Linux 系統建構實作　　劉浩然 05-2011

1　目標

運用 QEMU 虛擬機器環境，學習如何從無到有，建構一個可開機的 Linux 系統。

一個 Linux 系統包含了 boot loader（開機程式）、kernel（作業系統核心）、initramfs（初始檔案系統）、rootfs（開機完成後使用的檔案系統，內存放系統的應用程式）。

Boot loader: 在 BIOS 之後第一個被執行的程式，它依據使用者設定，載入作業系統核心，並執行之。
Kernel: 作業系統核心，我們採用 Linux。
initramfs: 初始檔案系統，任務為載入開機階段用到的各種驅動程式（如: SATA 驅動程式）
rootfs: 系統用的檔案系統，內存各種基本系統程式（如: init,getty, sh, ls 等）

2　作業系統與程式

這裡使用 Linux 作業系統為實作的操作環境。你可以使用任何喜愛的 Linux 套件，如 openSuSE, Fedora 等等。這裡使用 Ubuntu 11.304 為本次實作的標準環境。Ubuntu 11.04 可以在 http://www.ubuntu.com 下載。至於 QEMU，使用 Ubuntu 內附的即可。

實作中使用 GRUB 作為 boot loader; busybox 則提供 rootfs 裡的基本系統程式。你還需要至下列網站下載下列程式:

GRUB 1.99~rc2　— http://www.gnu.org/software/grub/
kernel 2.6.38.6 — http://www.kernel.org
Busybox 1.18.4　— http://www.busybox.net

3　建立虛擬磁碟

以一個檔案 vmhd 來作為 QEMU 的虛擬硬碟，當它內容都準備好之後，就可以在 QEMU 裡驗證。首先在命令列下使用 'dd' 建立一個 100MB 的檔案。

```
$ dd if=/dev/zero of=vmhd bs=1M count=100
```

接著以 'fdisk' 對 vmhd 做 partition 分割。

```
$ fdisk -C 1024 -u vmhd

Command (m for help): n
Command action
   e   extended
   p   primary partition (1-4)
p
Partition number (1-4): 1
First sector (63-16450559, default 63):
Using default value 63
Last sector, +sectors or +size{K,M,G} (63-16450559, default 16450559):
Using default value 16450559

Command (m for help): p

Disk vmhd: 0 MB, 0 bytes
255 heads, 63 sectors/track, 1024 cylinders, total 0 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x4b7dd39a

Device Boot      Start         End      Blocks   Id  System
 vmhd1              63    16450559     8225248+  83  Linux

Command (m for help): w
The partition table has been altered!

Syncing disks.
```

注意 partition 的劃分，起點在第 63 個 sector。第 0 個 sector 為 MBR（主開機磁區，Master Boot Record）；GRUB 會應用第 1 至第 62 個 sector 存放其部份開機程式。

在 vmhd 上建立檔案系統

使用 loopback device，將 vmhd 變成系統上的一個裝置，以便建立檔案系統。整個 vmhd 對應到 /dev/loop0，vmhd 的 partition 1 對應到 /dev/loop1。在 partition 1 上用 mkfs 建立 ext3 檔案系統，並將它掛載（mount）在 /mnt。

```
$ sudo losetup /dev/loop0 vmhd
$ sudo losetup -o 32256 /dev/loop1 vmhd
$ sudo mkfs -t ext3 /dev/loop1
$ sudo mount /dev/loop1 /mnt
```

4　安裝 GRUB boot loader

將 grub-1.99~rc2.tar.gz 解開、編譯及安裝。編譯過程需要使用 flex 及 bison, 請先安裝這二個軟體。

```
$ sudo apt-get install flex bison
$ tar zxf grub-1.99~rc2.tar.gz
$ cd grub-1.99~rc2/
$ ./configure
$ make
$ sudo make install
$ cd ..
```

把 GRUB 寫入 vmhd

下面步驟會覆寫 vmhd 的 MBR、將 GRUB core 寫入 MBR 至 partition1 間的小空間（第 1 至 62 sector）及將 GRUB 模組（驅動程式, 以存取不同的儲存裝置或檔案系統）存入 partition 1 上的 ext4 檔案系統（放在 /boot/grub 目錄下）。

```
$ mkdir -p /mnt/boot/grub
$ sudo bash -c "cat > /mnt/boot/grub/device.map" <<EOF
> (hd0)   /dev/loop0
> (hd0,1) /dev/loop1
> EOF
$ sudo grub-install --no-floppy --boot-directory=/mnt/boot /dev/loop0
```

這裡順便寫入 GRUB 的開機選單設定檔。Linux 核心在稍後會放在 /boot/vmlinuz

```
$ sudo bash -c "cat > /mnt/boot/grub/grub.cfg" <<EOF
> set timeout=5
> set default=0
> menuentry "My Linux kernel" {
>    set root=(hd0,1)
>    linux /boot/vmlinuz
>    initrd /boot/initramfs.img
> }
> EOF
```

device.map 用途在告訴 grub-install: 對應到被安裝的 /dev/loop0 及 /dev/loop1, 開機時是那個 BIOS 磁碟編號, 這資訊沒告訴它, GRUB core 找不到 GRUB modules。

5　編譯 Linux kernel

將 linux-2.6.38.6.tar.bz2 解開、設定及編譯。你可以使用 make menuconfig 文字選單，自定 Linux 核心要包含那些功能及驅動程式，這裡使用預設值 （make defconfig）。

make menuconfig 需要安裝 ncurses library header。

```
$ sudo apt-get install libncurses5-dev
```

```
$ tar jxf linux-2.6.38.6.tar.bz2
$ cd linux-2.6.38.6/
$ make defconfig
$ make
```

將完成的 Linux kernel 複製到 vmhd 的 /boot 目錄下。

```
$ sudo cp arch/x86/boot/bzImage /mnt/boot/vmlinuz
$ cd ..
```

6　製作 initramfs

initramfs 的內容組成，由 BusyBox 提供。BusyBox 是常用的 unix 工具程式之縮小版，組合為單一執行檔。嵌入式系統上常見。

編譯 BusyBox

```
$ tar jxf busybox-1.18.4.tar.bz2
$ cd busybox-1.18.4/
$ make defconfig
$ make
```

建立 initramfs 目錄

```
$ mkdir ../initramfs
```

安裝 BusyBox 至 initramfs 目錄

```
$ make CONFIG_PREFIX=../initramfs install
```

initramfs 階段的目的，在讓 kernel 可以順利 mount 起 rootfs，存取 rootfs 所必需的驅動

程式必須放入 initramfs。並在 initramfs 裡的 /init script 裡用 insmod 載入。本實作並無特殊驅動程式需求，因此而接帶入 rootfs 即可。

動手寫 initramfs/init script

```
$ cd ../initramfs
$ vi init
```

將下列內容寫入 initramfs/init script。

```
======開始======
#!/bin/sh
# Mount things needed by this script
mount -t proc proc /proc
mount -t sysfs sysfs /sys

# Create device nodes
mdev -s

# Function for parsing command line options with "=" in them
# get_opt("root=/dev/sda") will return "/dev/sda"
get_opt() {
        echo "$@" | cut -d "=" -f 2
}

# init and root defaults
init=/sbin/init
root=/dev/sda1

# Override default values if kernel command line has definitions
for i in `cat /proc/cmdline` ; do
        case $i in
        root=*)
                root=`get_opt $i`
                ;;
        init=*)
                init=`get_opt $i`
                ;;
        esac
```

```
done

# Mount the root device
mount $root /newroot

# Check if init exists and is executable
if [ -x /newroot/$init ] ; then
    # unmount all other mounts so that the ram used by
    # the initramfs can be cleared after switch_root
    umount /sys /proc

    # switch to the new root and execute init
    exec switch_root /newroot $init
fi

# This will only be run if the exec above failed
echo "Failed to chroot, dropping to a shell"
exec sh
```

======結束======

```
$ chmod a+x init
```

建立 initramfs init script 階段使用的目錄。

```
$ mkdir proc sys newroot lib
```

Copy busybox使用的 shared library 到 initramfs

用 ldd 查看 busybox 使用了那些 shared library, 複製過來到 initramfs/lib 下。

```
$ ldd bin/busybox
      linux-gate.so.1 =>  (0x00500000)
      libm.so.6 => /lib/i386-linux-gnu/libm.so.6 (0x00876000)
      libc.so.6 => /lib/i386-linux-gnu/libc.so.6 (0x00168000)
      /lib/ld-linux.so.2 (0x0014a000)

$ cp /lib/ld-linux.so.2 lib/
$ cp /lib/i386-linux-gnu/{libm.so.6,libc.so.6} lib/
```

將 initramfs 目錄打包成 initramfs.img

find將 initramfs 內所有檔案列表送給 cpio，cpio 依列表抓檔打包，再以 gzip壓縮。

```
$ find ./ | cpio -o -H newc | gzip > ../initramfs.img
$ cd ..
```

將完成的 initramfs.img 移至 vmhd 的 /boot 目錄下，並更正檔案擁有者為 root。

```
$ sudo mv initramfs.img /mnt/boot/
$ sudo chown root.root /mnt/boot/initramfs.img
```

到這裡，我們已完成了大半，只剩下 rootfs 下的 UNIX 程式和設定檔。

7　製作 rootfs

當 initramfs switch_root 至 /newroot （即 vmhd 的 partition 1），便執行 /sbin/init。它是rootfs裡第一個被啓動的程式，按 /etc/inittab的設定，依序啓動其他應用程式（啓動 server process、mount 檔案系統及執行 init.d 目錄下的 rcX scripts）。

安裝 UNIX 工具程式至 rootfs

把 busybox 安裝到 vmhd 根目錄下。

```
$ cd busybox-1.18.4/
$ sudo make CONFIG_PREFIX=/mnt install
```

建立 rootfs 裡會用到的目錄 。

```
$ sudo mkdir /mnt/{lib,sys,proc,dev}
```

複製busybox 會用到的 shared library 。

```
$ sudo cp /lib/ld-linux.so.2 /mnt/lib/
$ sudo cp /lib/i386-linux-gnu/{libm.so.6,libc.so.6} /mnt/lib/
```

etc 目錄下的設定

BusyBox 內提供了一組基本設定，複製過來使用。

```
$ cp -R examples/bootfloppy/etc /mnt/
$ cd /mnt/etc
```

下面第一行指示 /sbin/init 先執行 rcS（系統啓動設定都在這），之後在 console 執行 sh 提供命令列環境，respawn 表示 sh process 結束的話要再重生一個，第三行為在 tty2 也提供一個 sh 環境，第四行指示 init 在收到 ctrl-alt-del 時 umount 所有已掛載的檔案系統。

```
$ cat inittab
::sysinit:/etc/init.d/rcS
::respawn:-/bin/sh
tty2::askfirst:-/bin/sh
::ctrlaltdel:/bin/umount -a -r
```

在 fstab 裡定義的掛載點，在 mount -a 時都會被自動掛載到指定目錄下。注意這個檔案和 busybox 的不同，加了第二行。你要自行補上，rc.S 在執行 mdev -s 時才不會失敗。

```
$ cat fstab
proc      /proc     proc      defaults    0   0
sysfs     /sys      sysfs     defaults    0   0
```

系統啓動設定都集中寫在 rcS 裡，第三行掛載 fstab 裡定義的檔案系統，第四行 mdev 會從 /sys 讀取系統裝置資訊，並在 /dev 下自動產生 device files 。注意 rcS 檔案和 busybox 的不同，加上了第四行。請自行補上。

```
$ cat init.d/rcS
#! /bin/sh

/bin/mount -a
/sbin/mdev -s
```

OK !大功告成。將 vmhd 從 /mnt 卸載，並將 loop0，loop1 映射至 vmhd 的設定解除。

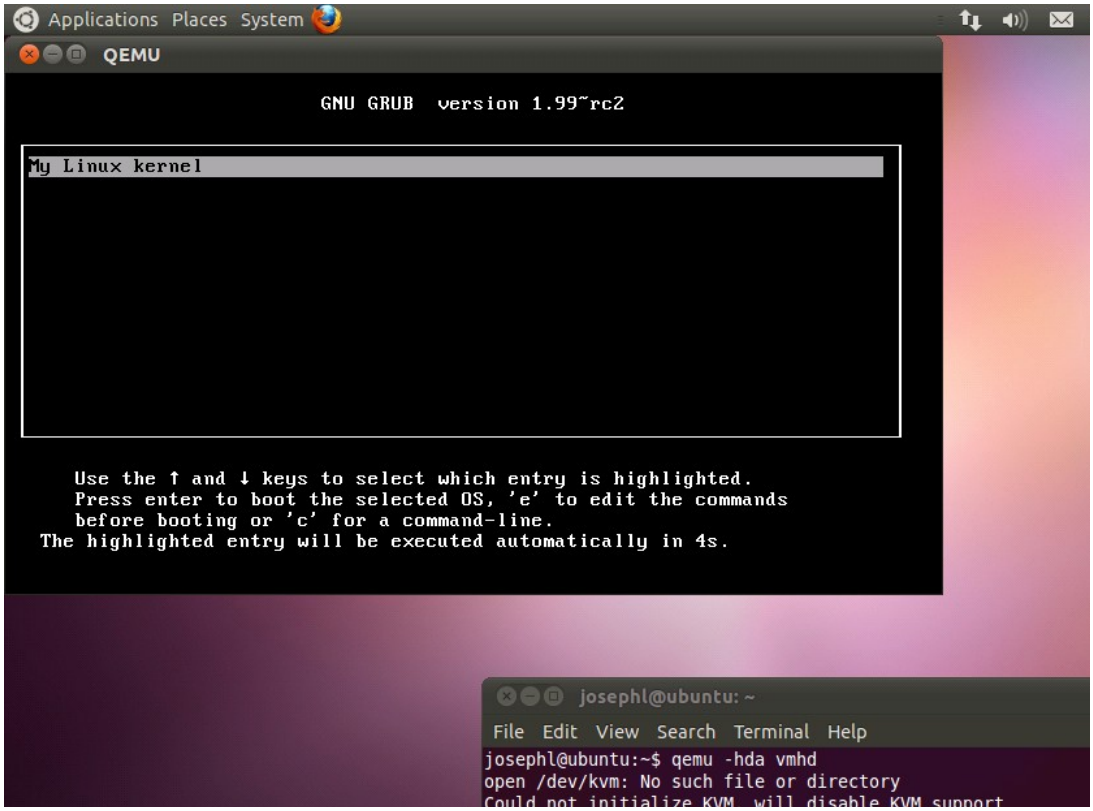
```
$ cd
$ sudo umount /mnt
$ sudo losetup -d /dev/loop0 /dev/loop1
```

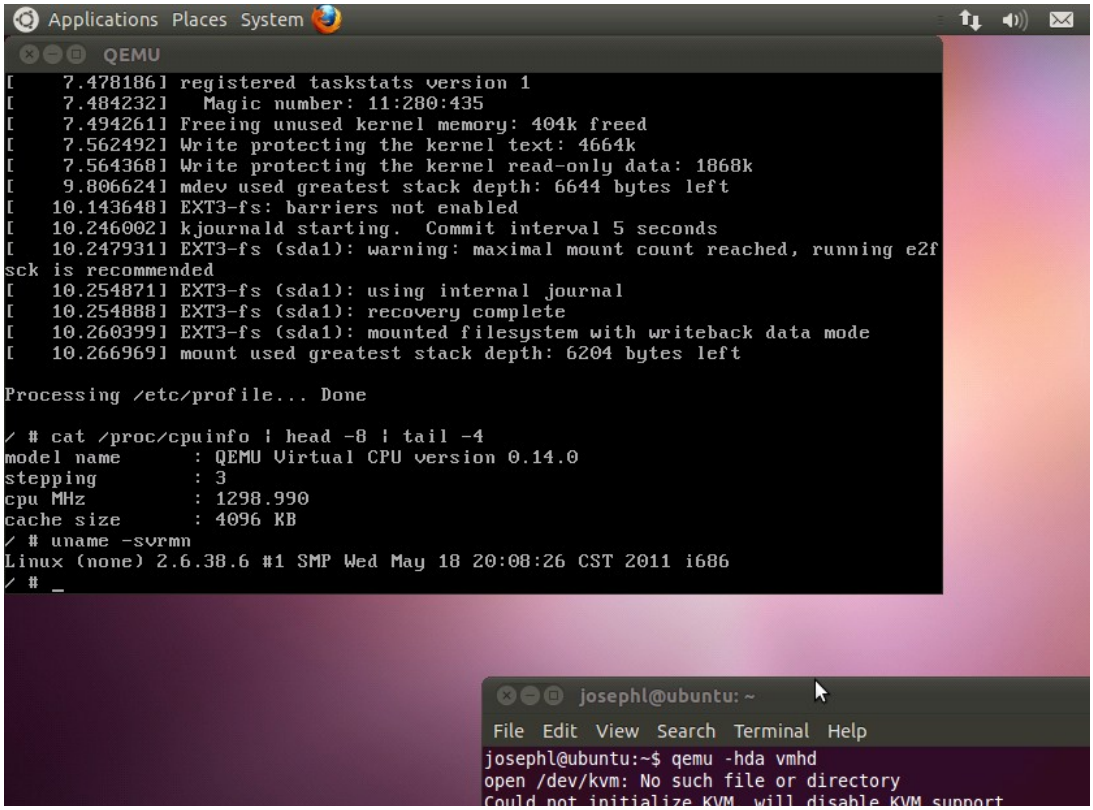
8　QEMU 實驗開機

安裝 QEMU 套件後，執行 QEMU 並指定 vmhd 為第一個系統硬碟。請你也按本實作指引自行演練。確實瞭解系統構成元件之間的關係，並熟習相關操作指令。

```
$ sudo apt-get install qemu
$ qemu -hda vmhd
```

QEMU 開機畫面快照


Applications Places System
QEMU
GNU GRUB version 1.99~rc2
My Linux kernel
Use the ↑ and ↓ keys to select which entry is highlighted.
Press enter to boot the selected OS, 'e' to edit the commands
before booting or 'c' for a command-line.
The highlighted entry will be executed automatically in 4s.
josephl@ubuntu: ~
File Edit View Search Terminal Help
josephl@ubuntu:~$ qemu -hda vmhd
open /dev/kvm: No such file or directory


Applications Places System
QEMU
[7.478186] registered taskstats version 1
[7.484232] Magic number: 11:280:435
[7.494261] Freeing unused kernel memory: 404k freed
[7.562492] Write protecting the kernel text: 4664k
[7.564368] Write protecting the kernel read-only data: 1868k
[9.806624] mdev used greatest stack depth: 6644 bytes left
[10.143648] EXT3-fs: barriers not enabled
[10.246002] kjournald starting. Commit interval 5 seconds
[10.247931] EXT3-fs (sda1): warning: maximal mount count reached, running e2fsck is recommended
[10.254871] EXT3-fs (sda1): using internal journal
[10.254888] EXT3-fs (sda1): recovery complete
[10.260399] EXT3-fs (sda1): mounted filesystem with writeback data mode
[10.266969] mount used greatest stack depth: 6204 bytes left
Processing /etc/profile... Done
/ # cat /proc/cpuinfo | head -8 | tail -4
model name : QEMU Virtual CPU version 0.14.0
stepping : 3
cpu MHz : 1298.990
cache size : 4096 KB
/ # uname -svrmn
Linux (none) 2.6.38.6 #1 SMP Wed May 18 20:08:26 CST 2011 i686
/ #
josephl@ubuntu: ~
File Edit View Search Terminal Help
josephl@ubuntu:~$ qemu -hda vmhd
open /dev/kvm: No such file or directory