

操作系统实验

lab0 初识操作系统

内容提要

- 背景知识
- 实验概述
- 实验内容
 - 掌握实验相关工具的使用
 - 掌握git的管理模式
 - 实验完成方式
- 实验实战

背景知识

- 工欲善其事必先利其器
- 操作系统课程设计要求同学们大量使用Linux操作系统的命令行交互界面，这意味着我们需要掌握一些基础操作命令，同时需要硬件模拟器GXemul模拟mips运行环境，最终由Git进行版本控制和评测。
- 课程服务器会为同学们提供成熟的实验环境，但考虑到很多同学对于没有图形化的操作系统界面几乎没有接触，故本章会着重为同学们进行实验环境的介绍，旨在“扫盲”
- 熟练掌握本章内容会让同学们后续实验如鱼得水

实验概述

- 熟悉实验环境配置：
- 连接工具：通过CG平台远程连接到我们的实验环境
- 操作系统：Linux 虚拟机Ubuntu，需掌握常用命令
- 硬件模拟器：GXemul，本章暂不需要了解
- 编译器：GCC，需掌握命令格式及常用选项
- 版本控制：Git，需掌握如何提交本地变更，并同步到远端库以触发评测脚本；了解版本回退等实用功能
- 进程与fork
- Makefile
- 掌握相关技巧后，完成课下测试任务，提交评测

实验内容—CG平台

- 具体实验所需环境已经集成到CG平台上(<http://course.educg.net>), 每位同学都有自己的账号和密码, 登陆即可。
- 点击在线实验, 选中对应的lab即可进入CG平台开始实验。

实验内容—CG平台

操作系统（北京航空航天大学）

欢迎你, 202006105! [\[个人信息\]](#) [\[注销\]](#)

[首页](#) | [课程信息](#) | [在线作业](#) | [在线考试](#) | [在线实验](#) | [在线答疑](#) | [成绩查询](#) | [操作系统内核实验功能演示](#)

切换课程: 操作系统（北京航空航天大学） ▾



lab0 实验环境介绍

在本实验中, 我们需要去了解实验环境, 熟悉Linux 操作系统 (Ubuntu), 了解控制终端, 掌握一些常用工具并能够脱离可视化界面进行工作。

进入实验环境后, 使用以下命令查看git服务器地址:

```
cat .cgconfig
```

假如, git地址是192.168.100.20, 使用以下命令克隆实验的代码库:

```
git clone git@192.168.100.20:$CGUSERID-lab
```

实验代码库将被check out到本地, 然后按照实验指导手册 (双击guide-book可打开手册) 开始实验。

开始实验

操作系统内核实验作业-TEST

🕒 开始时间: 2020-02-14 20:00:00

🕒 截止时间: 2020-02-28 23:00:00

操作系统内核实验作业-TEST
共7个实验

实验1: lab0 实验环境介绍

实验2: lab1 内核、Boot和printf

实验3: lab2 内存管理

实验4: lab3 进程与异常

实验5: lab4 系统调用与fork

实验6: lab5 文件系统

实验7: lab6 管道与Shell

实验步骤

✓ 1. lab0 实验环境介绍

在线时长: 00:35:30

开始实验

实验内容—获取lab0环境

- `cat .cgconfig`
- `cd work` (这一步是为了后续将实验环境复制到work中)
- `git clone git@192.168.100.21:\$CGUSERID-lab`
(192.168.100.21是第一部显示的结果)
- `cd xxxxxxx` (xxxxxxx为当前路径下唯一文件夹)
- `git branch -a` (查看远程仓库分支)
- `git checkout lab0` (此时成功将lab0环境拷贝到当前目录,即可进行实验)

实验内容一-获取lab0环境

```
jovyan@ce6eed2d20ef:~$ cat .cgconfig
192.168.100.21
jovyan@ce6eed2d20ef:~$ cd work/
jovyan@ce6eed2d20ef:~/work$ git clone git@192.168.100.21:$CGUSERID-lab
Cloning into 'J5QUJQBYMVDVU-lab'...
remote: Counting objects: 10, done.
remote: Compressing objects: 100% (6/6), done.
remote: Total 10 (delta 0), reused 0 (delta 0)
Receiving objects: 100% (10/10), done.
warning: remote HEAD refers to nonexistent ref, unable to checkout.

jovyan@ce6eed2d20ef:~/work$ cd J5QUJQBYMVDVU-lab/
jovyan@ce6eed2d20ef:~/work/J5QUJQBYMVDVU-lab$ git branch -a
remotes/origin/lab0
jovyan@ce6eed2d20ef:~/work/J5QUJQBYMVDVU-lab$ git checkout lab0
Branch 'lab0' set up to track remote branch 'lab0' from 'origin'.
Switched to a new branch 'lab0'
```


实验内容--基础操作命令

- ls: 显示当前目录文件
- mkdir: 建立目录文件
- rmdir: 删除空目录
- rm: 删除文件
- cd: 变更工作目录
- cat: 连接并输出文件
- cp: 复制文件
- mv: 移动文件
- source: 运行可执行文件
- find: 查找文件
- grep: 文本搜索, 可跨文件
- man: 帮助手册

实验内容一实用工具

```
1 =====
2 这是一个使用Vim编辑器打开的文本文件
3 刚打开Vim时会默认进入命令模式
4 在命令模式中，可以使用Vim的操作命令对文本进行操作
5 只有在插入模式中，才可以键入字符
6 下面列举一些Vim的常用基本操作：
7     (在命令模式下)
8     i      切换为插入模式
9     Esc    返回普通模式
10    o      在当前行之下插入
11    O      在当前行之上插入
12    u      撤销(undo)
13    Ctrl+r  重做(redo)
14    yy     复制一行
15    dd     剪切一行
16    y      复制(按y后按方向键左/右,复制光标左/右边的字符)
17    d      剪切(按d后按方向键左/右,剪切光标左/右边的字符)
18    2yy    复制下面2行
19    3dd    剪切下面3行
20    4y     复制4个字符(按方向键左/右开始复制光标左/右的字符)
21    5d     剪切5个字符(按方向键左/右开始剪切光标左/右的字符)
22    p      在当前位置之后粘贴
23    P      在当前位置之前粘贴
24    :q     不保存直接退出
25    :q!    强制不保存退出
26    :w     保存
27    :wq    保存后退出
28    :N     将光标移至第N行
29    :set nu 显示行号
30    /word  查询word在文中出现的位置，若有多个，按n/N分别移至下/上一个
31 =====
```

31,80 全部

实验内容一实用工具2

快
原文

1 语法: gcc [选项]... [参数]...

2 选项 (常用):

3 -o	指定生成的输出文件
4 -S	将 C 代码转换为汇编代码
5 -Wall	显示警告信息
6 -c	仅执行编译操作, 不进行链接操作
7 -M	列出依赖

8 参数:

9 C 源文件: 指定 C 语言源代码文件

10 e.g.

11 \$ gcc test.c

12 # 默认生成名为 a.out 的可执行文件

13 #Windows 平台为 a.exe

14 \$ gcc test.c -o test

15 # 使用 -o 选项生成名为 test 的可执行文件

16 #Windows 平台为 test.exe

实验内容--Git介绍

- 最原始的版本控制是纯手工的版本控制：修改文件，保存文件副本。任意，时间长了使知道新旧，上一版作了什是下面的样子

毕业论文.docx
毕业论文改.docx
毕业论文改1.docx
毕业论文改2.docx
毕业论文完成版.docx
毕业论文完成版1.docx
毕业论文最终版.docx
毕业论文最终版1.docx
毕业论文最最终版.docx
毕业论文最最终版1.docx
毕业论文最最终绝对不改版.docx
毕业论文最最终绝对不改版1.docx
毕业论文最最终绝对不改版2.docx
大五重修申请.docx
遗书.docx

修改文件，保
本时命名比较随
个是老的了，即
什么内容，相对
去后，很可能就

- 而Git则是一个撤销改动。2.简洁的指令与操
作。3.可以像
回去，还能在
次改动，只需

改动，可以轻松
简洁的指令与操
且不但能穿越
如果想查看某

实验内容--Git介绍

- `git init`: 初始化当前目录为Git工作区
- `git add`: 追踪文件变更
- `git commit`: 提交文件变更
- `git log`: 查看提交记录
- `git status`: 查看文件状态
- `git reset`: 版本回退
- `git branch <branch-name>`: 创建新分支
- `git checkout <branch-name>`: 切换分支
- `git push`: 将本地变更推送到远端仓库
- `git pull`: 将远端仓库抓回到本地库

进程与fork

```
■ 1 int main(void) {  
■ 2     int var;  
■ 3     pid_t pid;  
■ 4     printf("Before fork.\n");  
■ 5     pid = fork();  
■ 6     printf("After fork.\n");  
■ 7     if (pid == 0) {  
■ 8         printf("this is son.\n");  
■ 9     } else {  
■ 10    printf("this is father.\n");  
■ 11    }  
■ 12    return 0;  
■ 13 }
```

Makefile

- target: dependencies
 - command 1
 - command 2
 - ...
 - command n
- all: hello_world.c
 - `gcc -o hello_world hello_world.c`
- <http://www.cs.colby.edu/maxwell/courses/tutorials/maketutor>
- <http://www.gnu.org/software/make/manual/make.html#Reading-Makefiles>

实验实战

- 1、在lab0工作区的src目录中，存在一个名为palindrome.c的文件，使用刚刚学过的工具打开palindrome.c，使用c语言实现判断输入整数 n ($1 \leq n \leq 10000$) 是否为回文数的程序。 n 为输入数据，若这个数字为回文数则输出Y，否则输出N。[注意：正读倒读相同的整数叫回文数]
- 2、在src目录下，存在一个未补全的Makefile文件，借助刚刚掌握的Makefile知识，将其补全，以实现通过make指令触发src目录下的palindrome.c文件的编译链接的功能，生成的可执行文件命名为palindrome。

实验实战

- 3、在src/sh_test目录下，有一个file文件和hello_os.sh文件。
hello_os.sh是一个未完成的脚本文档，请同学们借助shell编程的知识，将其补完，以实现通过指令**bash hello_os.sh AAA BBB.c**，在hello_os.sh所处的文件夹新建一个名为BBB.c的文件，其内容为AAA文件的第8、32、128、512、1024行的内容提取。**[注意：对于指令bash hello_os.sh AAA BBB.c，AAA及BBB可为任何合法文件或路径的名称，例如bash hello_os.sh file hello_os.c，若以有hello_os.c文件，则将其原有内容覆盖]**
- 4、补全后的palindrome.c、Makefile、hello_os.sh依次复制到路径
/dst/palindrome.c /dst/Makefile /dst/sh_test/hello_os.sh **[注意：文件名和路径必须与题目要求相同]**



完成Step1~Step4:

要求按照测试1~测试4要求完成后，最终提交的文件树图示如下：

```
— dst
  |— Makefile
  |— palindrome.c
  |— sh_test
    |— hello_os.sh
— src
  |— Makefile
  |— palindrome.c
  |— sh_test
    |— file
    |— hello_os.sh
```

实验实战

- 5、在lab0工作区ray/sh_test1目录中，存在一个名为changefile.sh的文件，将其补完，以实现通过指令bash changefile.sh，可以删除该文件夹内file71~file100共计30个子文件夹，将file41~file70共计30个子文件夹重命名为newfile41~newfile70。[注意：changefile.sh必须提交]



完成Step5:

要求按照测试5要求完成后，最终提交的文件树图示如左(file下标只显示1~12，newfile下标只显示41~55)

```
— file1
— file10
— file11
— file12
— file2
— file3
— file4
— file5
— file6
— file7
— file8
— file9
— newfile41
— newfile42
— newfile43
— newfile44
— newfile45
— newfile46
— newfile47
— newfile48
— newfile49
— newfile50
— newfile51
— newfile52
— newfile53
— newfile54
— newfile55
```

实验实战

- 6、在lab0工作区的ray/sh_test2目录下，存在一个未补全的search.sh文件，将其补完，以实现通过指令**bash search.sh file int result**，可以在当前文件夹下生成result文件，内容为file文件含有int文本所在的行数。**[注意：对于指令bash search.sh file int result, file及int及result可为任何合法文件或路径的名称，若已有result文件，则将其原有内容覆盖，匹配时大小写不忽略]**



完成Step6:

要求按照测试6要求完成后，result内显示样式如下(一个答案占一行):

```
4
15
100
230
~
~
~
```

实验实战

- 7、在lab0工作区的csc/code目录下，存在fibonacci.c、main.c，其中fibonacci.c有点小问题，还有一个未补全的modify.sh文件，将其补完，以实现通过指令bash modify.sh fibonacci.c char int，可以将fibonacci.c中所有的char文本字段更改为int字段。[注意：对于指令bash modify.sh fibonacci.c char int，fibonacci.c及char及int可为任何合法文件或字符串，评测时评测modify.sh的正确性，而不是检查修改后fibonacci.c的正确性]
- 8、lab0工作区的csc/code/fibonacci.c成功更换字段后(bash modify.sh fibonacci.c char int)，现已有csc/Makefile和csc/code/Makefile，不全两个Makefile文件，要求在csc目录下通过指令make可在csc/code文件夹中生成fibonacci.o、main.o，在csc文件夹中生成可执行文件fibonacci，再输入指令make clean后只删除两个.o文件。[注意：不能修改fibonacci.h和main.c文件中的内容，提交的文件中fibonacci.c必须是修改后正确的fibonacci.c，可执行文件fibonacci作用是输入一个整数n，可以输出斐波那契数列前n项，每一项之间用空格分开。比如n=5，输出1 1 2 3 5]



完成Step7~8:

要求成功使用脚本文件modify.sh修改fibo.c，实现使用make指令可以生成.o文件和可执行文件，再使用指令make clean可以将.o文件删除，但保留fibo和.c文件。最终提交的文件树如下：

make后文件树

```
├── code
│   ├── fibo.c
│   ├── fibo.o
│   ├── main.c
│   ├── main.o
│   ├── Makefile
│   └── modify.sh
├── fibo
├── include
│   └── fibo.h
└── Makefile
```

make clean后文件树

```
├── code
│   ├── fibo.c
│   ├── main.c
│   ├── Makefile
│   └── modify.sh
├── fibo
├── include
│   └── fibo.h
└── Makefile
```

最终提交文件树

```
├── code
│   ├── fibo.c
│   ├── main.c
│   ├── Makefile
│   └── modify.sh
├── include
│   └── fibo.h
└── Makefile
```


实验提交流程

- `modify` 写代码。
- `git add & git commit <modified-file>` 提交到本地版本库。
- `git pull` 从服务器拉回本地版本库，并解决服务器版本库与本地代码的冲突。
- `git add & git commit <conflict-file>` 将远程库与本地代码合并结果提交到本地版本库。
- `git push` 将本地版本库推到服务器