操作系统实验 1

题目：编译运行 Linux 内核并通过 qemu+gdb 调试

廖洲洲，PB17081504

- 主要步骤

一、 环境准备

1) 准备 Linux 内核：解压缩 linux-2.6.26.tar.gz，编译

```
tar -zvxf linux-2.6.26.tar.gz
cd linux-2.6.26
make i386_defconfig
make
```

2) 准备 qemu

二、 制作根文件系统

1) 准备一个应用程序，使用静态链接将其编译为可执行文件

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

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"
```

三、 利用 busybox 生成根文件系统

1) 下载 busybox

```
tar -jxvf busybox-1.30.1.tar.bz2 #解压
cd ~/oslab/busybox-1.30.1
```

2) 编译 busybox

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
#!/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-busyboxx86.cpio.gz
```

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"
```

四、 熟悉 linux 简单指令

五、 gdb+qemu 调试内核

1) 在 qemu 中启动 gdb server

```
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"

2) 建立 gdb 与 gdb server 之间的链接

```
gdb
target remote:1234
c
```

3) 加载 vmlinux 中的符号表并设置断点

```
gdb
file ~/oslab/linux-2.6.26/vmlinux
target remote:1234
break start_kernel
c
```

4) 重新配置 Linux，使之携带调试信息

- 实验过程中遇到的技术问题及解决方法

一、 问题一

```
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
```

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'

二、 问题二

```
undefined reference to `__mutex_lock_slowpath'
undefined reference to `__mutex_unlock_slowpath'
```

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);
```

三、 问题三

修改配置时：[*] Build static binary（no share libs）

*号不是打出来的，而是由空格切换

四、 问题四

准备根文件系统，在 init 中写入内容时，#!/bin/sh 需写入，它不是注释

五、 问题五

建立 gdb 与 gdb server 之间的链接时，需在另一个终端运行 gdp，不能在 qemu 的窗口输入。

- 实验总结

通过本次实验，初步熟悉了 linux 系统运行环境，制作了根文件系统，学习了 linux 内核编译方法，学习了如何使用 gdb 调试内核，同时熟悉了 linux 下常用的文件操作指令。