

## lab0 实验报告

### 一、实验目的

1. 了解操作系统实验开发环境和基本工具
2. 熟悉命令行方式的编译、调试
3. 掌握基于硬件模拟器的调试技术

### 二、实验过程

1. 安装配置实验开发环境：在已有的 VirtualBox 中创建新虚拟机，配置虚拟镜像文件 mooc-os-2015.vdi

2. 使用实验工具

#### 1) shell 命令行

Shell 本身是一个用 C 语言编写的程序，是用户使用 Linux 的桥梁。bash 是 Linux 标准默认的一种 shell，是 BourneAgain Shell 的缩写，内部命令一共有 40 个。单击屏幕左边的图标或者 Shift+Ctrl+T 可以进入 shell 命令行

#### ● ls (查询文件列表)

——默认状态下按首字母升序列出当前文件夹下面所有内容

当前用户主目录下有 8 个文件夹和文件

```
[~]
moocos-> ls
backup  Documents  examples.desktop  moocos  Pictures  readme.txt  Templates  tools  Videos
Desktop Downloads  local           Music   Public   README.txt  test     usefulshell workspace
```

ls -l 列出一个更详细的文件清单，包括读写权限、建立日期等

```
[~]
moocos-> ls -l
total 76
drwxrwxr-x 3 moocos moocos 4096 3月 2 2015 backup
drwxr-xr-x 2 moocos moocos 4096 9月 23 10:49 Desktop
drwxr-xr-x 2 moocos moocos 4096 9月 4 2014 Documents
drwxr-xr-x 2 moocos moocos 4096 3月 2 2015 Downloads
-rw-r--r-- 1 moocos moocos 8980 9月 4 2014 examples.desktop
drwxrwxr-x 4 moocos moocos 4096 2月 13 2015 local
drwxrwxr-x 4 moocos moocos 4096 3月 2 2015 moocos
drwxr-xr-x 2 moocos moocos 4096 9月 4 2014 Music
drwxr-xr-x 2 moocos moocos 4096 9月 4 2014 Pictures
drwxr-xr-x 2 moocos moocos 4096 9月 4 2014 Public
-rw-r--r-- 1 moocos moocos 89 3月 2 2015 readme.txt
lrwxrwxrwx 1 moocos moocos 10 3月 2 2015 README.txt -> readme.txt
drwxr-xr-x 2 moocos moocos 4096 9月 4 2014 Templates
-rw-rw-r-- 1 moocos moocos 7 9月 28 2014 test
drwxrwxr-x 5 moocos moocos 4096 2月 13 2015 tools
drwxrwxr-x 2 moocos moocos 4096 9月 4 2014 usefulshell
drwxr-xr-x 2 moocos moocos 4096 9月 4 2014 Videos
drwxrwxr-x 3 moocos moocos 4096 2月 10 2015 workspace
```

后面接路径可以把该参数当作命令行的工作目录

```
[~]
moocos-> ls moocos
tools ucore_lab
[~]
moocos-> ls moocos/ucore_lab
labcodes labcodes_answer LICENSE README-english.md README.md related_info
```

进入系统根目录

```
[~]
moocos-> ls /
bin  cdrom  etc  initrd.img  lib32  libx32  media  opt  root  sbin  sys  usr  vmlinuz
boot  dev  home  lib  lib64  lost+found  mnt  proc  run  srv  tmp  var
```

- pwd (查询当前所在目录)
- cd (进入其它目录, 直接路径或间接路径都可以)
  - cd /进入系统根目录
  - cd ..进入上一级目录
  - cd ~进入当前用户主目录
  - cd -返回此前所在的目录

```
[~]
moocos-> pwd
/home/moocos
[~]
moocos-> cd moocos/ucore_lab
[~/moocos/ucore_lab]
moocos-> cd /home/moocos/moocos/ucore_lab
[~/moocos/ucore_lab]
moocos-> pwd
/home/moocos/moocos/ucore_lab
[~/moocos/ucore_lab]
moocos-> cd ..
[~/moocos]
moocos-> cd -
/home/moocos/moocos/ucore_lab
[~/moocos/ucore_lab]
moocos-> cd /
[/]
moocos-> cd ~
[~]
moocos-> 
```

- mkdir (建立一个目录)

```
[~]
moocos-> mkdir lab0_test
[~]
moocos-> ls
backup      examples.desktop  Music      README.txt  usefulshell
Desktop     lab0_test         Pictures   Templates   Videos
Documents   local             Public     test         workspace
Downloads   moocos            readme.txt tools
```

- touch (新建一个空文本文件)

```
[~]
moocos-> cd lab0_test
[~/lab0_test]
moocos-> touch temp.txt
[~/lab0_test]
moocos-> ls
temp.txt
```

- cp (复制文件)

```
[~/lab0_test]
moocos-> ls
temp.txt
[~/lab0_test]
moocos-> cp temp.txt temp1.txt
[~/lab0_test]
moocos-> ls
temp1.txt  temp.txt
```

- mv (移动文件或重命名)

```
moocos-> ls
temp1.txt temp.txt
[~/lab0_test]
moocos-> mv temp.txt temp0.txt
[~/lab0_test]
moocos-> ls
temp0.txt temp1.txt
[~/lab0_test]
moocos-> mv temp1.txt /home/moocos
[~/lab0_test]
moocos-> ls
temp0.txt
[~/lab0_test]
moocos-> ls ~
backup      examples.desktop  Music      README.txt  tools
Desktop     lab0_test        Pictures   temp1.txt   usefulshell
Documents   local            Public     Templates   Videos
Downloads   moocos           readme.txt test         workspace
```

- rm (删除文件/目录)

-i 用于在执行前输出一条确认提示; 删除文件夹时需要加上 -R

```
[~/lab0_test]
moocos-> ls
apple temp0.txt temp2.txt
[~/lab0_test]
moocos-> rm -i temp2.txt
rm: remove regular empty file 'temp2.txt'? y
[~/lab0_test]
moocos-> ls
apple temp0.txt
[~/lab0_test]
moocos-> rm -i apple
rm: cannot remove 'apple': Is a directory
[~/lab0_test]
moocos-> rm -R apple
[~/lab0_test]
moocos-> ls
temp0.txt
```

每次都要加上 -i 太麻烦, 执行如下命令, 让 -i 成为默认参数

```
[~]
moocos-> alias rm='rm -i'
[~]
moocos-> cd lab0_test
[~/lab0_test]
moocos-> rm temp0.txt
rm: remove regular file 'temp0.txt'?
```

- cat (显示文件内容)

```
[~/lab0_test]
moocos-> cat temp0.txt
Five nuts mooncake is the most delicious mooncake in the world!
```

more/less 用于分页查看较大的文件

- find (文件查找)  
——find . -name "\*\*\*"可以根据关键字查找

```
[~/moocos]
moocos-> find . -name "*.txt"
./ucore_lab/labcodes_answer/lab1_result/report.txt
./tools/scitools/conf/plugin/SciTools/Codecheck/SciTools' Recommended Checks/de
./tools/scitools/conf/plugin/SciTools/Codecheck/Published Standards/description
./tools/scitools/conf/plugin/SciTools/Codecheck/Published Standards/MISRA-C++ 2
./tools/scitools/conf/plugin/SciTools/Codecheck/Published Standards/MISRA-C 200
./tools/scitools/conf/plugin/SciTools/Codecheck/Published Standards/Effective C
./tools/scitools/conf/license/users.txt
./tools/scitools/conf/license/readme.txt
./tools/scitools/conf/understand/plm/predeclared.txt
./tools/scitools/conf/understand/javascript/nodejs_predefined.txt
```

- echo (在屏幕上输出字符)

```
[~]
moocos-> echo "Hello, world!"
Hello, world!
```

现在看起来好像没多大实际意义，或许用于提示其他人上一步很长的执行做了什么？

- ps (查询当前进程)

```
[~]
moocos-> ps -a
  PID TTY          TIME CMD
 3944 pts/0    00:00:00 ps
```

## 2) 系统维护工具: apt, git

apt-get 自动从互联网软件库中搜索、安装、升级以及卸载软件，一般需要 root 执行权限

```
[~] Files
moocos-> sudo apt-get install gcc
Reading package lists... Done
Building dependency tree
Reading state information... Done
gcc is already the newest version.
The following packages were automatically installed and are no longer required:
  gyp javascript-common libc-ares-dev libc-ares2 libjs-node-uuid libssl-dev
  libssl-doc libv8-3.14-dev libv8-3.14.5
Use 'apt-get autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 845 not upgraded.
```

说明 gcc 已经安装好了并且是最新版本

git 作为一个版本控制程序可以用于查看、维护、回退旧版本的文件，或者检查修改是否成功，可以说是非常好用了。

首先 apt-get git, git --version 查看版本

```
[~]
moocos-> git --version
git version 1.9.1
```

然后输入名字和邮箱配置 git;

```
[~]  
moocos-> git config --global user.name "Jiawei Qi"  
[~]  
moocos-> git config --global user.email "16307130293@fudan.edu.cn"
```

git config -l 可以查看刚才的信息

```
[~]  
moocos-> git config --list  
user.email=16307130293@fudan.edu.cn  
user.name=Jiawei Qi
```

新建文件夹 git\_test, 将其放在 git 的管理之下, 完成一个本地仓库的建立和初始化, 该目录下有一个文件夹.git/的生成, 可以通过 ls -l 查看其内容

```
[~]  
moocos-> mkdir git_test  
[~]  
moocos-> cd git_test  
[~/git_test]  
moocos-> git init  
Initialized empty Git repository in  
/home/moocos/git_test/.git/
```

```
[~/git_test]  
moocos-> ls -l .git/  
total 44  
drwxrwxr-x  2 moocos moocos 4096  9月 25 14:16 branches  
-rw-rw-r--  1 moocos moocos  14  9月 25 14:53 COMMIT_EDITMSG  
-rw-rw-r--  1 moocos moocos  92  9月 25 14:16 config  
-rw-rw-r--  1 moocos moocos  73  9月 25 14:16 description  
-rw-rw-r--  1 moocos moocos  23  9月 25 14:16 HEAD  
drwxrwxr-x  2 moocos moocos 4096  9月 25 14:16 hooks  
-rw-rw-r--  1 moocos moocos 104  9月 25 14:53 index  
drwxrwxr-x  2 moocos moocos 4096  9月 25 14:16 info  
drwxrwxr-x  3 moocos moocos 4096  9月 25 14:23 logs  
drwxrwxr-x 10 moocos moocos 4096  9月 25 14:53 objects  
drwxrwxr-x  4 moocos moocos 4096  9月 25 14:16 refs
```

在 git\_test 目录下新建测试文件 hello.c

```
1#include<stdio.h>  
2int main()  
3{  
4    printf("Hello,world!\n");  
5    return 0;  
6}
```

此时执行会出现红色标注没有添加到 git 索引中的文件, 使用 add 后文字变绿说明文件添加到了索引中

```

[~/git_test]
moocos-> git status
On branch master

Initial commit

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        hello.c

nothing added to commit but untracked files present (use "git add" to track)
[~/git_test]
moocos-> git add hello.c
[~/git_test]
moocos-> git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   hello.c

```

接下来使用 commit 来提交

```

[~/git_test]
moocos-> git commit -m "First commit"
[master (root-commit) 424f6fb] First commit
 1 file changed, 6 insertions(+)
 create mode 100644 hello.c
[~/git_test]
moocos-> git status
On branch master
nothing to commit, working directory clean

```

可以使用 git log 查看提交记录

```

commit 424f6fb936f849a845bb361ea4dfdf7cc6bbfde5
Author: Jiawei Qi <16307130293@fudan.edu.cn>
Date:   Tue Sep 25 14:23:12 2018 +0800

    First commit

```

修改 hello.c 后再次提交

```

1#include<stdio.h>
2int main()
3{
4    printf("Hello,world!\n");
5    printf("This is line 2\n");
6    return 0;
7}

```

英 C . ʹ ʹ

```
[~/git_test]
moocos-> git add hello.c
[~/git_test]
moocos-> git commit -m "Second commit"
[master 09abc6b] Second commit
1 file changed, 1 insertion(+)
```

git log 查看提交记录

```
commit 09abc6ba70bfac863e169d8a6c2ed5d90bce5b9b
Author: Jiawei Qi <16307130293@fudan.edu.cn>
Date: Tue Sep 25 14:53:30 2018 +0800

    Second commit

commit 424f6fb936f849a845bb361ea4dfdf7cc6bbfde5
Author: Jiawei Qi <16307130293@fudan.edu.cn>
Date: Tue Sep 25 14:23:12 2018 +0800

    First commit
```

使用 git show [commit number]命令可以查看提交的具体内容（以 First commit 为例）

```
[~/git_test]
moocos-> git show 424f6fb936f849a845bb361ea4dfdf7cc6bbfde5
```

```
1 commit 424f6fb936f849a845bb361ea4dfdf7cc6bbfde5
2 Author: Jiawei Qi <16307130293@fudan.edu.cn>
3 Date: Tue Sep 25 14:23:12 2018 +0800
4
5     First commit
6
7 diff --git a/hello.c b/hello.c
8 new file mode 100644
9 index 0000000..8da6da3
10 --- /dev/null
11 +++ b/hello.c
12 @@ -0,0 +1,6 @@
13 +#include<stdio.h>
14 +int main()
15 +{
16 +     printf("Hello,world!\n");
17 +     return 0;
18 +}
```

(END)

使用 git diff 可以查看某两次提交之间的差异

```
[~/git_test]
moocos-> git diff 424f6fb936f849a845bb361ea4dfdf7cc6bbfde5 09abc6ba70bfac863e169d8a6c2ed5d90bce5b9b
```

```

1 diff --git a/hello.c b/hello.c
2 index 8da6da3..c8ed16d 100644
3 --- a/hello.c
4 +++ b/hello.c
5 @@ -2,5 +2,6 @@
6  int main()
7  {
8      printf("Hello,world!\n");
9  +   printf("This is line 2\n");
10     return 0;
11 }
(END)

```

还有一个很常见的 git 指令是 git checkout，回到之前的版本后提交，可以看到 hello.c 文件已经恢复之前的版本

```

[~/git_test]
moocos-> git checkout 424f6fb936f849a845bb361ea4dfdf7cc6bbfde5 hello.c
[~/git_test]
moocos-> git commit -m "reverse to previous version"
[master 6b01731] reverse to previous version
1 file changed, 1 deletion(-)
[~/git_test]
moocos-> git log

```

```

commit 6b017317158c8266dab658aadf18f851917f3b53
Author: Jiawei Qi <16307130293@fudan.edu.cn>
Date: Tue Sep 25 15:24:13 2018 +0800

    reverse to previous version

commit 09abc6ba70bfac863e169d8a6c2ed5d90bce5b9b
Author: Jiawei Qi <16307130293@fudan.edu.cn>
Date: Tue Sep 25 14:53:30 2018 +0800

    Second commit

commit 424f6fb936f849a845bb361ea4dfdf7cc6bbfde5
Author: Jiawei Qi <16307130293@fudan.edu.cn>
Date: Tue Sep 25 14:23:12 2018 +0800

    First commit

```

```

1 #include<stdio.h>
2 int main()
3 {
4     printf("Hello,world!\n");
5     return 0;
6 }

```



### 3) 源码编辑工具

- vim

上学期学 ICS 的时候着实被 vim 恶心了一把，花两个小时学习了一些诡异的操作，例如 i 表示开始 input，每次转换操作都需要 Esc 等等。但鉴于整个学期都用 vim 做的 PA，所以慢慢习惯了，也有点儿开始喜欢上这个“编译器之神”了

<回顾常用命令>

- 【i】进入插入模式（从光标所在处）
- 【ESC】退出编辑模式回到一般模式
- 【:wq】保存后离开
- 【h/j/k/l】光标左下上右移动一个字符
- 【(】光标移动到当前行的最前面
- 【)】光标移动到当前行的末尾
- 【x/X】Delete/Backspace
- 【dd】删除光标所在行，前面加 n 表示行数
- 【yy】复制光标所在行，前面加 n 表示行数
- 【p/P】在光标下一行粘贴/上一行粘贴
- 【/word】在文件中查找内容为 word 的字符串（向下查找）
- 【n】查找下一个匹配
- 【u】撤销上一个操作
- 【.] 重复上一个操作

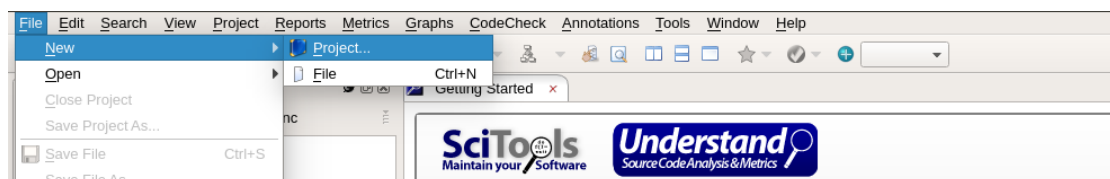
重新进行了安装，并在.vimrc 中进行了配置（开启高亮设置、字符输入法等）命令行里可以直接打开，右键文件也可以选择用安装好的 gvim 打开

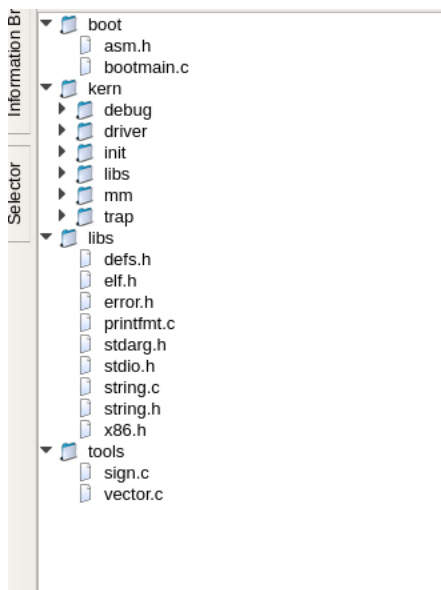
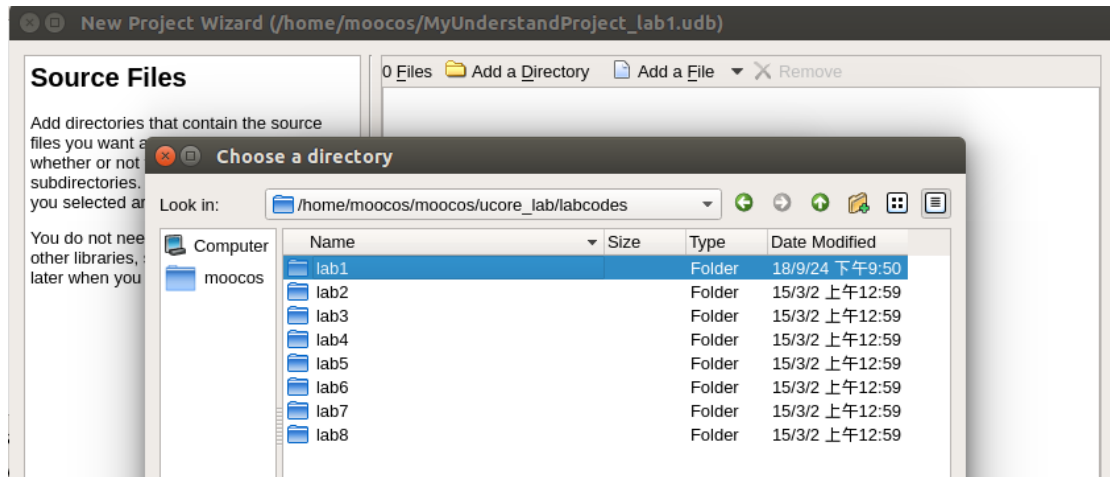
```
[~]  
moocos-> sudo apt-get install vim-gtk  
[sudo] password for moocos:  
Reading package lists... Done  
Building dependency tree  
Reading state information... Done  
The following packages were automatically installed and are no longer required:  
  gyp javascript-common libc-ares-dev libc-ares2 libjs-node-uuid libssl-dev  
  libssl-doc libv8-3.14-dev libv8-3.14.5  
Use 'apt-get autoremove' to remove them.  
The following extra packages will be installed:
```

- understand

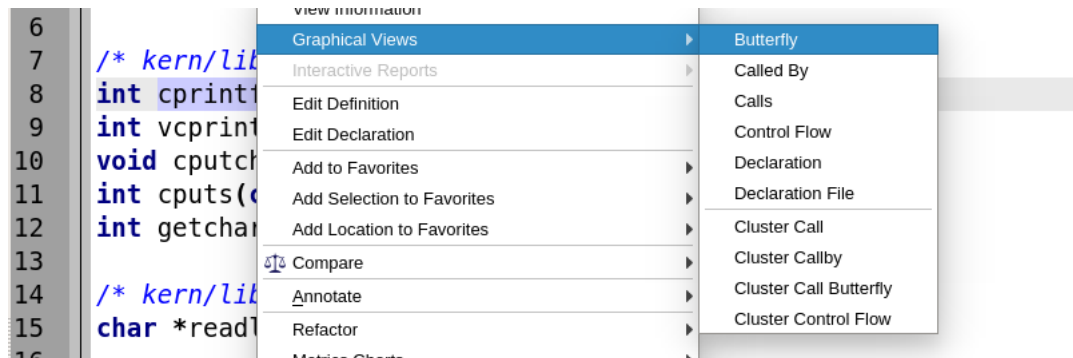
understand 是支持多语言的、可视化的理解和分析代码的工具我们以 lab1 为例练习使用 understand.

首先新建一个工程，将 lab1 导入

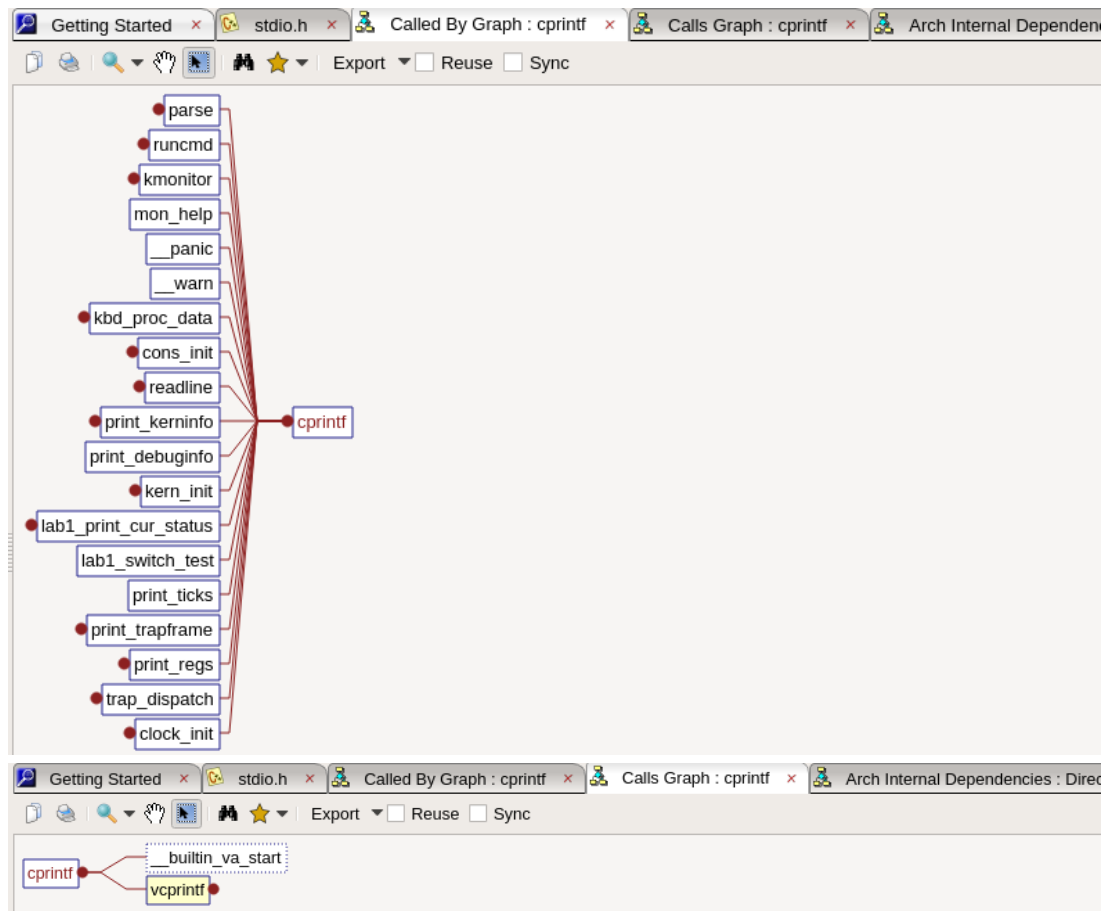




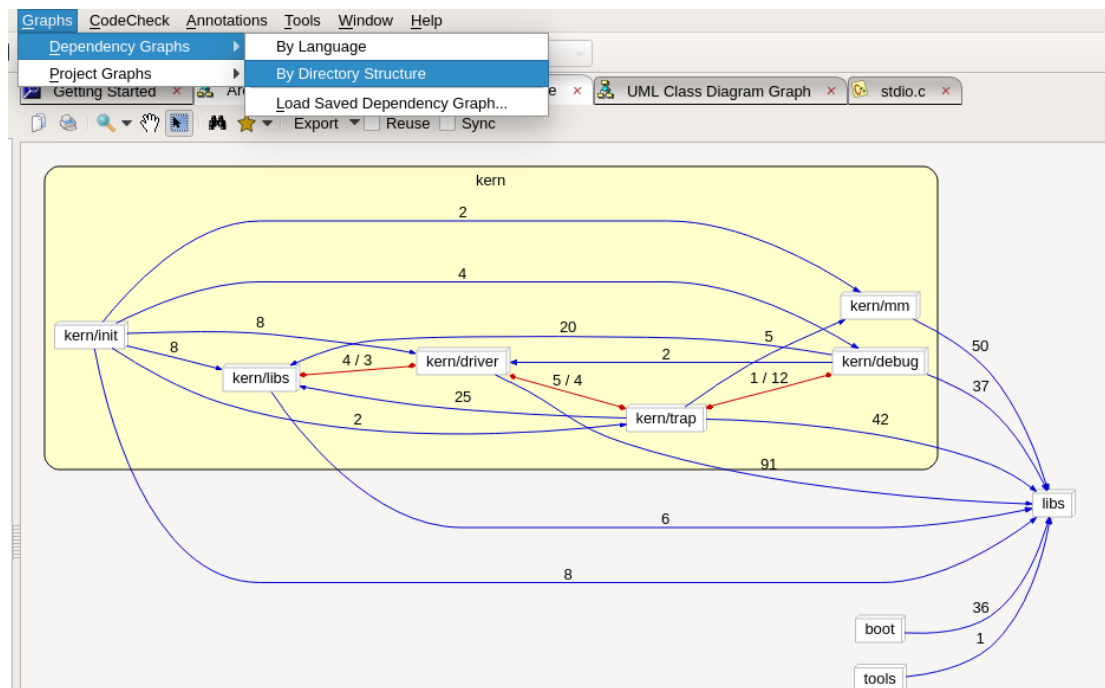
任意打开一个文件，例如 libs/stdio.h，选中 cprintf 这一函数名可以查看其具体信息（例如返回值类型等），右键可选择生成分析图表



例如 called by 可显示该函数被哪些函数调用；calls 则是调用了哪些函数。点击即可进入相应函数



上面菜单栏 Graphs 可以选择生成依赖关系图



还有其它功能例如界面右上角可以 search 函数、文件等；可以生成文件包含关系图，还有表格的形式

- 其它：gedit 为 ubuntu 自带，简单易用；emacs 与 vim 同为经典编辑器，在本学期实验中计划选择较为熟练的 vim

#### 4) 源码比较工具

- diff&patch  
diff 用于列出两个文件的不同之处，patch 用于根据补丁文件修改源文件  
例如有 file1 和 file2 两个文件只在第二行有不同：

```
[~/lab0_test]
moocos-> cat file1
this is line1
this is line2
this is line3
[~/lab0_test]
moocos-> cat file2
this is line1
this is line2 hi
this is line3
```

使用 diff 命令进行比较，并生成补丁文件 patch.log

```
[~/lab0_test]
moocos-> diff file1 file2 > patch.log
[~/lab0_test]
moocos-> cat patch.log
2c2
< this is line2
---
> this is line2 hi
```

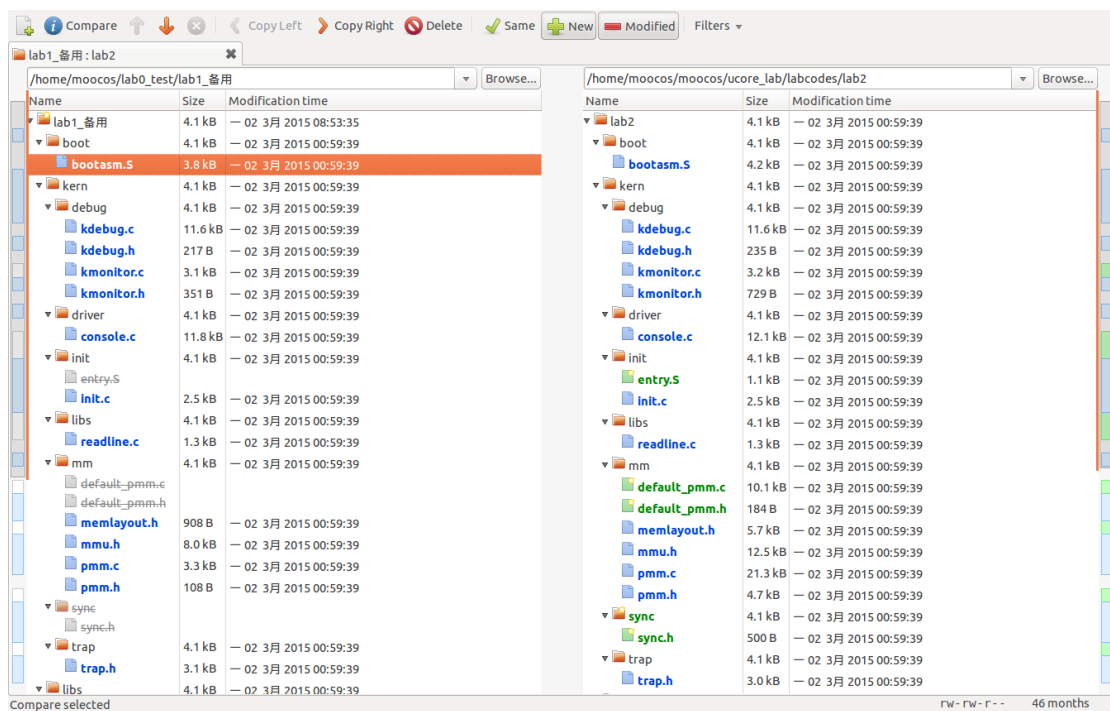
使用 patch 命令可以给 file1 打补丁，变为与 file2 相同

```
[~/lab0_test]
moocos-> patch file1 patch.log
patching file file1
[~/lab0_test]
moocos-> cat file1
this is line1
this is line2 hi
this is line3
[~/lab0_test]
```

patch -R 用于反向补丁，使 file1 恢复原貌

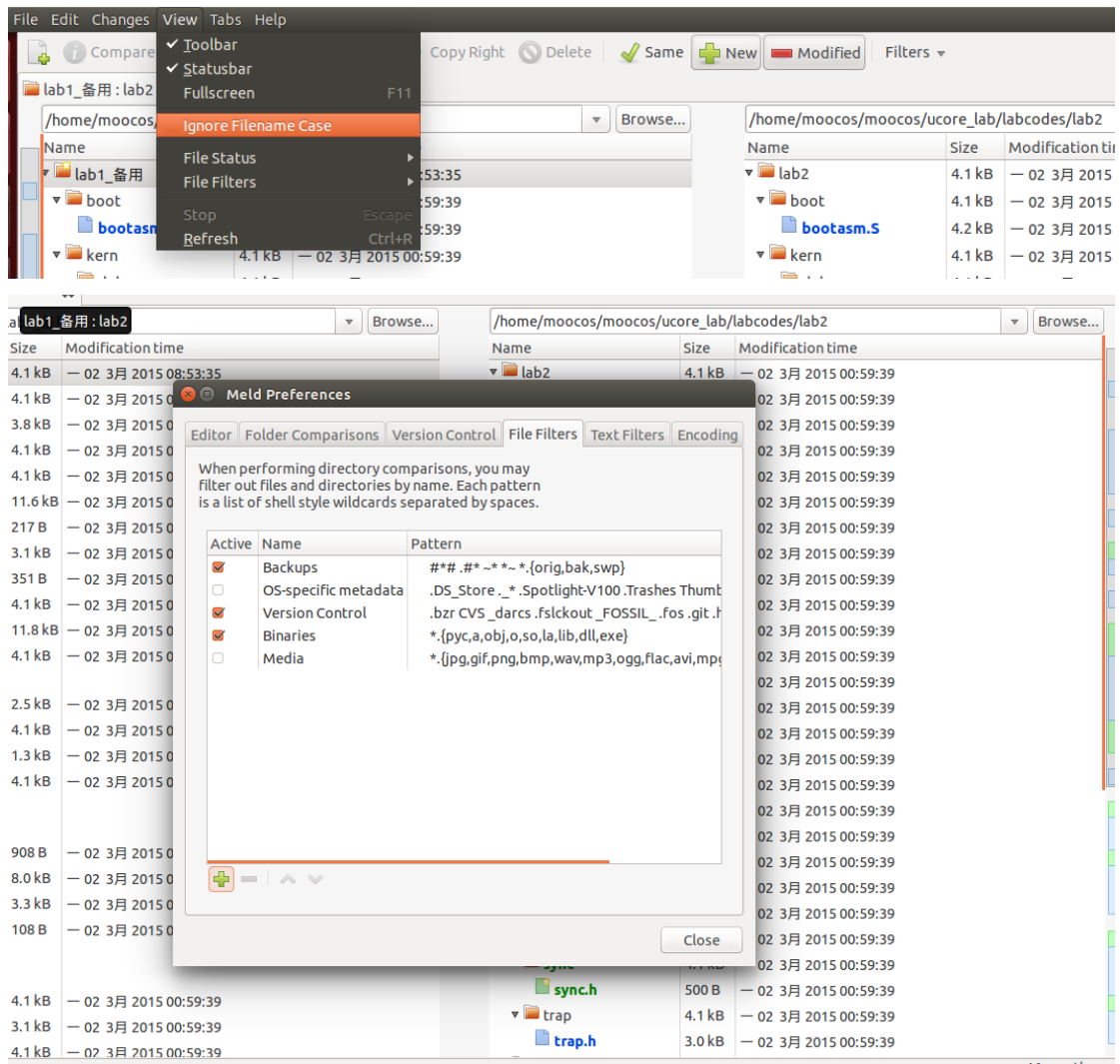
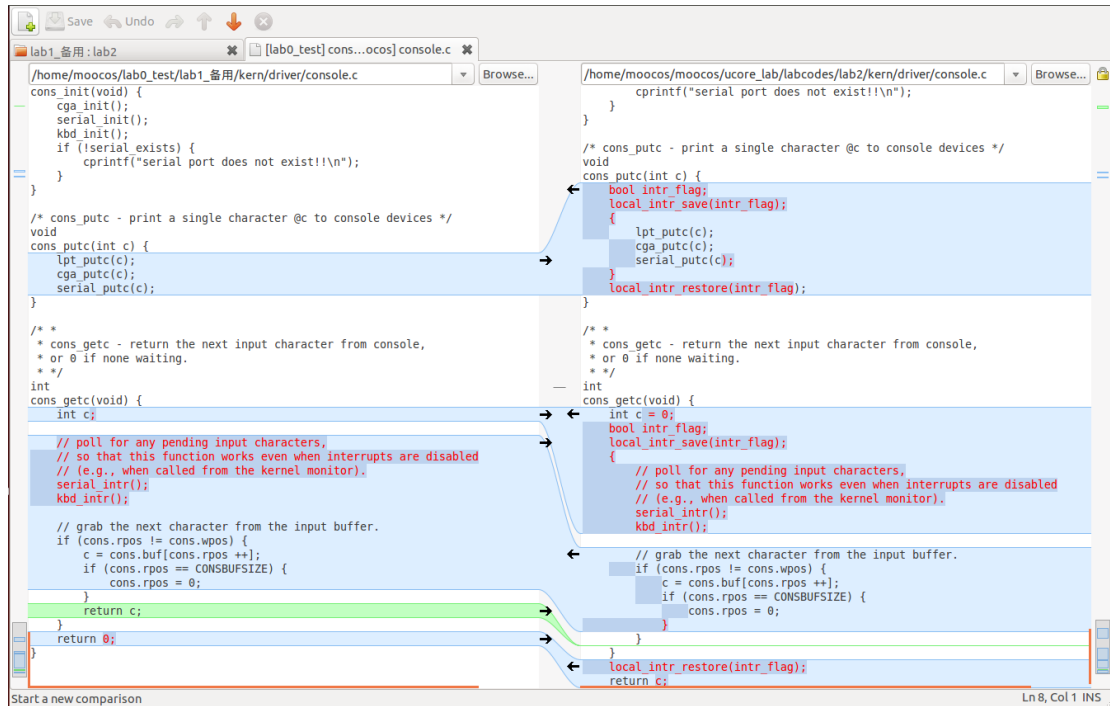
```
[~/lab0_test]
moocos-> patch -R file1 patch.log
patching file file1
[~/lab0_test]
moocos-> cat file1
this is line1
this is line2
this is line3
```

- meld  
图形化比较文件/文件夹工具，相比 diff 更加直观，操作方便  
例如对比 ucore 中 lab1 和 lab2 两个文件夹：



左右分栏比较。其中正常字体表示相同部分，蓝色粗体表示不同的文件，灰色被划去表示相对绿色粗体缺少的文件。上方工具栏中绿色加号表示打开新的比较；Compare 可以打开具体两个不同的文件比较；上下箭头可以选中不同的 change，copy 可以使得一方与另一方相同；默认选中 Same 显示出相同的文件；Modified 展开文件夹。右边不同颜色标志不同区域，可以通过点击直接到达。

可以选择新建过滤条目；也可通过“Ignore Filename Case”忽视大小写



## 5) 开发编译调试工具

- gcc

以最熟悉的 hello 程序为例

```
1 #include<stdio.h>
2 int main()
3 {
4     printf("Hello,world!\n");
5     return 0;
6 }
```

两种方式生成可执行文件 ./hello 可以执行文件，打印 Hello,world!

- ① 直接通过 gcc hello.c -o hello 命令，实现预处理、编译、汇编并链接生成

```
[~/lab0_test]
moocos-> gcc hello.c -o hello1
[~/lab0_test]
moocos-> ./hello1
Hello,world!
```

- ② 四步分开进行

预处理: gcc -E hello.c -o hello.i

编译: gcc -S hello.i -o hello.s

汇编: gcc -c hello.s -o hello.o

链接: gcc -o hello hello.o

```
[~/lab0_test]
moocos-> gcc -E hello.c -o hello.i
[~/lab0_test]
moocos-> gcc -S hello.i -o hello.s
[~/lab0_test]
moocos-> gcc -c hello.s -o hello.o
[~/lab0_test]
moocos-> gcc -o hello hello.o
[~/lab0_test]
moocos-> ./hello
Hello,world!
```

-wall 可用于显示警告信息

-g 用于后面的 gdb 调试

objdump -d hello.o/hello 可以反汇编出汇编指令代码

- make

make 是间接调用 gcc 来简化编译过程的命令工具，当执行 make 时，会在当前的目录下搜寻 Makefile/makefile 文本文档，执行相应的操作。

编译输出 Hello world! 需要三个文件

- 1) print.h

```
1 #include<stdio.h>
2 void printhello();
```

- 2) print.c

```
1 #include "print.h"
2 void printhello()
3 {
4     printf("Hello,world\n");
5 }
```

### 3) main.c

```
1 #include "print.h"
2 int main()
3 {
4     printhello();
5     return 0;
6 }
```

### 4) Makefile

Makefile 文档定义了一系列规则指定了文件的编译顺序，由三部分（目标、依赖、命令）组成。执行 make 命令会去找当前目录下 Makefile 的第一个目标，本例中为 helloworld，根据依赖文件来执行命令。发现依赖文件 main.o 和 print.o 找不到，则会向下查找看是否在另一个文件中去生成依赖文件，后两条规则说明了 main.o 和 print.o 的来源。

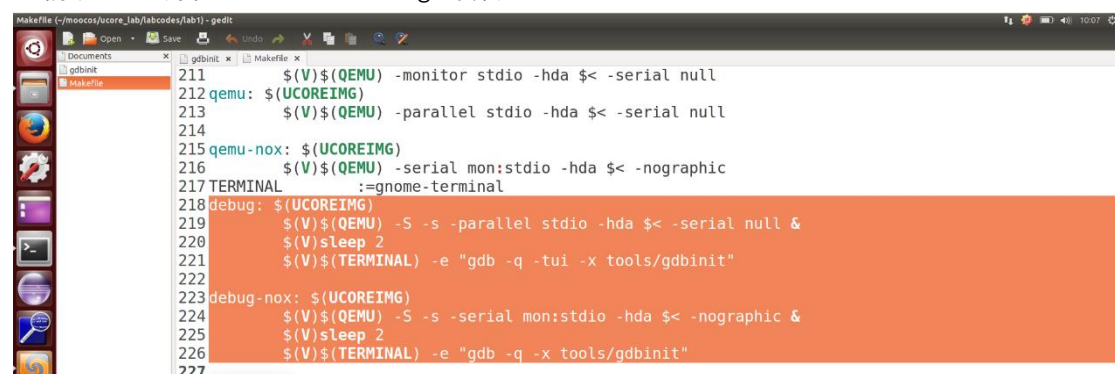
当执行 make clean 时候会将生成的两个目标文件删除

```
1 helloworld: main.o print.o
2     gcc -o helloworld main.o print.o
3 main.o: main.c print.h
4     gcc -c main.c
5 print.o: print.c print.h
6     gcc -c print.c
7
8 clean:
9     rm helloworld main.o print.o
```

使用 make 命令编译文件，make clean 清除目标文件

```
[~/lab0 test]
moocos-> make helloworld
gcc -c main.c
gcc -c print.c
gcc -o helloworld main.o print.o
[~/lab0 test]
moocos-> ./helloworld
Hello,world
[~/lab0 test]
moocos-> make clean
rm helloworld main.o print.o
```

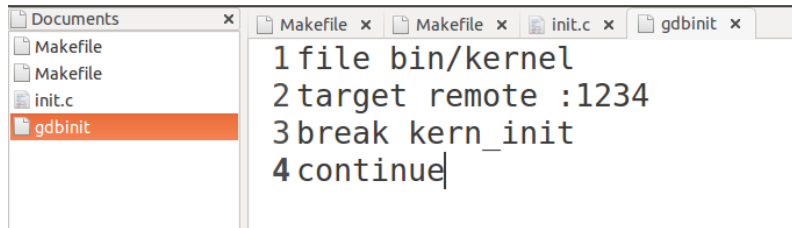
我们以 lab1 为例查看 make debug 的作用



```
Makefile (/moocos/ucore_lab1/codes/lab1) - gedit
211 $(V)$(QEMU) -monitor stdio -hda $< -serial null
212 qemu: $(UCOREIMG)
213 $(V)$(QEMU) -parallel stdio -hda $< -serial null
214
215 qemu-nox: $(UCOREIMG)
216 $(V)$(QEMU) -serial mon:stdio -hda $< -nographic
217 TERMINAL :=gnome-terminal
218 debug: $(UCOREIMG)
219 $(V)$(QEMU) -S -s -parallel stdio -hda $< -serial null &
220 $(V)sleep 2
221 $(V)$(TERMINAL) -e "gdb -q -tui -x tools/gdbinit"
222
223 debug-nox: $(UCOREIMG)
224 $(V)$(QEMU) -S -s -serial mon:stdio -hda $< -nographic &
225 $(V)sleep 2
226 $(V)$(TERMINAL) -e "gdb -q -x tools/gdbinit"
227
```

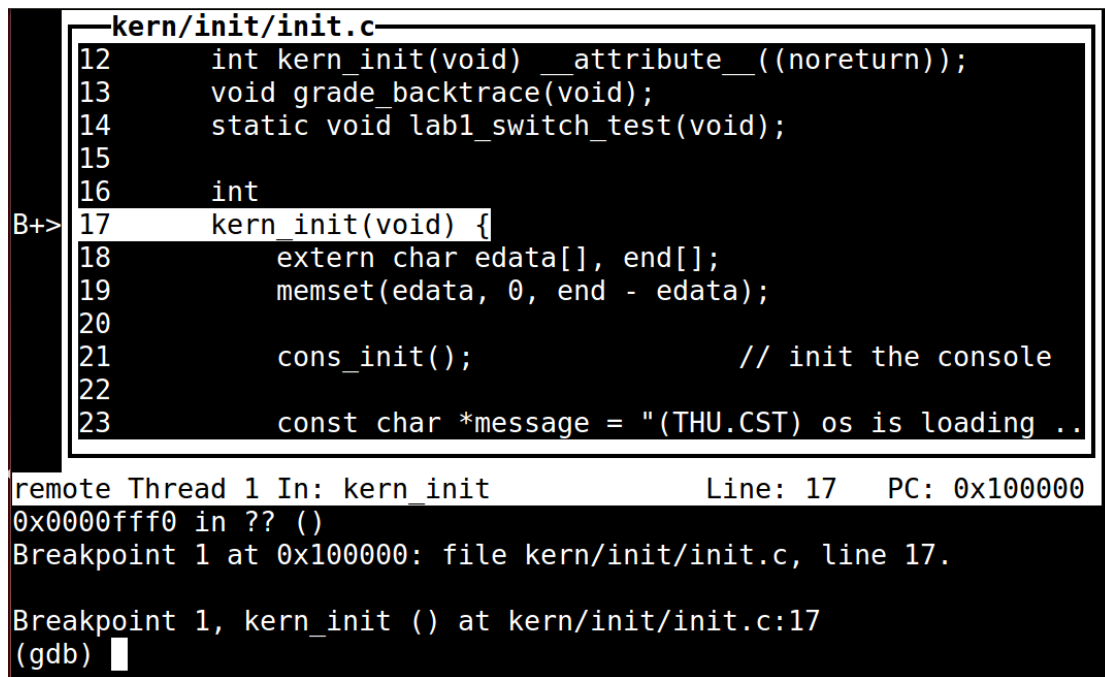
首先是对 qemu 进行操作，sleep 2 等待一段时间，然后对 tools/gdbinit 文件进行调试，





可以看到是在 kern\_init 处添加断点，然后 continue

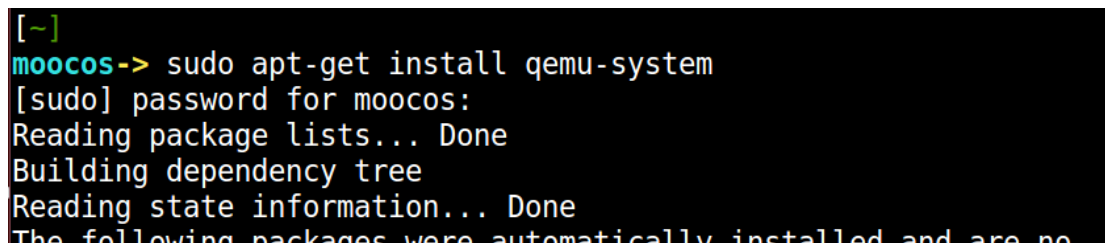
执行 make debug 以后会出现一个新的终端，分为上下两个窗口，上面窗口显示运行到的源码，下面的窗口是 debug 调试界面，程序断在第 17 行，kern\_init() 函数



- gdb  
见下文结合 gdb 和 qemu 源码级调试 ucore

## 6) 硬件模拟器 qemu

首先使用 apt-get 安装 ubuntu 中提供的 qemu



结合 gdb 和 qemu 源码级调试 ucore，因为 ucore 代码中已经有了 Makefile，直接使用 make 命令即可

```
[~/moocos/ucore_lab/labcodes/lab1]
moocos-> make
+ cc kern/init/init.c
kern/init/init.c:95:1: warning: 'lab1_switch_test' defined but
not used [-Wunused-function]
lab1_switch_test(void) {
^
+ cc kern/libs/readline.c
+ cc kern/libs/stdio.c
+ cc kern/debug/kdebug.c
kern/debug/kdebug.c:251:1: warning: 'read_eip' defined but not
```

在 bin 目录下生成一系列目标文件

```
moocos-> ls bin
bootblock kernel sign ucore.img
```

使用 meld 工具进行前后对比，主要区别在于 make 以后多出 bin 和 obj 两个文件夹

Name	Size	Modification time	Name	Size	Modification time
lab1_备用	4.1 kB	— 02 3月 2015 08:53:35	lab1	4.1 kB	— 24 9月 2018 16:21:42
bin			bin	4.1 kB	— 24 9月 2018 16:21:42
boot	4.1 kB	— 02 3月 2015 00:59:39	boot	4.1 kB	— 02 3月 2015 00:59:39
kern	4.1 kB	— 02 3月 2015 00:59:39	kern	4.1 kB	— 02 3月 2015 00:59:39
libs	4.1 kB	— 02 3月 2015 00:59:39	libs	4.1 kB	— 02 3月 2015 00:59:39
obj			obj	4.1 kB	— 24 9月 2018 16:21:42
boot			boot	4.1 kB	— 24 9月 2018 16:21:42
kern			kern	4.1 kB	— 24 9月 2018 16:21:41
debug			debug	4.1 kB	— 24 9月 2018 16:21:42
driver			driver	4.1 kB	— 24 9月 2018 16:21:42
mm			init	4.1 kB	— 24 9月 2018 16:21:41
trap			libs	4.1 kB	— 24 9月 2018 16:21:41
libs			mm	4.1 kB	— 24 9月 2018 16:21:42
sign			trap	4.1 kB	— 24 9月 2018 16:21:42
tools	4.1 kB	— 02 3月 2015 00:59:39	libs	4.1 kB	— 24 9月 2018 16:21:42
			sign	4.1 kB	— 24 9月 2018 16:21:41
			tools	4.1 kB	— 24 9月 2018 16:21:42
			tools	4.1 kB	— 02 3月 2015 00:59:39

使用远程调试

1) 首先使用 -S -s 两个参数启动 qemu，等待 gdb 启动

```
[~/moocos/ucore_lab/labcodes/lab1]
moocos-> qemu -S -s -hda ./bin/ucore.img -monitor stdio
QEMU 2.0.0 monitor - type 'help' for more information
(qemu) c
```

2) 然后另外调出一个命令行窗口，在 gdb 命令行界面下，使用 target remote 连接到 qemu

3) qemu 中输入 c 会继续执行下去，在 gdb 中需要使用 file 命令载入文件中符号信息

4) 设置断点并执行

5) qemu 单步调试

```
[~/moocos/ucore_lab/labcodes/lab1]
moocos-> gdb ./bin/kernel

(gdb) target remote:1234
Remote debugging using :1234
0x00000fff0 in ?? ()
(gdb) file ./bin/kernel
A program is being debugged already.
Are you sure you want to change the file? (y or n) y

Load new symbol table from "./bin/kernel"? (y or n) y
Reading symbols from ./bin/kernel...done.
(gdb) break kern_init
Breakpoint 1 at 0x100006: file kern/init/init.c, line 19.
(gdb) run
The "remote" target does not support "run". Try "help target" or "continue".
(gdb) c
Continuing.

Breakpoint 1, kern_init () at kern/init/init.c:19
19      memset(edata, 0, end - edata);
(gdb) █
```

```
QEMU 2.0.0
(qemu) c
(qemu) █

SeaBIOS (version 1.7.4-20140219_122725-roseapple)

iPXE (http://ipxe.org) 00:03.0 C900 PCI2.10 PnP PMM+07FC1110+07F21110 C900

Booting from Hard Disk...
(THU.CST) os is loading ...

Special kernel symbols:
  entry 0x00100000 (phys)
  etext 0x00103209 (phys)
  edata 0x0010da16 (phys)
  end    0x0010ed20 (phys)
Kernel executable memory footprint: 60KB
++ setup timer interrupts
-
```

```
QEMU [Stopped]
SeaBIOS (version 1.7.4-20140219_122725-roseapple)

iPXE (http://ipxe.org) 00:03.0 C900 PCI2.10 PnP PMM+07FC1110+07F21110 C900

Booting from Hard Disk...
```

为避免每次调试都需要输入较多东西，可以将上述命令存在脚本中，让 gdb 在启动的时候自动载入。代码框架已经设置好，即前面分析 makefile 时 make debug 的作用。使用该直观界面练习常用的 gdb 命令

**l(list)** 可以显示指定行数的源代码

**info breakpoints** 查看所有断点信息

**break** 设置断点（地址可以是行号、函数名或其它）

**delete breakpoints** 删除断点

```
(gdb) info break
Num      Type           Disp Enb Address      What
1        breakpoint     keep y   0x00100000 in kern_init at kern/init/init.c:17
          breakpoint already hit 1 time
```

```
(gdb) break 26
Breakpoint 2 at 0x10004b: file kern/init/init.c, line 26.
(gdb) info break
Num      Type           Disp Enb Address      What
1        breakpoint     keep y   0x00100000 in kern_init at kern/init/init.c:17
          breakpoint already hit 1 time
2        breakpoint     keep y   0x0010004b in kern_init at kern/init/init.c:26
```

**c(continue)**继续程序的运行，直到遇到下一个断点

```
(gdb) c
Continuing.

Breakpoint 2, kern_init () at kern/init/init.c:26
```

```
(gdb) d break 2
(gdb) info break
Num      Type           Disp Enb Address      What
1        breakpoint     keep y   0x00100000 in kern_init at kern/init/init.c:17
          breakpoint already hit 1 time
```

```
(gdb) c
Continuing.

Breakpoint 1, kern_init () at kern/init/init.c:17
```

**n(next)**执行下一条语句，不会进入函数调用

```
—kern/init/init.c—
12  int kern_init(void) __attribute__((noreturn));
13  void grade_backtrace(void);
14  static void lab1_switch_test(void);
15
16  int
17  kern_init(void) {
18      extern char edata[], end[];
19      memset(edata, 0, end - edata);
20
21      cons_init();                // init the console
22
23      const char *message = "(THU.CST) os is loading ...";
24      cprintf("%s\n\n", message);
```

```
—kern/init/init.c—
12  int kern_init(void) __attribute__((noreturn));
13  void grade_backtrace(void);
14  static void lab1_switch_test(void);
15
16  int
17  kern_init(void) {
18      extern char edata[], end[];
19      memset(edata, 0, end - edata);
20
21      cons_init();                // init the console
22
23      const char *message = "(THU.CST) os is loading ...";
24      cprintf("%s\n\n", message);
```

s(step) 执行下一条语句, 会进入函数调用

```
kern/driver/console.c
410     }
411
412     /* cons_init - initializes the console devices */
413     void
414     cons_init(void) {
415         cga_init();
416         serial_init();
417         kbd_init();
418         if (!serial_exists) {
419             cprintf("serial port does not exist!!\n");
420         }
421     }
422
```

finish 退出当前函数调用

```
kern/init/init.c
18     extern char edata[], end[];
19     memset(edata, 0, end - edata);
20
21     cons_init();           // init the console
22
23     const char *message = "(THU.CST) os is loading ...";
24     cprintf("%s\n\n", message);
25
26     print_kerninfo();
27
28     grade_backtrace();
29
30     pmm_init();           // init physical memory management

```

p(print)打印变量的值

```
kern/init/init.c
18     extern char edata[], end[];
19     memset(edata, 0, end - edata);
20
21     cons_init();           // init the console
22
23     const char *message = "(THU.CST) os is loading ...";
24     cprintf("%s\n\n", message);
25
26     print_kerninfo();
27
28     grade_backtrace();
29
30     pmm_init();           // init physical memory management

```

```
(gdb) finish
Run till exit from #0  cons_init () at kern/driver/console.c:416
kern_init () at kern/init/init.c:23
(gdb) n
(gdb) print message
$1 = 0x103220 "(THU.CST) os is loading ..."
```

q(quit)退出 gdb 环境

### 三、实验感悟

一开始看到 lab0 的手册竟然有 60 多页… (不过竟然是中文的) 一边回想起上学期被 ICS 的 PA 支配的恐惧, 一遍又庆幸已经接触过并且比较熟悉 Linux, vim, gdb 这些了, 所以说 No pains, no gains 诚不我欺啊

看完慕课 lab0 的视频觉得 OS lab 好难, 大概是出于未知吧。然后还是从最基础的 lab0 开始! 下载、安装、练习熟悉后面实验要用到的工具, 回顾了 shell 命令行、apt、vim、gcc、gdb、make, 学习了 git、understand、diff、meld…

比较大的收获就是重新熟悉了环境, 从已经掌握的工具中也学到新的东西 (比如阅读理解 Makefile), 新工具有得很简单易懂 (比如 meld), 有的就有点儿复杂 (比如 git) 为了记录就在实验过程中把这些操作都截了图

对于后面的实验可以说是畏惧中有期待吧