实验一：

编译运行Linux内核并通过qemu+gdb调试

实验目的

- 熟悉Linux系统运行环境
- 制作根文件系统
- 掌握Linux内核编译方法
- 学习如何使用gdb调试内核
- 熟悉Linux下常用的文件操作指令

实验环境

- OS：Ubuntu 14.04 i386（32位）
- Linux内核版本：Kernel 2.6.26
- **注意**：本次实验必须在Ubuntu系统上实现，可直接在机房完成，或在个人PC上安装虚拟环境完成，请注意需要安装32位的Ubuntu镜像。

实验内容

一、制作根文件系统

1、下载并编译Linux内核

- 下载linux-2.6.26.tar.gz，解压缩得到目录linux-2.6.26，不妨称之为Linux源代码根目录(以下简称源码根目录)

```
mkdir ~/oslab
cd ~/oslab
wget https://mirrors.edge.kernel.org/pub/linux/kernel/v2.6/linux-2.6.26.tar.gz
tar -zvxf linux-2.6.26.tar.gz
```

- 进入源代码根目录，并执行编译指令(32位缺省编译)

```
cd ~/oslab/linux-2.6.26
make i386_defconfig
make
```

- make过程中若遇到问题，可参考以下方案解决：

```
# 问题1
gcc: error: elf_x86_64: No such file or directory
make[1]: *** [arch/x86/vdso/vdso.so.dbg] Error 1
make: *** [arch/x86/vdso] Error 2
```

```
# 问题1解决方案:
将linux-2.6.26/arch/x86/vdso目录下的Makefile文件中的 ‘-m elf_x86_64’ 改成 ‘-m64’, ’-m
elf_i386’ 改成 ‘-m32’

cd ~/oslab/linux-2.6.26
make    # 进入源代码根目录，再次执行make指令

# 问题2
undefined reference to `__mutex_lock_slowpath'
undefined reference to `__mutex_unlock_slowpath'
# 问题2解决方案
将linux-2.6.26/kernel目录下的mutex.c文件中的
# 如下行
static void noinline __sched
__mutex_lock_slowpath(atomic_t *lock_count);
# 改为:
static __used void noinline __sched __mutex_lock_slowpath(atomic_t *lock_count);

# 如下行
static noinline void __sched __mutex_unlock_slowpath(atomic_t *lock_count);
# 改为
static __used noinline void __sched __mutex_unlock_slowpath(atomic_t *lock_count);

cd ~/oslab/linux-2.6.26
make    # 进入源代码根目录，再次执行make指令
```

- **注意**：在编译过程中可能因为Ubuntu内的配置问题而产生各类bug，具体解决方案由报错而定，若仍有其他错误，请参考以下方式处理。（更好的处理方法是直接在搜索引擎中输入：linux编译问题+报错内容）
- 编译问题处理1
- 编译问题处理2
- 编译问题处理3
- 编译问题处理4
- 编译问题处理5
- 编译问题处理6
- 编译问题处理7
- 内核配置(make menuconfig)详述

2、准备模拟器qemu

- 直接安装qemu包即可

```
sudo apt-get install qemu
```

- 可能ubuntu官方镜像源上没有qemu包，将镜像源切换成ustc源即可，具体方法见下
- 更换apt-get源为ustc镜像源

3、制作根文件系统——方法一

- (1) 创建一个简单的应用程序

```
cd ~/oslab
touch test.c
```

加入以下代码:

```
#include <stdio.h>
#include <unistd.h>

void main(){
    while(1){
        printf("Hello!\n");
        sleep(2);
    }
}
```

创建好后使用GCC进行静态编译:

```
cd ~/oslab
gcc -static -o init test.c
```

- (2) 建立目标根目录映像

```
cd ~/oslab
dd if=/dev/zero of=myinitrd4M.img bs=4096 count=1024
mkfs.ext3 myinitrd4M.img
mkdir rootfs
sudo mount -o loop myinitrd4M.img rootfs
```

- (3) 将init拷贝到目标根目录下

```
sudo cp init rootfs/
```

- (4) 准备dev目录

```
sudo mkdir rootfs/dev
sudo mknod rootfs/dev/console c 5 1
sudo mknod rootfs/dev/ram b 1 0
sudo umount rootfs
```

- 至此即完成了包含简单应用的根目录镜像myinitrd4M.img
- (5) 使用qemu启动系统

```
cd ~/oslab
qemu-system-i386 -kernel ~/oslab/linux-2.6.26/arch/x86/boot/bzImage -initrd
~/oslab/myinitrd4M.img --append "root=/dev/ram init=/init"
```

- 至此,既可以看到系统能够启动，并且在启动后看到init的输出结果
- **提示**：可以使用以下指令终止qemu进程：

```
ps -ef | grep qemu | grep -v grep | awk '{print $2}' | xargs sudo kill
```

3、制作根文件系统——方法二：利用busybox生成根文件系统（推荐）

- （1）下载busybox

```
cd ~/oslab
wget https://busybox.net/downloads/busybox-1.30.1.tar.bz2   #下载
tar -jxvf busybox-1.30.1.tar.bz2     #解压
cd ~/oslab/busybox-1.30.1
```

- （2）编译busybox

```
make defconfig
make menuconfig  #修改配置如下:
    Settings ->
        Build Options
            [*] Build static binary (no share libs)
    Settings ->
        (-m32 -march=i386) Additional CFLAGS
        (-m32) Additional LDFLAGS
    Settings ->
        What kind of applet links to install ->
            (X) as soft-links
make
make install

# 某些同学在自己电脑上运行make时可能会报错，可输入 sudo apt install libncurses5-dev 解决
```

- （3）准备根文件系统

```
cd ~/oslab/busybox-1.30.1/_install
sudo mkdir dev
sudo mknod dev/console c 5 1
sudo mknod dev/ram b 1 0
touch init
    # 在init中写入以下内容（你可以使用vim或gedit编辑器写入，或在图形化界面中找到该文件，双击编辑）
    #!/bin/sh
    echo "INIT SCRIPT"
    mkdir /proc
    mkdir /sys
    mount -t proc none /proc
    mount -t sysfs none /sys
    mkdir /tmp
    mount -t tmpfs none /tmp
    echo -e "\nThis boot took $(cut -d' ' -f1 /proc/uptime) seconds\n"
    exec /bin/sh

chmod +x init

cd ~/oslab/busybox-1.30.1/_install
find . -print0 | cpio --null -ov --format=newc | gzip -9 > ~/oslab/initramfs-busybox-
x86.cpio.gz  # 注意：该命令一定要在busybox的 _install 目录下执行
# 注意：每次修改_install,都要重新执行该命令
```

- （4）运行

```
cd ~/oslab
qemu-system-i386 -s -kernel ~/oslab/linux-2.6.26/arch/x86/boot/bzImage -initrd
~/oslab/initramfs-busybox-x86.cpio.gz --append "root=/dev/ram init=/init"
```

  - 在qemu窗口可以看到成功运行，且进入shell环境

4、熟悉linux简单指令

- 目标：掌握ls、touch、cat、echo、mkdir、mv、cd、cp等基本指令
- 在上一步“利用busybox生成根文件系统” 运行成功之后，在qemu窗口可以看到已进入shell环境。此时就可以在我们自己制作的根文件系统中执行指令了。如下指令创建写入一个txt文件并移动文件：

```
/ # ls          # 查看当前目录下的所有文件/文件夹
/ # touch 1.txt # 创建1.txt
/ # ls
/ # echo i am 1.txt > 1.txt  # 向1.txt写入内容
/ # cat 1.txt   # 查看1.txt内容
/ # ls -l       # 查看当前目录下的所有文件/文件夹的详细信息
/ # mkdir 1     # 创建目录1
/ # mv 1.txt 1  # 将1.txt移动到目录1
/ # cd 1        # 打开目录1
/ # ls
```

二、gdb+qemu调试内核

1、gdb简介

- gdb是一款终端环境下常用的调试工具
- 使用gdb调试程序
  - ubuntu下安装gdb：sudo apt install gdb
  - 编译程序时加入-g选项，如：gcc -g -o test test.c
  - 运行gdb调试程序：gdb test
- 常用命令

```
r/run                 # 开始执行程序
b/break <location>    # 在location处添加断点，location可以是代码行数或函数名
b/break <location> if <condition> # 在location处添加断点，仅当caondition条件满足才中断运行
c/continue            # 继续执行到下一个断点或程序结束
n/next                # 运行下一行代码，如果遇到函数调用直接跳到调用结束
s/step                # 运行下一行代码，如果遇到函数调用则进入函数内部逐行执行
ni/nexti              # 类似next，运行下一行汇编代码（一行c代码可能对应多行汇编代码）
si/stepi              # 类似step，运行下一行汇编代码
list                  # 显示当前行代码
p/print <expression>  # 查看表达式expression的值
```

2、在qemu中启动gdb server

- 在终端中执行以下指令启动qemu运行内核：

```
qemu-system-i386 -s -S -kernel ~/oslab/linux-2.6.26/arch/x86/boot/bzImage -initrd
~/oslab/initramfs-busybox-x86.cpio.gz --append "root=/dev/ram init=/init"   # 可以看到
qemu在等待gdb连接
```

- 关于-s和-S选项的说明
  - -S freeze CPU at startup (use 'c' to start execution)
  - -s shorthand for -gdb tcp::1234 若不想使用1234端口，则可以使用-gdb tcp:xxxx来取代-s选项

3、建立gdb与gdb server之间的链接

- 在另外一个终端运行gdb，然后在gdb界面中运行如下命令：

```
gdb                 #这里一定是在另外一个终端运行，不能在qemu的窗口上输入
target remote:1234 #则可以建立gdb和gdbserver之间的连接
c                   #让qemu上的Linux继续运行
```

可以看到gdb与qemu已经建立了连接。但是由于没有加载符号表，无法根据符号设置断点。下面说明如何加入断点。

4、加载vmlinux中的符号表并设置断点

- 退出之前打开的qemu终端，重新执行第2步 "在qemu中启动gdb server "
- 在另外一个终端输入如下指令运行gdb，加载符号表

```
gdb                 #这里一定是在另外一个终端运行，不能在qemu的窗口上输入
file ~/oslab/linux-2.6.26/vmlinux    #加载符号表
target remote:1234 #建立gdb和gdbserver之间的连接
```

- 在gdb界面中设置断点

```
break start_kernel
c                   #继续运行到断点
```

5、重新配置Linux，使之携带调试信息

- 在原来配置的基础上，重新配置Linux，使之携带调试信息

```
cd ~/oslab/linux-2.6.26/
make menuconfig
    kernel hacking->
        [*] compile the kernel with debug info
```

- 重新编译并按照原方法执行

```
make
```

- 此时，若按照前面相同的方法来运行，则在start_kernel停下 来后，可以使用list来显示断点处相关的源代码

参考资料

- 内核编译与制作根文件系统