و مواظب باشید که این مسیر را پاک نکنید.
- به منظور متوقف کردن ماشین مجازی می‌توانید از دستور `vagrant halt` استفاده کنید.

۱.۲.۲ Windows

از آنجایی که سیستم عاملِ ویندوز از ssh پشتیبانی نمی‌کند، بارگیری و نصب Cygwin می‌تواند گزینه مناسبی باشد. شما می‌توانید از اینجا راهنمای نصب تنظیمات Vagrant در ویندوز به کمک Cygwin را مشاهده نمایید.

[۱] image

Troubleshooting Vagrant .۲.۲.۲

اگر در دستور

```
vagrant up
```

با مشکل مواجه شد، تلاش کنید تا با اجرای دستور

```
vagrant provision
```

مشکل را برطرف نمایید. اگر کماکان مشکل برطرف نشد، برای تلاش آخر می‌توانید به کمک دستور

```
vagrant destroy
```

ماشین مجازی خود را از کار بیندازید و مجدداً تلاش کنید تا دستور زیر اجرا شود.

```
vagrant up
```

Git Config .۳.۲.۲

دستورهای زیر را داخل ماشین مجازی اجرا کنید تا تنظیماتی که برای کامیت‌های خود [۲] استفاده می‌کنید برقرار گردد.

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

ssh-key .۴.۲.۲

در این مرحله نیاز دارید که کلیدهای ssh خود را به منظور احراز هویت از درون ماشین مجازی خود تنظیم کنید. برای این کار دو دستور زیر را به ترتیب اجرا کنید :

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

دستور اول یک جفت کلید ssh برای شما تولید می‌کند. دستور دوم کلید عمومی‌تان را در ترمینال نمایش می‌دهد. شما می‌بایست به این سامانه ورود کرده و سپس از این قسمت کلید عمومی خود را به حساب خود اضافه کنید. کلید شما باید با عبارت ssh-rsa شروع شده و با «vagrant@development» پایان یافته باشد.

[۲] Commit

Repos ۵.۲.۲

برای دسترسی به پرونده‌های مورد نیاز هر تمرینِ درس یک مخزن عمومی ساخته شده است. شما قبلاً از این مخزن برای بارگیری vagrant و نحوه‌ی کار آن استفاده کرده‌اید. این مخزن از آدرس زیر در دسترس است:

```
git@tarasht.ce.sharif.ir:ce424-971-students/ce424-971-handouts.git
```

همه‌ی پرونده‌های مورد نیاز شما در پوشه‌ی مربوط به تمرین است. این مخزن در حال حاضر در آدرس «/code/ce424-971-handouts» از ماشین مجازی شما قرار دارد.
برای به‌روزرسانی این مخزن می‌توانید از این دستور استفاده کنید:

```
$ cd /code/ce424-971-handouts
$ git pull origin
```

برای اتصال به مخازنی که از قبل برای شما به صورت خصوصی تعریف شده کافی است به مسیر دلخواه (در این مستند فرض کرده‌ایم در مسیر /code/~ مخزن‌های گیت را بارگیری می‌کنید) خود رفته و دستورات زیر را اجرا کنید:

```
git clone REPO-URL
```

که PERSONAL-REPO-URL ساختاری شبیه به:

```
git@tarasht.ce.sharif.ir:ce424-971-students/ce424-971-"student-id".git
```

و GROUP-REPO-URL ساختاری شبیه به:

```
git@tarasht.ce.sharif.ir:ce424-971-groups/ce424-971-"group-id".git
```

خواهند داشت.

Issues ۶.۲.۲

استفاده از Issue بهترین روش جهت پیگیری و دنبال کردن روند وظایف، پیشرفت‌ها و یا رفع مشکلات یک پروژه است، به گونه ای که می‌توان آن‌ها را جهت بحث و گفت و گو میان اعضای تیم به اشتراک گذاشت. در سامانه Gogs هر مخزنی یک بخش مجزا برای این قسمت دارد.
هر Issue را می‌توان به یکی از اعضای تیم assign و میزان پیشرفت آن را به‌روزرسانی کرد.

۳.۲. ویرایش کد در ماشین مجازی

ماشین مجازی قابلیت انطباق پوشه home در سیستم عامل میزبان با پوشه vagrant/ در سیستم عامل اصلی را به شما می دهد. به کمک این امکان شما می‌توانید از هر ویرایشگر متنی که بر روی سامانه‌ی خود دارید استفاده کنید تا کدها را ویرایش کرده و دستورهای git را از درون ماشین مجازی خود اجرا کنید. این روش پیشنهادی برای کار برروی کدها در این درس است، اما شما آزادید که از هر روشی که به نظرتان مناسب‌تر است استفاده کنید.

۳. چند ابزار مفید

در این بخش با چند ابزار مفید آشنا می‌شوید که معمولاً در جعبه ابزار هر کاربر پیش‌رفته‌ای یافت می شود. از میان این ابزار یادگیری git و make الزامی هستند، زیرا بدون کسب دانش کافی از این ابزار قادر نخواهید بود کد خود را کامپایل نموده و سپس ارسال کنید. سایر ابزار یا در جهت رفع باگ به کار می‌روند یا در جهت چندوظیفگی[۳] با کارایی بیشتر مورد استفاده قرار می‌گیرند.

نکته: تمامی این ابزارها بر روی ماشین مجازی نصب گردیده‌اند.

۱.۳ Git

یک برنامه مدیریت نسخه [۴] است که به کمک آن می‌توانید روند کدها را دنبال کنید. GitHub یکی از سامانه های تحت وب است که امکان hosting کدهای شما را برایتان فراهم می‌کند. درواقع این سامانه فضای تعاملی و اشتراک گذاری کدها را فراهم ساخته است. اگر چه تا این مرحله شما تنظیمات مقدماتی را انجام داده اید، اما تسلط شما به قابلیتهای git می‌تواند در طول این درس به کمک شما بیاید، خصوصاً هنگامی که با هم گروهی هایتان تمرین را انجام می دهید. برای افزایش سطح دانش خود نسبت به گیت می‌توانید اینجا را مشاهده کنید.

۲.۳ make

make ابزاری است که به صورت خودکار برنامه های اجرایی و کتابخانه‌ها را از کد منبع تولید می‌کند و این کار را به کمک خواندن فایل Makefile انجام می دهد. Makefile تعیین می کند که چگونه به برنامه هدف دسترسی پیدا کند. به این صورت که فهرست تمامی وابستگی‌ها[۵] را در آن قرار می دهید و make با پیمایش آن‌ها برنامه اجرایی شما را تولید می‌کند. متاسفانه make ، ساختار بسیار پیچیده‌ای دارد که اگر به صورت درست از آن‌ها استفاده نکنید برای فهم آنها دچار مشکل خواهید شد. برای یادگیری make می‌توانید اینجا و اینجا را مشاهده کنید. هم‌چنین مستندسازی رسمی GNU هم از اینجا قابل دسترسی‌ست که مسلماً حجم بیشتری دارد. فعلاً ما از ساده‌ترین فرم make که نیازی به Makefile ندارد استفاده می کنیم. بنابراین با اجرای دستور زیر میتوانید کد خود یعنی wc.c را به راحتی کامپایل و لینک کنید.

```
make wc
```

این دستور یک پرونده‌ی اجرایی ایجاد می‌کند که شما می‌توانید اجرایش کنید. حال دستور زیر را اجرا کنید:

```
./wc wc.c
```

تفاوت دستور بالا با دستوری که در ادامه می آید چیست؟ (راهنمایی: ابتدا دستور which wc را اجرا کنید.)

[۳] multitasking
[۴] control version
[۵] dependency

```
wc wc.c
```

تمرین اول شما این خواهد بود که wc.c را به گونه ای تغییر دهید که تعداد کلمات را بشمارد. درواقع می‌توانید با توجه به `man wc` عملکرد آن را پیاده سازی کنید. توجه کنید که نیاز به پشتیبانی هایflag مختلف ندارید و تنها باید از یک پرونده‌ی ورودی پشتیبانی کنید (یا STDIN اگر هیچ نامی تعیین نشده بود).

توجه کنید که wc در macOS کاملاً متفاوت از Ubuntu عمل می کند، بنابراین انتظار می رود که رفتار آن را در Ubuntu شبیه‌سازی کنید.

gdb .۳.۳

دیباگ کردن برنامه های با زبان C بسیار سخت است اما خوشبختانه gdb امکان دیباگ کردن آسان را فراهم نموده است. اگر شما برنامه هایتان را با پرچم[۶] خاص g- کامپایل کنید و برنامه را داخل gdb اجرا کنید علاوه بر trace stack می‌توانید متغیرها را inspect کنید، تغییر دهید، کد را متوقف کنید و ... .

gdb ساده امکانات کمی دارد، به همین دلیل cgdb روی ماشین مجازی برایتان نصب شده است که یک سری قابلیت‌ها از قبیل syntax highlighting را داراست. در cgdb میتوانید با استفاده از i و ESC بین پنجره[۷] های بالایی و پایینی switch کنید. همچنین gdb می‌تواند پردازه های جدید را شروع کند و به پردازه‌های درحال اجرا ملحق کند.

برای یادگیری gdb پیشنهاد می‌کنیم اینجا را مشاهده کنید. البته مستند اصلی gdb نیز علی رغم طولانی بودن مفید است.

برای یادگیری بیشتر تلاش کنید با wc.c کار کنید. با پرچم[۸] g- برنامه‌ی خود را کامپایل کنید. برنامه را از طریق gdb شروع کنید و یک نقطه وقفه[۹] بر سر main بگذارید. سپس برنامه را تا آن‌جا اجرا کنید و دستورهای مختلف را تمرین کنید. بفهمید که چگونه آرگومان های خط فرمان را می‌توان pass داد. متغیرهای محلی اضافه کنید و مقادیر[۱۰] نظیرشان را پیدا کنید. step ، next و break را فرابگیرید.

tmux .۴.۳

یک multiplexer مربوط به terminal است که چندین tab مربوط به terminal را شبیه سازی می کند اما آن‌ها را در یک session از terminal نمایش می‌دهد. البته این چند tab را هنگام ssh به ماشین مجازی حفظ می‌کند. شما می‌توانید یک session جدید را با دستور زیر ایجاد کنید:

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

[۶] flag
[۷] pane
[۸] flag
[۹] break point
[۱۰] value

سپس می‌توانید با فشار دادن کلیدهای :

```
ctrl-b + c
```

یک پنجره[11] جدید ایجاد کنید و با فشار دادن کلیدهای :

```
ctrl-b + n
```

به پنجره n ام پرش کنید.
اگر کلیدهای :

```
ctrl-b + d
```

را فشار دهید از session مربوط به tmux جدا می‌شوید درحالیکه کماکان درحال اجراست و هر برنامه ای درون آن نیز درحال اجراست. برای آنکه session خود را ادامه دهید می‌توانید از دستور زیر استفاده کنید:

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

برای شروع به یادگیری tmux می‌توانید اینجا را مشاهده کنید.

vim .۵.۳

یک ویرایشگر متن زیبا برای استفاده درون terminal است. بسیاری نیز emacs را بر vim ترجیح می دهند اما آنچه که اهمیت دارد آن است که در یک ویرایشگر به تسلط برسید تا بتوانید در نوشتن کدها از آن بهره ببرید. برای تسلط در vim می‌توانید اینجا را مطالعه کنید.

ctags .۶.۳

از آنجاکه تعداد خط کد بسیاری را خواهید خواند این ابزار در استفاده از وقت‌تان صرفه‌جویی می‌کند و سبب می‌گردد که بین قطعات مختلف کد به راحتی جابه‌جا شوید.

دستورالعمل نصب این ابزار را برای vim از اینجا و برای sublime از اینجا مشاهده کنید. البته از ویرایشگرهای دیگر نیز پشتیبانی می کند که در این صورت می‌بایست دستورالعمل مرتبط را جستجو کنید.

[11] window

۴. تمارین مقدماتی

۱.۴ make

احتمالاً از gcc برای کامپایل برنامه هایتان استفاده کرده‌اید، اما این کار وقتی تعداد پرونده‌ها افزایش یابد پیچیده و خسته کننده می‌شود. برای این تمرین نیاز دارید که یک Makefile بنویسید که با اجرای دستور make، پرونده‌های main.c ، wc.c و map.c (بدنه اصلی این پرونده‌ها از مخزن تمرین قابل دریافت است و بعداً در طول تمرین آن‌ها را تکمیل خواهید کرد) را کامپایل کند. ممکن است برای این مرحله به پرچم g- نیاز پیدا کنید. نوشتن یک target برای پاک کردن پرونده‌های باینری بخش اختیاری این‌قسمت از تمرین است.

۲.۴ wc

اولین اقدامی که باید انجام دهید این است که یک clone از ابزار wc بنویسید که تعداد خطوط، کلمات و کاراکترهای یک پرونده‌ی text را بشمارد. می‌توانید wc خود را در ماشین مجازی اجرا کنید تا ببینید خروجی باید چه قالبی داشته باشد، سپس تلاش کنید که از عملکردهای[۱۲] اصلی در wc.c تقلید کنید (از بابت پرچم‌ها و فاصله‌گذاری‌ها در خروجی نگران نباشید).

تنها نیاز دارید که اجرای «wc FILE_NAME» و wc (که باید داده را از ورودی استاندارد بخواند) را پشتیبانی کنید.

در طول مدت زمانی که درحال کار روی این بخش هستید توصیه می‌شود نهایت استفاده را از gdb ببرید.

۳.۴ پرونده‌های اجرایی و آدرس‌ها

۱.۳.۴ gdb

پرونده‌ی اجرایی wc خود را در gdb با یک پرونده‌ی ورودی از طریق آرگومان خط فرمان بار[۱۳] کنید. سپس نقطه وقفه‌ای روی main بگذارید و برنامه را شروع کنید. فرآیند اجرا را تا زمانی که در میانه‌ی اجرای برنامه هستید خط به خط ادامه دهید. با استفاده از where یا backtrace (bt) پشته[۱۴] را بررسی کنید. هنگامی که درحال استفاده از gdb هستید به سوالات زیر فکر کنید و پاسخ را در فایل gdb.txt قرار دهید.

- مقدار argv چند است؟(راهنمایی: print argv)
- argv به کجا اشاره می کند؟(راهنمایی: print argv[۰])
- آدرس تابع main چیست؟

[۱۲] functionality
[۱۳] load
[۱۴] stack

- دستور stack info را اجرا کنید و مشاهدات خود را بیان کنید.
- دستور frame info را اجرا کنید و مشاهدات خود را بیان کنید.
- دستورregisters info را اجرا کنید. کدام ثبات‌ها[۱۵] ویژگی‌های برنامه‌ای که می‌شناسید را در بر دارد؟

۲.۳.۴. objdump

برای آنکه جزئیات بیشتری از برنامه‌ی خود را ببینید کافی است برای مثال دستور زیر را اجرا کنید:

```
objdump -x -d wc
```

پس از اجرای این دستور مشاهده می کنید که برنامه چندین سگمنت دارد و نام توابع و متغیرهای برنامه به همراه آدرس‌ها و مقادیر نظیرشان نمایش داده می شوند. در خروجی objdump این سگمنت‌ها زیر بخش[۱۶] heading قرار دارند. هنگامی که درحال استفاده از objdump هستید به سوالات زیر فکر کنید و پاسخ را در فایل objdump.txt قرار دهید.

- از چه قالب پرونده‌ای برای این باینری استفاده می شود؟ و برای چه معماری کامپایل می شود؟
- تعدادی از نام‌های سگمنت‌ها/بخش‌هایی که یافتید را نام ببرید.
- چه سگمنت/بخشی در بردارنده تابع main است؟ و آدرس main چیست؟ (می‌بایست همان آدرسی باشد که در gdb مشاهده کرده‌اید)
- آیا سگمنتی که مربوط به پشته یا heap باشد را مشاهده کردید؟ چگونه؟

۳.۳.۴. map

حال شما قادرید یک برنامه را که ساختار اجرایی خود را نشان دهد بنویسید. پرونده‌ی دومی که در hw۰ قرار دارد (map.c) یک ساختار تقریباً کامل را فراهم می‌کند. شما نیاز دارید که این پرونده را به گونه ای تغییر دهید که آدرس‌هایی که به دنبالشان می‌گردید را پیدا کنید. خروجی این قسمت شبیه زیر است (آدرس‌ها ممکن است تفاوت داشته باشند).

```
_main @ 0x4005c2
recur @ 0x40057d
_main stack: 0x7fffdf73c
static data: 0x601048
```

[۱۵] Register
[۱۶] section

```
Heap: malloc 1: 0x671010
Heap: malloc 2: 0x671080
recur call 3: stack@ 0x7fffda11f6fc
recur call 2: stack@ 0x7fffda11f6cc
recur call 1: stack@ 0x7fffda11f69c
recur call 0: stack@ 0x7fffda11f66c
```

حال به سوالات زیر فکر کنید و پاسخ را در فایل map.txt قرار دهید.

- از objdump با پرچم D- روی پر ونده‌ی اجرایی map استفاده کنید. کدام آدرس‌ها از خروجی اجرای map/. در پرونده‌ی اجرایی تعریف شده اند؟ و هریک در چه سگمنت/بخش؟
- فهرستی از سگمنت‌های مهم و کاربردشان تهیه کنید.
- پشته در چه جهتی افزایش می یابد؟
- stack frame برای هر فراخوانی بازگشتی[۱۷] چقدر است؟
- آیا دو ناحیه از حافظه که malloc شده اند پیوسته اند؟(فضای آدرسی اضافی مابین آنها قرار ندارد؟)

۴.۴ محدودیت‌های کاربر

سیستم‌عامل می‌بایست با سایز سگمنت‌هایی نظیر پشته و heap به دلیل پویایی آن‌ها هماهنگ باشد. سایز آن‌ها چه مقدار باید باشد؟ کمی جستجو کنید و بررسی کنید که این محدودیت‌ها روی linux چگونه set و get می‌شوند.

main.c را به گونه ای تغییر دهید که بیشینه سایز پشته، بیشینه تعداد پردازه‌ها و بیشینه تعداد توصیف‌گرهای پرونده را چاپ کند. درحال حاضر اگر آن‌را اجرا کنید مشاهده می‌کنید که قسمتی از محدودیت‌های منابع سامانه را چاپ می‌کند(stack size ، heap size و ...). متاسفانه تمامی این مقادیر صفر هستند و شما باید کاری کنید که این اعداد مقادیر واقعی باشند. برای این کار می‌توانید از soft limit ها به جای hard limit ها استفاده کنید.(راهنمایی: دستور " man getrlimit " را اجرا کنید.) خروجی مورد انتظار باید شبیه زیر باشد:

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

[۱۷] recursive

۵. تحویل دادنی‌ها

- گزارشی از روند انجام تمرین به همراه پاسخ سوالات مطرح شده در متن تمرین (یک پرونده با قالب PDF و بدون توضیحات اضافی به نام report.pdf)
- Makefile مربوط به بخش ۱.۴
- Clone مربوط به wc
- فایل gdb.txt
- فایل objdump.txt
- فایل map.txt
- فایل map.c
- فایل main.c مربوط به بخش ۴.۴

به ازای هر تمرین در مخزن شخصی خود یک پوشه با حروف کوچک و با شماره‌ی تمرین بسازید، سوالات تمرین را پاسخ داده و همه‌ی پرونده‌های لازم را با همان نامی که در مستند تمرین ذکر شده است جهت نمره دهی با دستورهای زیر ارسال کنید:

```
cd ~/code/personal/hw0
git status
git add report.pdf wc.c main.c map.c gdb.txt objdump.txt map.txt Makefile
git commit -m "Finished my first OS assignment"
git push origin master
```