学　　号 14284060xx
等　　第

苏 州 大 学 实 验 报 告

智能环境监测项目实训

院 (系) 名 称：　电子信息学院

专 业 名 称：　14 通信工程 (嵌入式培养)

学 生 姓 名：　某某某

课 程 名 称：　Linux 操作系统

2015-2016 学年第一学期

摘要

本文档是项目实训报告的模板。但因为实施过程更接近于模仿 (emulation)，所以很难形成正式的项目报告，因此并不按照正式的章节划分。而是按照实训的时间顺序，用各阶段 (phase) 做一级标题，二级标题 (如 1.1, 1.2 等) 则限定为本阶段安排的各种活动 (activity)，在写作自己的报告时应秉承此原则。如果各活动关系比较松散，可以每一个活动做为一个一级标题 (相当于一章)，可根据实际情况考虑是否采用后一种组织方案。

Activity. A distinct, scheduled portion of work performed during the course of a project. 活动：在进度计划中所列，并在项目过程中实施的工作组成部分。**Project Phase.** A collection of logically related project activities that culminates in the completion of one or more deliverables. 项目阶段：一组具有逻辑关系的项目活动的集合，通常以一个或多个可交付成果的完成为结束。

这里的所有模板仅作为排版参考，内容上应以自己所做实验为依据。中文的模板在后面。

关键词：Xilinx SDK；Zedboard；Appweb；MJPG-Streamer

合作者： 合作甲 14284060xx 14 通信工程 (嵌入式培养)
合作乙 14284060xx 14 通信工程 (嵌入式培养)
合作丙 14284060xx 14 通信工程 (嵌入式培养)

目录

1 Zynq Linux 系统介绍 **5**

1.1 A First Look 5

1.1.1 Introduction 5

1.1.2 Objectives 5

1.1.3 Typographic Conventions 5

1.1.4 Before You Start 6

1.1.5 Initializing the Workshop Environment 6

1.1.6 General Flow for this Lab 6

1.1.7 Power up the Board and Log in 7

1.1.8 Exploring the Embedded Linux Environment 7

1.1.9 Conclusion 12

1.1.10 Completed Solution 13

1.2 Build and Boot an Image 13

1.2.1 Introduction 13

1.2.2 Objectives 13

1.2.3 Preparation 14

1.2.4 General Flow for this Lab 14

1.2.5 Choosing a Linux Platform 14

1.2.6 Building the Linux Image 15

1.2.7 Booting the System 19

1.2.8 Conclusion 25

1.2.9 Completed Solution 26

2 网络和 Linux TCP/IP 编程 **27**

2.1 Networking and TCP/IP 27

2.1.1 Introduction 27

2.1.2 Objectives 27

2.1.3 Preparation 28

2.1.4 Exploring Network Features . . . 28
2.1.5 General Flow for this Lab . . . 28
2.1.6 Logging In Using Telnet Step . . . 28
2.1.7 Transferring Files with FTP . . . 30
2.1.8 Using NFS . . . 30
2.1.9 Navigating the Web Page on HTTP . . . 35
2.1.10 Building the Web-Enabled Application . . . 36
2.1.11 Conclusion . . . 39
2.1.12 Completed Solution . . . 40
2.2 Linux 网络编程实验 . . . 41
2.2.1 实验目的 . . . 41
2.2.2 实验原理 . . . 41
2.2.3 步骤与现象 . . . 41
2.2.4 关键代码分析 . . . 44

1 Zynq Linux 系统介绍

1.1 A First Look

1.1.1 Introduction

Embedded Linux is the use of a Linux operating system in embedded systems. Unlike desktop and server versions of Linux, embedded versions of Linux are designed for devices with relatively limited resources. The ARM® Cortex™-A9 processor used in Xilinx Zynq® All Programmable SoCs support embedded Linux. In most of the labs in this workshop, you will run embedded Linux, build using the PetaLinux tools, on the ARM Cortex-A9 MPcore.

This first lab is a basic introduction to embedded Linux and the development board that you are using for the workshop. The basic activities covered here will be used repeatedly through the later lab sessions, so be sure to ask your instructor if you have any questions or concerns.

1.1.2 Objectives

After completing this lab, you will be able to:

- Power on the development board used in the workshop
- Log in to the Zynq Linux system
- Make comparisons between the embedded Linux and desktop Linux environments

1.1.3 Typographic Conventions

Commands to be executed on the development (desktop) workstation look like the following:

```
[host]$ command and parameters
```

Commands to be executed on the ARM processor Linux target look like the following:

```
# run my Linux application
```

1.1.4 Before You Start

Before you start, ensure that the:

- Power switch is in the 'off' position
- JTAG cable is connecting the development board to the PC
- Serial cable is connecting the development board to the PC
- Power cable for the development board is connected
- The Ethernet port on the development board connects to the Ethernet port of the desktop (host) machine
- The `BOOT.BIN` and `image.ub` files are copied from the `~/emblnx/sources/lab1/SDCard` directory
- The SD card is inserted back into the target board.
- Set the jumpers to boot from the SD card as shown in Fig. 1.1.

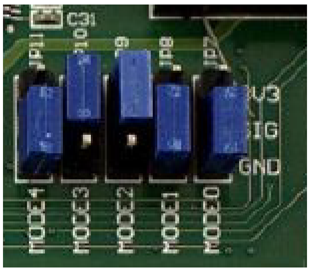


图 1.1: SD Card Boot - jumper settings

1.1.5 Initializing the Workshop Environment

By default, your Ubuntu image has already set up the workshop environment for you.

If your workstation has been restarted or logout, run the following command to start DHCP server on the host:

```
[host] $ sudo service isc-dhcp-server restart
```

1.1.6 General Flow for this Lab

- Step 1: Power up the Board and Login In
- Step 2: Explore the Embedded Linux Environment

1.1.7 Power up the Board and Log in

1. Power up the board and run the DHCP server on the host.
   1) Power ON the board.
   2) Run the DHCP server:
      `[host] $ sudo service isc-dhcp-server restart`
2. Set the serial port terminal.
   1) Ensure that `/dev/ttyACM0` is set to read/write access:
      `[host]$ sudo chmod 666 /dev/ttyACM0`
   2) In the dashboard, in the Search field, enter the serial port.
   3) Select the **Serial port terminal** application from the desktop.
   4) Reset (BTN7) the board to see the booting info.
      Watch the GtkTerm (Serial Port) console as the board goes through the boot process. Messages similar to the following can be found in the board console as Fig. 1.2 and Fig. 1.3.

```
U-Boot 2014.01 (Oct 06 2014 - 23:22:45)

Memory: ECC disabled
DRAM:   512 MiB
MMC:    zynq_sdhci: 0
SF: Detected S25FL256S_64K with page size 256 Bytes, erase size 64 KiB, total 32 MiB
*** Warning - bad CRC, using default environment

In:     serial
Out:    serial
Err:    serial
Net:    Gem.e000b000
U-BOOT for petalinux

Hit any key to stop autoboot:  1 █
```

图 1.2: Linux booting process in the board console

1.1.8 Exploring the Embedded Linux Environment

1. Explore the booting message and basic Linux commands.
   1) Scroll up in the terminal window and review the bootup log. Existing Linux users should recognize the output. You will see `image.ub` loading, drivers loading such as USB, SD, etc., and the starting of the uWeb server.

```
Freeing unused kernel memory: 3644K (c067c000 - c0a0b000)
INIT: version 2.88 booting
Starting Bootlog daemon: bootlogd.
Creating /dev/flash/* device nodes
random: dd urandom read with 8 bits of entropy available
starting Busybox inet Daemon: inetd... done.
Starting uWeb server:
NET: Registered protocol family 10
update-rc.d: /etc/init.d/run-postinsts exists during rc.d purge (continuing)
 Removing any system startup links for run-postinsts ...
  /etc/rcS.d/S99run-postinsts
INIT: Entering runlevel: 5
Configuring network interfaces... done.
Stopping Bootlog daemon: bootlogd.

Built with PetaLinux v2014.2 (Yocto 1.6) petalinux /dev/ttyPS0
petalinux login:
```

图 1.3: Linux booting process in the board console

2) Log in by entering `root` as both the login and password.
3) Spend the next 15 minutes exploring basic Linux commands such as: `ls -l, vi, whoami, date`
4) Run the following command to list the applications currently installed:

   `# ls /bin`

2. Use the gpio-demo application to test the GPIOs. The gpio-demo application is used to write value to the GPIO peripheral or read value from the GPIO peripheral.

   1) Use the following command to see the available GPIOs in the system. See Fig. 1.4

      `# ls /sys/class/gpio`

      The GPIOs are presented as `gpiochip<ID>` in the directory. Have a look at the file `/sys/class/gpio/gpiochip<ID>/label`.

      For example:

      `# cat /sys/class/gpio/gpiochip243/label`

      The GPIO label file contains the GPIO label. The label contains the GPIO's physical address information. The GPIO label format is `/amba@0/gpio@<PYHSICAL_ADDRESS>`. As shown in Fig. 1.5.

      In the ZedBoard system, the on-board GPIOs are: eight LEDs (eight channels), five buttons (five channels), and eight switches (eight channels).

      The `gpiochip<ID>` to GPIOs mapping is:

`gpiochip235 for 8 switches;`

`gpiochip243 for 8 LEDs;`

`gpiochip251 for 5 buttons;`

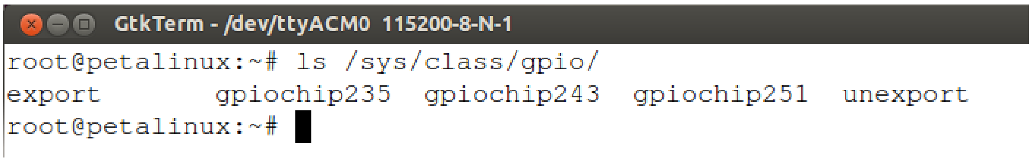


图 1.4: Checking for the available GPIOs in the system

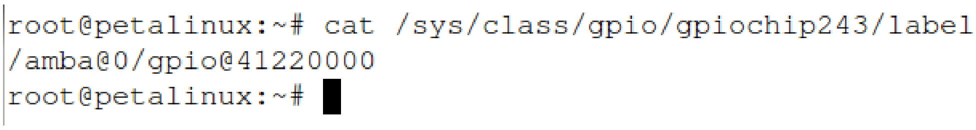


图 1.5: GPIO Physical Address Information

2) Run the following command to turn ON all eight LEDs (labeled as LD0 to LD7 on the board):

   `# gpio-demo -g 243 -o 255`

   Note: The output is in HEX format using the lower eight bits of the HEX value written. For example, in this case the equivalent of d'255 is 0xFF.

3) Run the following command to print the status of the eight DIP switches (labeled as SW0 to SW7 on the board):

   `# gpio-demo -g 235 -i`

   Note that you can try changing the DIP switch values.

3. Find the CPU information and interrupts.

   Another interesting place to explore is the `/proc` directory. This is a virtual directory that provides a window into the kernel. For example, the file `/proc/cpuinfo` contains details about the CPU, `/proc/interrupts` provides interrupt statistics, and so on.

   1) Enter the following command (Results shows as Fig. 1.6):

      `# cat /proc/cpuinfo`

      The contents of `/proc/cpuinfo` shows the processor information, such as its version and hardware features. Your `/proc/cpuinfo` may be different than above, depending on the configuration of the processor.

   2) Enter the following command (ref Fig. 1.7):

      `# cat /proc/interrupts`

```
root@petalinux:~# cat /proc/cpuinfo
processor       : 0
model name      : ARMv7 Processor rev 0 (v7l)
Features        : swp half thumb fastmult vfp edsp neon vfpv3 tls vfpd32
CPU implementer : 0x41
CPU architecture: 7
CPU variant     : 0x3
CPU part        : 0xc09
CPU revision    : 0

processor       : 1
model name      : ARMv7 Processor rev 0 (v7l)
Features        : swp half thumb fastmult vfp edsp neon vfpv3 tls vfpd32
CPU implementer : 0x41
CPU architecture: 7
CPU variant     : 0x3
CPU part        : 0xc09
CPU revision    : 0

Hardware        : Xilinx Zynq Platform
Revision        : 0000
Serial          : 0000000000000000
```

图 1.6: Viewing the CPU information

`/proc/interrupts` shows the interrupts information of the system. Your results may be different depending on when the command was executed and what were the other commands executed to access the hardware devices. `/proc/interrupts` tells you what interrupts are present in your system, their type, and how many interrupts have happened.

3) Open another terminal window on the desktop machine.
4) Enter `cat /proc/cpuinfo` and compare with the embedded Linux information by using the same command (Fig. 1.8).

   Another thing to note is the standard Linux directory structure: `/bin`, `/dev`, `/tmp`, `/var`, and so on.

4. Use `ping` command to test the network connection.

   1) After the system boots, log into the system by entering `root` as both the login name and password.
   2) Execute the ping command to ping the host machine.

      `# ping 192.168.1.1`

      You should see the response from the host machine.

```
root@petalinux:~# cat /proc/interrupts
           CPU0       CPU1
 27:          0          0       GIC  27  gt
 29:       1093        671       GIC  29  twd
 35:          0          0       GIC  35  f800c000.ps7-ocmc
 40:          0          0       GIC  40  f8007000.ps7-dev-cfg
 43:       4245          0       GIC  43  ttc_clockevent
 45:          0          0       GIC  45  f8003000.ps7-dma
 46:          0          0       GIC  46  f8003000.ps7-dma
 47:          0          0       GIC  47  f8003000.ps7-dma
 48:          0          0       GIC  48  f8003000.ps7-dma
 49:          0          0       GIC  49  f8003000.ps7-dma
 51:          8          0       GIC  51  e000d000.ps7-qspi
 54:          7          0       GIC  54  eth0
 56:         35          0       GIC  56  mmc0
 72:          0          0       GIC  72  f8003000.ps7-dma
 73:          0          0       GIC  73  f8003000.ps7-dma
 74:          0          0       GIC  74  f8003000.ps7-dma
 75:          0          0       GIC  75  f8003000.ps7-dma
 82:        618          0       GIC  82  xuartps
IPI1:          0        152  Timer broadcast interrupts
IPI2:       1481       1746  Rescheduling interrupts
IPI3:          0          0  Function call interrupts
IPI4:         27         74  Single function call interrupts
IPI5:          0          0  CPU stop interrupts
IPI6:         92         96  IRQ work interrupts
IPI7:          0          0  completion interrupts
Err:          0
```

图 1.7: Viewing the Interrupts

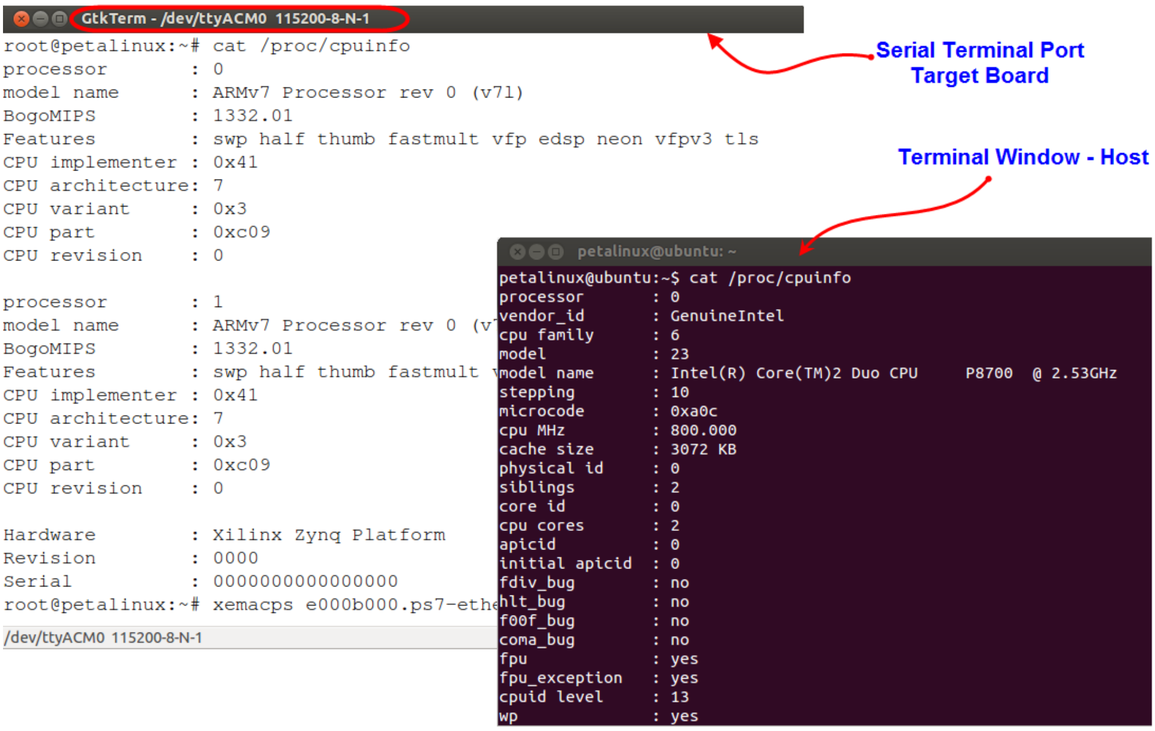


图 1.8: Serial Port and Terminal Window - cpuinfo

3) Execute the ping command from the host machine terminal window to see the response from the target board.

   `[host]$ ping 192.168.1.2`

   The static ip address has been assigned when the system was built. You should see the response from the target machine.

4) Close the GtkTerm window

5) Power OFF the board.

1.1.9 Conclusion

The purpose of this lab was to introduce you to the embedded Linux target and demonstrate its heritage in the desktop Linux genealogy. This is one of the immediate benefits of embedded Linux. As an application and user environment, it has tremendous commonality with standard desktop Linux platforms.

Although brief, this introduction should have provided you with some basic experience with setting up and powering on the board, and logging into and navigating around the embedded Linux target. These basic capabilities will be expanded upon in subsequent lab sessions.

1.1.10 Completed Solution

If you want to run the solution then copy `BOOT.bin` and `image.ub` from the `sources\lab1\SDCard` directory onto a SD card. Place the SD card in the ZedBoard. Set ZedBoard in the SD Card boot mode. Connect the ZedBoard to the host machine using Ethernet cable.

Run the following command to start DHCP server on the host:

```
[host] $ sudo service isc-dhcp-server restart
```

Power ON the board. Set the terminal session.

Press PS-SRST (BTN7) button. Let the board boot. Login into the system and test the lab.

1.2 Build and Boot an Image

1.2.1 Introduction

The most basic skill required for developing embedded Linux is working in the cross-compilation environment: compiling the kernel, libraries, and applications and downloading the resulting image onto the embedded target. The purpose of this lab is to familiarize you with this process.

This lab will prepare you for the most basic task of working with embedded Linux: how to build and boot the operating system and applications. Embedded Linux target processors, such as the ARM® Cortex™-A9 MPcore, are usually developed in a cross-compilation environment. This means that the kernel and applications are compiled on a development machine (in this case, a Linux PC having a non-target processor), and then downloaded onto the target.

The PetaLinux tools support a number of configuration architectures that automate much of this process. In this lab, you will learn how to use these tools and how to download the resulting embedded Linux image onto the hardware platform.

QEMU is a generic and open-source machine emulator integrated into the PetaLinux tools. In this lab, you will use QEMU to run the Linux built for the ARM Cortex-A9 MP-core system. It can achieve near native performance by executing the guest code directly on the host CPU

1.2.2 Objectives

After completing this lab, you will be able to:

- Build the ARM Cortex-A9 MPcore Linux kernel and applications
- Boot the resulting system image in QEMU

- Download the resulting system image onto the development board

1.2.3 Preparation

If this is the first lab that you are performing, then refer to the "Before You Start" section of Lab 1 for necessary preparatory information on how to set up the environment.

If your workstation has been restarted or logout, run the following command to start DHCP server on the host:

```
[host] $ sudo service isc-dhcp-server restart
```

Please refer to the "Initializing the Workshop Environment" section of Lab 1 for detailed information.

1.2.4 General Flow for this Lab

- Step 1: Choosing a Linux Platform
- Step 2: Building the Linux Image
- Step 3: Booting the System

1.2.5 Choosing a Linux Platform

A Linux platform tells what to build into the Linux image; it tells the following information:

- The hardware platform information such as address mapping, interrupts, and the processor's characteristics, for example
- The Linux kernel settings
- User space applications settings
- File system settings
- Flash partition table settings

1. Change the path to the project directory.
   1) Run the following commands to create and change to the project directory path:
      ```
      [host] $ mkdir ~/emblnx/labs/lab2
      [host] $ cd ~/emblnx/labs/lab2
      ```
      Each lab in this workshop is installed in the `~/emblnx/labs` directory. Adjust the path if you have installed the labs at a different path.
2. Use the `petalinux-create` command to create a new embedded Linux platform and choose the platform.

1) Source the PetaLinux tools if you didn't do it. It assumes that PetaLinux is installed in `/opt/Xilinx` directory

```
[host] $ source /opt/Xilinx/petalinux-v2014.2-final/settings.sh
```

2) Run the following command from the `lab2` directory to create a new Petalinux project (Fig. 1.9):

```
[host] $ petalinux-create -t project -s \
/opt/pkg/Avnet-Digilent-ZedBoard-v2014.2-final.bsp
```

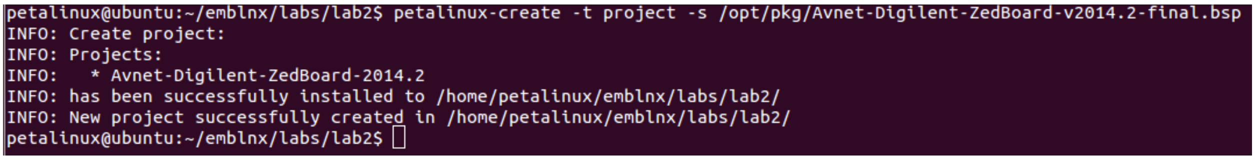


图 1.9: Creating a new PetaLinux project

The above command assumes that the board support package (BSP) is installed in the `/opt/pkg` directory. Modify the path if the BSP is in a different location.

The command will create the PetaLinux software project directory: `Avnet-Digilent-ZedBoard-2014.2` under `~/emblnx/labs/lab2`. (Fig. 1.10)

A PetaLinux project directory contains configuration files of the project, the Linux subsystem, and the components of the subsystem. `petalinux-build` builds the project with those configuration files. User can run `petalinux-config` to modify them. Below is the PetaLinux project directory.

3) Change the directory to `~/emblnx/labs/lab2/Avnet-Digilent-ZedBoard-2014.2`.

1.2.6 Building the Linux Image

1. Now that you have selected a pre-built platform, build a Linux image based on this platform.

   1) Enter the following command to build the Linux image: Note: if you find some errors like this, execute "petalinux-build" command 3 times. (Fig. 1.11 and 1.12)

   `$ petalinux-build`

   This may take a few minutes. During this time, the following will occur:

   - Cross-compiling and linking of the Linux kernel (`linux-3.x/*`)

```
<project-root>
    |-.petalinux/
    |-hw-description/
    |-config.project
    |-subsystems/
    |    |-linux/
    |    |    |-config
    |    |    |-hw-description/
    |    |    |-configs/
    |    |    |    |-device-tree/
    |    |    |    |    |-ps.dtsi
    |    |    |    |    |-pl.dtsi
    |    |    |    |    |-system-conf.dtsi
    |    |    |    |    |-system-top.dts
    |    |    |    |-kernel/
    |    |    |    |    |-config
    |    |    |    |-u-boot/
    |    |    |    |    |-config.mk
    |    |    |    |    |-platform-auto.h
    |    |    |    |    |-platform-top.h
    |    |    |    |-rootfs/
    |    |    |        |-config
    |-components/
    |    |-bootloader/
    |    |    |-fs-boot/ | zynq_fsbl/
    |    |-apps/
    |    |    |-myapp/
```

图 1.10: PetaLinux Project Directory

```
[ERROR] E: Sub-process /opt/Xilinx/petalinux-v2014.2-final/tools/packagemanager/
bin/dpkg returned an error code (1)
ERROR: Failed to build linux
```

图 1.11: Building the Linux image

```
petalinux@ubuntu:~/emblnx/labs/lab2/Avnet-Digilent-ZedBoard-2014.2$ petalinux-build
INFO: Checking component...
INFO: Generating make files and build linux
INFO: Generating make files for the subcomponents of linux
INFO: Building linux
[INFO ] pre-build linux/rootfs/fwupgrade
[INFO ] pre-build linux/rootfs/peekpoke
[INFO ] pre-build linux/rootfs/uWeb
[INFO ] build system.dtb
[INFO ] build linux/kernel
[INFO ] update linux/u-boot source
[INFO ] generate linux/u-boot configuration files
[INFO ] build linux/u-boot
[INFO ] Setting up stage config
[INFO ] Setting up rootfs config
[INFO ] Updating for armv7a-vfp-neon
[INFO ] Updating package manager
[INFO ] Expanding stagefs
[INFO ] build linux/rootfs/fwupgrade
[INFO ] build linux/rootfs/peekpoke
[INFO ] build linux/rootfs/uWeb
[INFO ] build kernel in-tree modules
[INFO ] modules linux/kernel
[INFO ] post-build linux/rootfs/fwupgrade
[INFO ] post-build linux/rootfs/peekpoke
[INFO ] post-build linux/rootfs/uWeb
[INFO ] pre-install linux/rootfs/fwupgrade
[INFO ] pre-install linux/rootfs/peekpoke
[INFO ] pre-install linux/rootfs/uWeb
[INFO ] install system.dtb
[INFO ] install linux/kernel
[INFO ] update linux/u-boot source
[INFO ] generate linux/u-boot configuration files
[INFO ] build linux/u-boot
[INFO ] install linux/u-boot
[INFO ] Expanding rootfs
[INFO ] install sys_init
[INFO ] install linux/rootfs/fwupgrade
[INFO ] install linux/rootfs/peekpoke
[INFO ] install linux/rootfs/uWeb
[INFO ] install kernel in-tree modules
[INFO ] modules_install linux/kernel
[INFO ] post-install linux/rootfs/fwupgrade
[INFO ] post-install linux/rootfs/peekpoke
[INFO ] post-install linux/rootfs/uWeb
[INFO ] package rootfs.cpio to /home/petalinux/emblnx/labs/lab2/Avnet-Digilent-ZedBoard-2014.2/images/linux
[INFO ] Update and install vmlinux image
[INFO ] vmlinux linux/kernel
[INFO ] install linux/kernel
[INFO ] package zImage
[INFO ] zImage linux/kernel
[INFO ] install linux/kernel
petalinux@ubuntu:~/emblnx/labs/lab2/Avnet-Digilent-ZedBoard-2014.2$
```

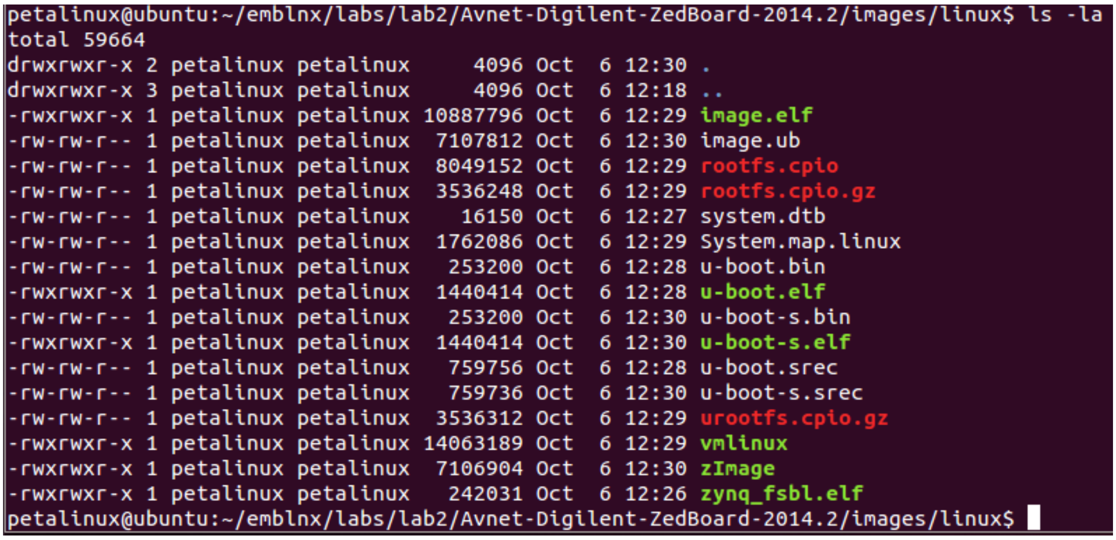
图 1.12: Building the Linux image

- Cross-compiling and linking of the default user libs and applications (`lib/*` and `user/*`)
- Building of a local copy of the ARM Cortex-A9 processor Linux root file system (`romfs/*`)
- Assembling of the kernel and root file system into a single downloadable binary image file (`images/*`)
- Copying of the image files from `images/` in to `/tftpboot`

The build log is saved in the `~/emblnx/labs/lab2/Avnet-Digilent-ZedBoard-2014.2/build.log` file.

2) Once compilation completes, look at the contents in the `images/linux` subdirectory by executing the following commands from the project directory (Fig. 1.13):

```
[host] $ cd images/linux
[host] $ ls -la
```



```
petalinux@ubuntu:~/emblnx/labs/lab2/Avnet-Digilent-ZedBoard-2014.2/images/linux$ ls -la
total 59664
drwxrwxr-x 2 petalinux petalinux     4096 Oct  6 12:30 .
drwxrwxr-x 3 petalinux petalinux     4096 Oct  6 12:18 ..
-rwxrwxr-x 1 petalinux petalinux 10887796 Oct  6 12:29 image.elf
-rw-rw-r-- 1 petalinux petalinux  7107812 Oct  6 12:30 image.ub
-rw-rw-r-- 1 petalinux petalinux  8049152 Oct  6 12:29 rootfs.cpio
-rw-rw-r-- 1 petalinux petalinux  3536248 Oct  6 12:29 rootfs.cpio.gz
-rw-rw-r-- 1 petalinux petalinux    16150 Oct  6 12:27 system.dtb
-rw-rw-r-- 1 petalinux petalinux  1762086 Oct  6 12:29 System.map.linux
-rw-rw-r-- 1 petalinux petalinux   253200 Oct  6 12:28 u-boot.bin
-rwxrwxr-x 1 petalinux petalinux  1440414 Oct  6 12:28 u-boot.elf
-rw-rw-r-- 1 petalinux petalinux   253200 Oct  6 12:30 u-boot-s.bin
-rwxrwxr-x 1 petalinux petalinux  1440414 Oct  6 12:30 u-boot-s.elf
-rw-rw-r-- 1 petalinux petalinux   759756 Oct  6 12:28 u-boot.srec
-rw-rw-r-- 1 petalinux petalinux   759736 Oct  6 12:30 u-boot-s.srec
-rw-rw-r-- 1 petalinux petalinux  3536312 Oct  6 12:29 urootfs.cpio.gz
-rwxrwxr-x 1 petalinux petalinux 14063189 Oct  6 12:29 vmlinux
-rwxrwxr-x 1 petalinux petalinux  7106904 Oct  6 12:30 zImage
-rwxrwxr-x 1 petalinux petalinux   242031 Oct  6 12:26 zynq_fsbl.elf
petalinux@ubuntu:~/emblnx/labs/lab2/Avnet-Digilent-ZedBoard-2014.2/images/linux$
```

图 1.13: Various generated files

3) Examine the contents of the `/tftpboot` directory by executing:

```
[host] $ ls /tftpboot
```

All these files in the `~/emblnx/labs/lab2/Avnet-Digilent-ZedBoard-2014.2/images/linux` directory have a copy in `/tftpboot` because as part of the build process, the image files have also been copied there. The development machine has been configured as a TFTP (trivial FTP) server, allowing the board to pull new kernel images directly over the network from the fixed known location (instead of knowing the actual paths of the project directories). You will use this capability in the next exercise.

Image Name	Descriptions
`image.elf`	Linux image in ELF format
`image.srec`	Linux image in SREC format
`image.ub`	Linux image in U-Boot format
`rootfs.cpio`	Root file system image
`u-boot.bin`	U-Boot image in binary format
`u-boot.srec`	U-Boot image in SREC format
`u-boot.elf`	U-Boot image in ELF format
`u-boot-s.*`	Relocatable U-Boot image

1.2.7 Booting the System

1. As mentioned earlier, you can run Linux for the ARM Cortex-A9 MPcore system on QEMU.

   Load the ARM Cortex-A9 MPcore Linux on QEMU.

   1) Enter the following command in the host Terminal window to load the kernel only (Fig. 1.14~1.17):

      `[host]$ petalinux-boot --qemu --kernel`

   2) Log into the system and explore it as you did in the "A First Look" lab.

      Note: Use `root` as the login name and password.

   3) Exit QEMU by pressing `<Ctrl+a>` then `<x>`.

2. Copy the `BOOT.BIN` file from the pre-built directory to the SD card.

   1) Copy only the `BOOT.BIN` file from the `~/emblnx/labs/lab2/Avnet-Digilent-ZedBoard-2014.2/pre-built/linux/images` directory to the SD card.

   2) Make sure that the board is turned OFF.

   3) Insert the SD card into the target board.

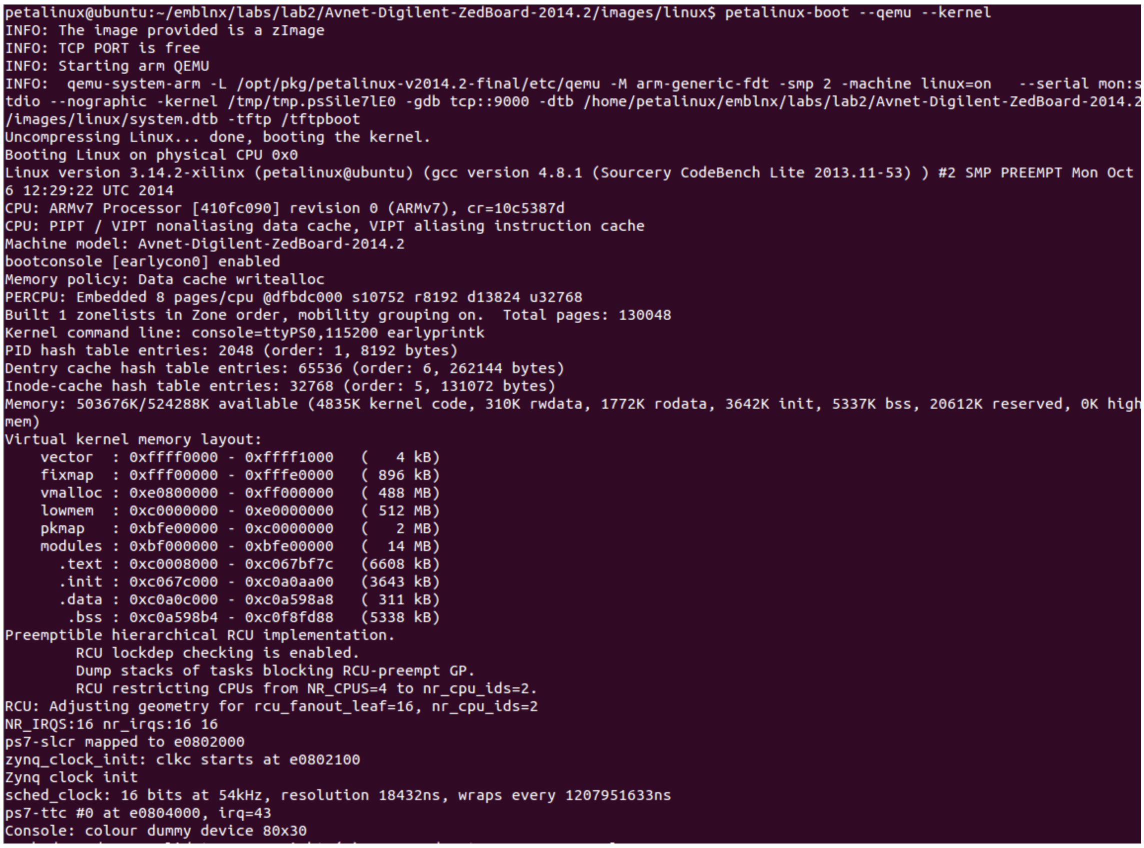
   4) Make sure that the board is set to boot from the SD card.

3. Run the DHCP server on the host.

   1) Run the DHCP server:

      `[host]$ sudo service isc-dhcp-server restart`

4. Power up the board and set the serial port terminal.

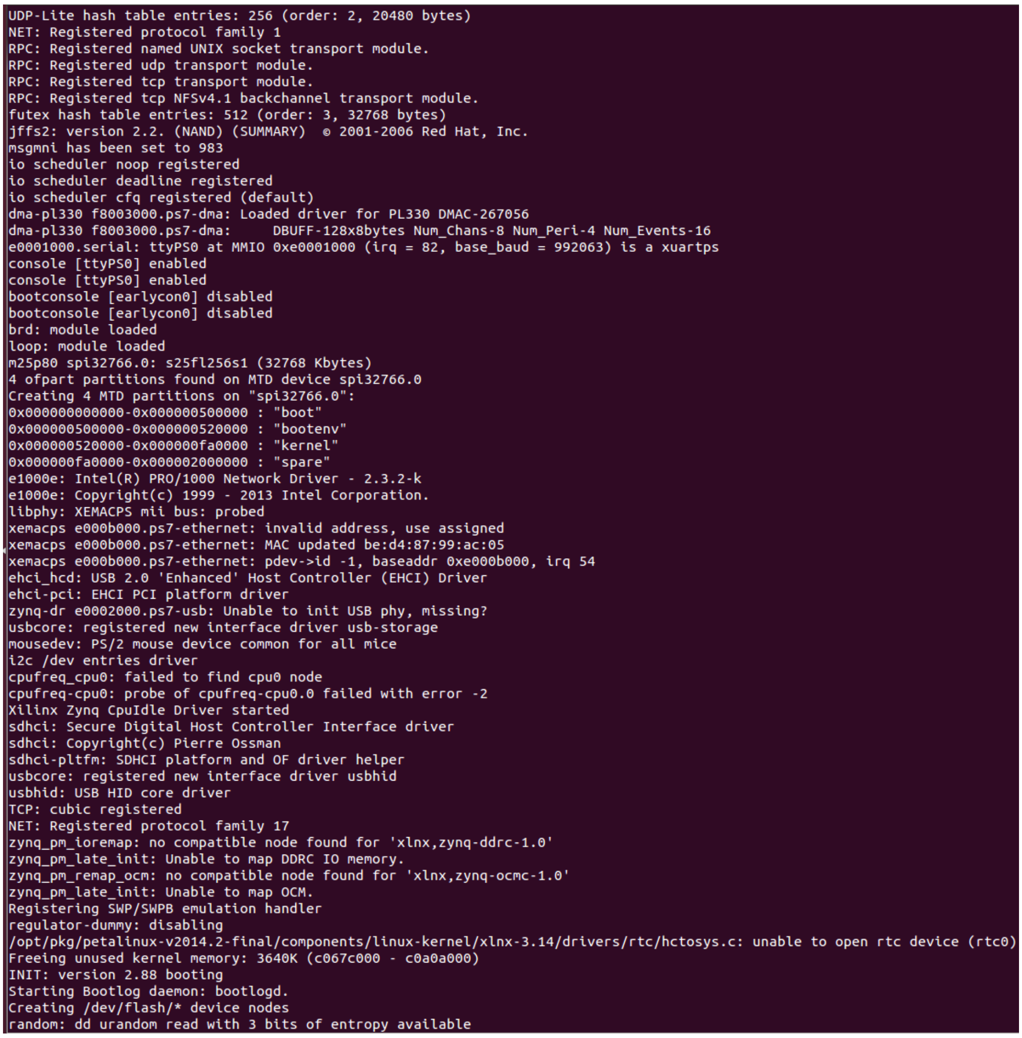


```
petalinux@ubuntu:~/emblnx/labs/lab2/Avnet-Digilent-ZedBoard-2014.2/images/linux$ petalinux-boot --qemu --kernel
INFO: The image provided is a zImage
INFO: TCP PORT is free
INFO: Starting arm QEMU
INFO:  qemu-system-arm -L /opt/pkg/petalinux-v2014.2-final/etc/qemu -M arm-generic-fdt -smp 2 -machine linux=on   --serial mon:s
tdio --nographic -kernel /tmp/tmp.psSile7lE0 -gdb tcp::9000 -dtb /home/petalinux/emblnx/labs/lab2/Avnet-Digilent-ZedBoard-2014.2
/images/linux/system.dtb -tftp /tftpboot
Uncompressing Linux... done, booting the kernel.
Booting Linux on physical CPU 0x0
Linux version 3.14.2-xilinx (petalinux@ubuntu) (gcc version 4.8.1 (Sourcery CodeBench Lite 2013.11-53) ) #2 SMP PREEMPT Mon Oct
6 12:29:22 UTC 2014
CPU: ARMv7 Processor [410fc090] revision 0 (ARMv7), cr=10c5387d
CPU: PIPT / VIPT nonaliasing data cache, VIPT aliasing instruction cache
Machine model: Avnet-Digilent-ZedBoard-2014.2
bootconsole [earlycon0] enabled
Memory policy: Data cache writealloc
PERCPU: Embedded 8 pages/cpu @dfbdc000 s10752 r8192 d13824 u32768
Built 1 zonelists in Zone order, mobility grouping on.  Total pages: 130048
Kernel command line: console=ttyPS0,115200 earlyprintk
PID hash table entries: 2048 (order: 1, 8192 bytes)
Dentry cache hash table entries: 65536 (order: 6, 262144 bytes)
Inode-cache hash table entries: 32768 (order: 5, 131072 bytes)
Memory: 503676K/524288K available (4835K kernel code, 310K rwdata, 1772K rodata, 3642K init, 5337K bss, 20612K reserved, 0K high
mem)
Virtual kernel memory layout:
    vector  : 0xffff0000 - 0xffff1000   (   4 kB)
    fixmap  : 0xfff00000 - 0xfffe0000   ( 896 kB)
    vmalloc : 0xe0800000 - 0xff000000   ( 488 MB)
    lowmem  : 0xc0000000 - 0xe0000000   ( 512 MB)
    pkmap   : 0xbfe00000 - 0xc0000000   (   2 MB)
    modules : 0xbf000000 - 0xbfe00000   (  14 MB)
      .text : 0xc0008000 - 0xc067bf7c   (6608 kB)
      .init : 0xc067c000 - 0xc0a0aa00   (3643 kB)
      .data : 0xc0a0c000 - 0xc0a598a8   ( 311 kB)
       .bss : 0xc0a598b4 - 0xc0f8fd88   (5338 kB)
Preemptible hierarchical RCU implementation.
        RCU lockdep checking is enabled.
        Dump stacks of tasks blocking RCU-preempt GP.
        RCU restricting CPUs from NR_CPUS=4 to nr_cpu_ids=2.
RCU: Adjusting geometry for rcu_fanout_leaf=16, nr_cpu_ids=2
NR_IRQS:16 nr_irqs:16 16
ps7-slcr mapped to e0802000
zynq_clock_init: clkc starts at e0802100
Zynq clock init
sched_clock: 16 bits at 54kHz, resolution 18432ns, wraps every 1207951633ns
ps7-ttc #0 at e0804000, irq=43
Console: colour dummy device 80x30
```

图 1.14: Console output 1

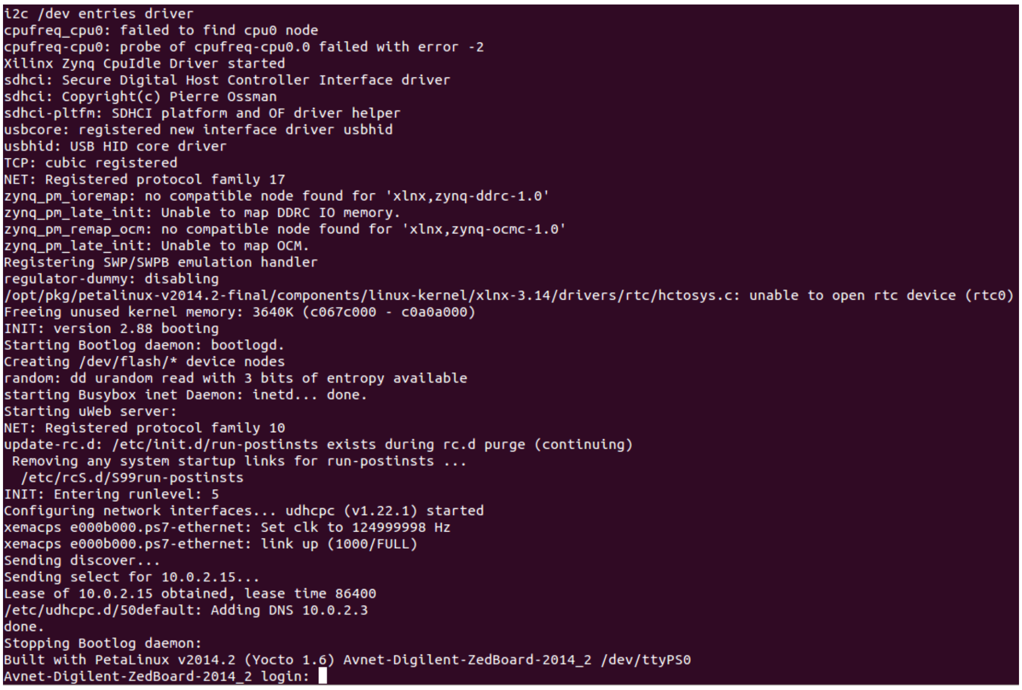
```
Lock dependency validator: Copyright (c) 2006 Red Hat, Inc., Ingo Molnar
... MAX_LOCKDEP_SUBCLASSES:  8
... MAX_LOCK_DEPTH:          48
... MAX_LOCKDEP_KEYS:        8191
... CLASSHASH_SIZE:          4096
... MAX_LOCKDEP_ENTRIES:     16384
... MAX_LOCKDEP_CHAINS:      32768
... CHAINHASH_SIZE:          16384
 memory used by lock dependency info: 3695 kB
 per task-struct memory footprint: 1152 bytes
Calibrating delay loop... 2039.80 BogoMIPS (lpj=10199040)
pid_max: default: 32768 minimum: 301
Mount-cache hash table entries: 1024 (order: 0, 4096 bytes)
Mountpoint-cache hash table entries: 1024 (order: 0, 4096 bytes)
CPU: Testing write buffer coherency: ok
missing device node for CPU 0
missing device node for CPU 1
CPU0: thread -1, cpu 0, socket 0, mpidr 80000000
Setting up static identity map for 0x496b30 - 0x496b88
L2x0 series cache controller enabled
l2x0: 8 ways, CACHE_ID 0x00000000, AUX_CTRL 0x00000000, Cache size: 512 kB
CPU1: Booted secondary processor
CPU1: thread -1, cpu 1, socket 0, mpidr 80000001
Brought up 2 CPUs
SMP: Total of 2 processors activated.
CPU: All CPU(s) started in SVC mode.
devtmpfs: initialized
VFP support v0.3: implementor 41 architecture 3 part 30 variant 9 rev 0
regulator-dummy: no parameters
NET: Registered protocol family 16
DMA: preallocated 256 KiB pool for atomic coherent allocations
cpuidle: using governor ladder
cpuidle: using governor menu
syscon f8000000.ps7-slcr: regmap [mem 0xf8000000-0xf8000fff] registered
hw-breakpoint: debug architecture 0x0 unsupported.
bio: create slab <bio-0> at 0
vgaarb: loaded
SCSI subsystem initialized
usbcore: registered new interface driver usbfs
usbcore: registered new interface driver hub
usbcore: registered new device driver usb
media: Linux media interface: v0.10
Linux video capture interface: v2.00
pps_core: LinuxPPS API ver. 1 registered
pps_core: Software ver. 5.3.6 - Copyright 2005-2007 Rodolfo Giometti <giometti@linux.it>
PTP clock support registered
EDAC MC: Ver: 3.0.0
DMA-API: preallocated 4096 debug entries
DMA-API: debugging enabled by kernel config
Switched to clocksource ttc_clocksource
NET: Registered protocol family 2
TCP established hash table entries: 4096 (order: 2, 16384 bytes)
TCP bind hash table entries: 4096 (order: 5, 147456 bytes)
TCP: Hash tables configured (established 4096 bind 4096)
TCP: reno registered
UDP hash table entries: 256 (order: 2, 20480 bytes)
```

图 1.15: Console output 2



```
UDP-Lite hash table entries: 256 (order: 2, 20480 bytes)
NET: Registered protocol family 1
RPC: Registered named UNIX socket transport module.
RPC: Registered udp transport module.
RPC: Registered tcp transport module.
RPC: Registered tcp NFSv4.1 backchannel transport module.
futex hash table entries: 512 (order: 3, 32768 bytes)
jffs2: version 2.2. (NAND) (SUMMARY)  © 2001-2006 Red Hat, Inc.
msgmni has been set to 983
io scheduler noop registered
io scheduler deadline registered
io scheduler cfq registered (default)
dma-pl330 f8003000.ps7-dma: Loaded driver for PL330 DMAC-267056
dma-pl330 f8003000.ps7-dma:      DBUFF-128x8bytes Num_Chans-8 Num_Peri-4 Num_Events-16
e0001000.serial: ttyPS0 at MMIO 0xe0001000 (irq = 82, base_baud = 992063) is a xuartps
console [ttyPS0] enabled
console [ttyPS0] enabled
bootconsole [earlycon0] disabled
bootconsole [earlycon0] disabled
brd: module loaded
loop: module loaded
m25p80 spi32766.0: s25fl256s1 (32768 Kbytes)
4 ofpart partitions found on MTD device spi32766.0
Creating 4 MTD partitions on "spi32766.0":
0x000000000000-0x000000500000 : "boot"
0x000000500000-0x000000520000 : "bootenv"
0x000000520000-0x000000fa0000 : "kernel"
0x000000fa0000-0x000002000000 : "spare"
e1000e: Intel(R) PRO/1000 Network Driver - 2.3.2-k
e1000e: Copyright(c) 1999 - 2013 Intel Corporation.
libphy: XEMACPS mii bus: probed
xemacps e000b000.ps7-ethernet: invalid address, use assigned
xemacps e000b000.ps7-ethernet: MAC updated be:d4:87:99:ac:05
xemacps e000b000.ps7-ethernet: pdev->id -1, baseaddr 0xe000b000, irq 54
ehci_hcd: USB 2.0 'Enhanced' Host Controller (EHCI) Driver
ehci-pci: EHCI PCI platform driver
zynq-dr e0002000.ps7-usb: Unable to init USB phy, missing?
usbcore: registered new interface driver usb-storage
mousedev: PS/2 mouse device common for all mice
i2c /dev entries driver
cpufreq_cpu0: failed to find cpu0 node
cpufreq-cpu0: probe of cpufreq-cpu0.0 failed with error -2
Xilinx Zynq CpuIdle Driver started
sdhci: Secure Digital Host Controller Interface driver
sdhci: Copyright(c) Pierre Ossman
sdhci-pltfm: SDHCI platform and OF driver helper
usbcore: registered new interface driver usbhid
usbhid: USB HID core driver
TCP: cubic registered
NET: Registered protocol family 17
zynq_pm_ioremap: no compatible node found for 'xlnx,zynq-ddrc-1.0'
zynq_pm_late_init: Unable to map DDRC IO memory.
zynq_pm_remap_ocm: no compatible node found for 'xlnx,zynq-ocmc-1.0'
zynq_pm_late_init: Unable to map OCM.
Registering SWP/SWPB emulation handler
regulator-dummy: disabling
/opt/pkg/petalinux-v2014.2-final/components/linux-kernel/xlnx-3.14/drivers/rtc/hctosys.c: unable to open rtc device (rtc0)
Freeing unused kernel memory: 3640K (c067c000 - c0a0a000)
INIT: version 2.88 booting
Starting Bootlog daemon: bootlogd.
Creating /dev/flash/* device nodes
random: dd urandom read with 3 bits of entropy available
```

图 1.16: Console output 3



```
i2c /dev entries driver
cpufreq_cpu0: failed to find cpu0 node
cpufreq-cpu0: probe of cpufreq-cpu0.0 failed with error -2
Xilinx Zynq CpuIdle Driver started
sdhci: Secure Digital Host Controller Interface driver
sdhci: Copyright(c) Pierre Ossman
sdhci-pltfm: SDHCI platform and OF driver helper
usbcore: registered new interface driver usbhid
usbhid: USB HID core driver
TCP: cubic registered
NET: Registered protocol family 17
zynq_pm_ioremap: no compatible node found for 'xlnx,zynq-ddrc-1.0'
zynq_pm_late_init: Unable to map DDRC IO memory.
zynq_pm_remap_ocm: no compatible node found for 'xlnx,zynq-ocmc-1.0'
zynq_pm_late_init: Unable to map OCM.
Registering SWP/SWPB emulation handler
regulator-dummy: disabling
/opt/pkg/petalinux-v2014.2-final/components/linux-kernel/xlnx-3.14/drivers/rtc/hctosys.c: unable to open rtc device (rtc0)
Freeing unused kernel memory: 3640K (c067c000 - c0a0a000)
INIT: version 2.88 booting
Starting Bootlog daemon: bootlogd.
Creating /dev/flash/* device nodes
random: dd urandom read with 3 bits of entropy available
starting Busybox inet Daemon: inetd... done.
Starting uWeb server:
NET: Registered protocol family 10
update-rc.d: /etc/init.d/run-postinsts exists during rc.d purge (continuing)
 Removing any system startup links for run-postinsts ...
  /etc/rcS.d/S99run-postinsts
INIT: Entering runlevel: 5
Configuring network interfaces... udhcpc (v1.22.1) started
xemacps e000b000.ps7-ethernet: Set clk to 124999998 Hz
xemacps e000b000.ps7-ethernet: link up (1000/FULL)
Sending discover...
Sending select for 10.0.2.15...
Lease of 10.0.2.15 obtained, lease time 86400
/etc/udhcpc.d/50default: Adding DNS 10.0.2.3
done.
Stopping Bootlog daemon:
Built with PetaLinux v2014.2 (Yocto 1.6) Avnet-Digilent-ZedBoard-2014_2 /dev/ttyPS0
Avnet-Digilent-ZedBoard-2014_2 login:
```

图 1.17: Console output 3

1) Power ON the board.
2) Run the following command to make sure that `/dev/ttyACM0` is set to read/write access:

   `[host]$ sudo chmod 666 /dev/ttyACM0`
3) In the dashboard, in the Search field, enter the serial port.
4) Select the **Serial port terminal** application.

5. Boot the new Linux image on the board.

1) Reset the board (BTN7) to see the booting info on the GtkTerm console as the board goes through the boot process.
2) Press any key to stop auto-boot when you see messages similar to the following in the GtkTerm window (Fig. 1.18):
3) If you did not see the "DHCP client bound to address" message during uboot bootup, you will need to run dhcp to obtain the IP address. (Fig. 1.19)

   `U-Boot-PetaLinux> dhcp`
4) Set the TFTP server IP to the host IP by running the following command in the u-boot console:

   `U-Boot-PetaLinux> set serverip 192.168.1.1`
5) Download and boot the new image using TFTP by executing this command in the

```
U-Boot 2014.01 (Jun 10 2014 - 13:33:51)

Memory: ECC disabled
DRAM:   512 MiB
MMC:    zynq_sdhci: 0
SF: Detected S25FL256S_64K with page size 256 Bytes, erase size 64 KiB, total 32 MiB
*** Warning - bad CRC, using default environment

In:     serial
Out:    serial
Err:    serial
Net:    Gem.e000b000
U-BOOT for Avnet-Digilent-ZedBoard-2014_2

Gem.e000b000 Waiting for PHY auto negotiation to complete...... done
BOOTP broadcast 1
DHCP client bound to address 192.168.1.6
Hit any key to stop autoboot:  0
```

图 1.18: Stopping the autoboot

```
Hit any key to stop autoboot:  0
U-Boot-PetaLinux> dhcp
Gem.e000b000:0 is connected to Gem.e000b000.  Reconnecting to Gem.e000b000
Gem.e000b000 Waiting for PHY auto negotiation to complete....... done
BOOTP broadcast 1
DHCP client bound to address 192.168.1.6
```

图 1.19: Running DHCP to obtain the IP address

u-boot console:

`U-Boot-PetaLinux> run netboot`

This command will download the image.ub file from /tftpboot on the host to the main memory of the ARM Cortex-A9 MPcore system and boot the system with the image.

6) Watch the GtkTerm window.

   Messages similar to the following show the image download progress. (Fig. 1.20)

   The `netboot` command will automatically boot the system as soon as the image is finished downloading.

7) Watch the booting messages on the GtkTerm window.

   Other booting messages are the same as from the Lab1 because you used the default configuration.

6. Use `ping` command to test the network connection.

   1) After the system boots, log into the system by entering `root` as both the login name and password.

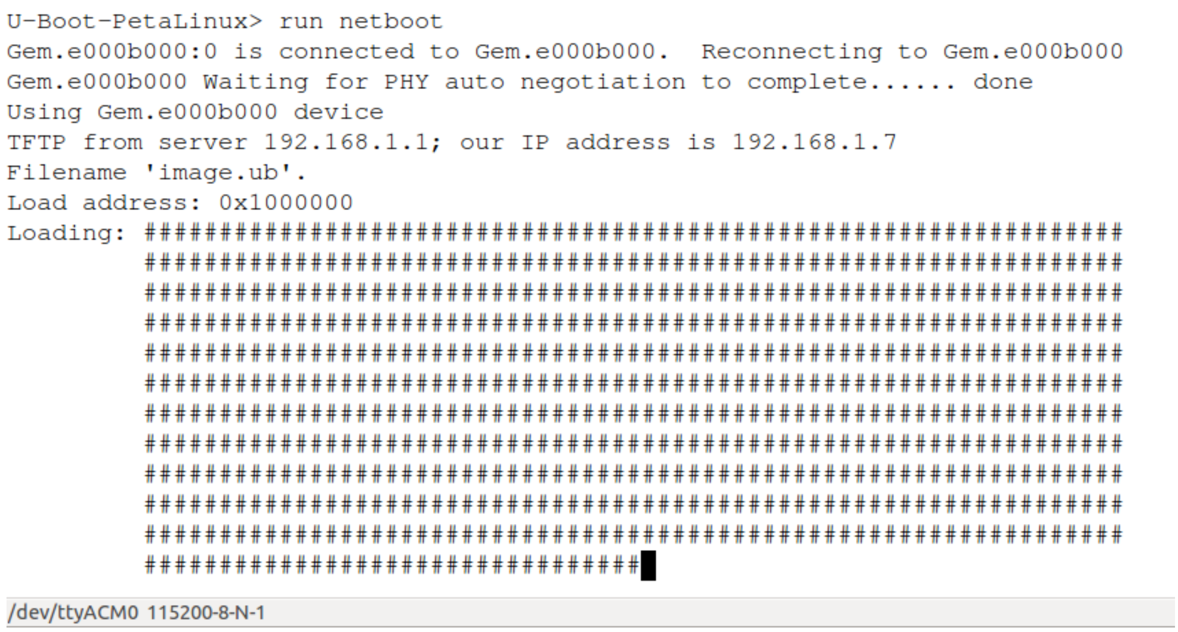


图 1.20: Downloading the built image

2) Execute the `ping` command to ping the host machine.

   `# ping 192.168.1.1`

   You should see the response from the host machine.

3) Execute the `ping` command from the host machine terminal window to see the response from the target board.

   `[host]$ ping 192.168.1.6`

   Use different ip address if the board is bound to different address (see Figure 1.18 to find out the address). You should see the response from the host machine.

7. Soft reboot from Linux.

   1) Run the `reboot` command in the serial terminal window tools to reboot the system:

      `# reboot`

      The system should reboot.

   2) Close the GtkTerm window.

   3) Power off the board.

1.2.8 Conclusion

In this lab, you have learned how to:

- Cross-compile Linux

- Boot Linux for an ARM Cortex-A9 MPcore system in QEMU
- Download a new image to the board via Ethernet

You will use these capabilities in subsequent labs.

1.2.9 Completed Solution

If you want to run the solution then copy `BOOT.bin` from the `labsolution/lab2/SDCard` directory onto a SD card. Place the SD card in the ZedBoard. Set ZedBoard in the SD Card boot mode. Connect the ZedBoard to the host machine using Ethernet cable.

Run the following command to start DHCP server on the host:

```
[host] $ sudo service isc-dhcp-server restart
```

Copy the `image.ub` file from the `labsolution\lab2\tftpboot` directory into `/tftpboot` directory.

Power ON the board. Set the terminal session. Interrupt the boot process when autoboot message is shown. Set the serverip address using the following command in the target board terminal window:

```
#set serverip 192.168.1.1
```

Run the netboot command:

```
#run netboot
```

Login into the system and test the lab.

2 网络和 Linux TCP/IP 编程

2.1 Networking and TCP/IP

2.1.1 Introduction

The ready availability of a complete TCP/IP stack, as well as a wide array of networking applications, is a prime capability that argues in favor of using embedded Linux. This lab will introduce you to embedded Linux networking and demonstrate how it can be useful both during application development and deployment.

In the previous labs, you have already used Linux networking capabilities—the TFTP utility—that pulls the Linux image over the network.

In this lab, you will make more explicit use of the system's networking capabilities, and in particular see how they can be used to dramatically speed up the application building/download/test cycle.

You will also build a web-enabled application that can control some physical I/O on the development board. This will be a fairly simple program, but it hints at something much more powerful.

2.1.2 Objectives

After completing this lab, you will be able to:

- Explore the kernel configuration menu and identify configuration sub-menus that enable Linux TCP/IP networking
- Log in to the ARM Cortex-A9™ processor Linux system by using telnet
- Transfer files to and from Linux by using FTP
- Use the Network File System (NFS) to mount your host file system on the Linux target and Investigate how this capability impacts the cross-development cycle
- Experiment with the embedded web server on the Linux target
- Build and experiment with web-based applications under Linux

2.1.3 Preparation

If this is the first lab that you are performing, then refer to the "Before You Start" section of Lab 1 for necessary preparatory information on how to set up the environment.

If your workstation has been restarted or logout, run the following command to start DHCP server on the host:

```
[host] $ sudo service isc-dhcp-server restart
```

Please refer to the "Initializing the Workshop Environment" section of Lab 1 for detailed information.

2.1.4 Exploring Network Features

The default embedded Linux image on the board supports network applications. If you are interested in Linux settings to enable Ethernet support and the network applications used in this lab, see the Appendix section of this lab.

2.1.5 General Flow for this Lab

- Step 1: Logging In Using Telnet
- Step 2: Transferring Files with FTP
- Step 3: Using NFS
- Step 4: Navigating a Web Page
- step 5: Building the Web-Enabled Application

2.1.6 Logging In Using Telnet Step

In the previous labs, you have logged in to the ARM Cortex-A9 MPcore system by using GtkTerm over a serial line. While this is convenient for debugging and development, it requires a direct serial connection, which may not be available when a system is deployed.

Linux supports the standard telnet protocol directly. In fact, this is already enabled on your ARM Cortex-A9 MPcore.

1. Change the path to the project directory.

   1) Run the following commands to create and change to the project directory path:

   ```
   [host] $ mkdir ~/emblnx/labs/lab4
   [host] $ cd ~/emblnx/labs/lab4
   ```

2. Use the `petalinux-create` command to create a new embedded Linux platform and choose the platform.

1) Run the following command from the `lab4` directory to create a new Petalinux project:

```
[host] $ petalinux-create -t project -s  /opt/pkg/Avnet-
Digilent-ZedBoard-v2014.2-final.bsp
```

The command will create the software project directory: `Avnet-Digilent-ZedBoard-2014.2` under `~/emblnx/labs/lab4`.

2) Change the directory to the PetaLinux project:

```
~/emblnx/labs/lab4/Avnet-Digilent-ZedBoard-2014.2
```

3. Telnet to the ARM Cortex-A9 processor system using QEMU.

1) Run the following command to run the prebuilt ARM Cortex-A9 MPcore Linux in QEMU:

```
[host] $ petalinux-boot --qemu --prebuilt 3 --root --subnet
\ 192.168.10.1/24
```

2) Press y to continue.

3) S2)the IP address of the target board to 192.168.10.2 using the command

```
#ifconfig eth0 192.168.10.2
```

4) Open a new terminal and run the `telnet` command on the host with the IP address noted in the previous step (192.168.10.2 in this case):

```
[host] $ telnet <IP address>
```

Note that the IP address above is the IP address of the virtual ARM Cortex-A9 MPcore system running under QEMU.

Fig. 2.1 is the output in the telnet console on the host:

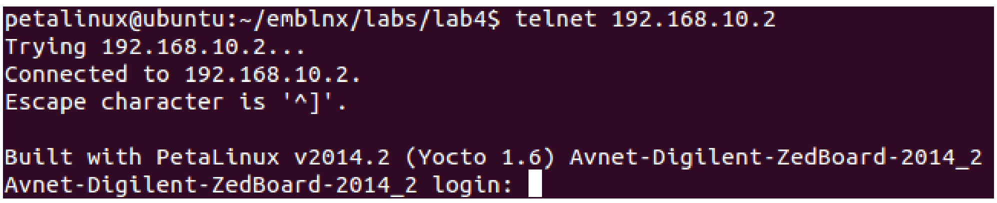


图 2.1: Telnet console on the host

5) Log in using `root` as the login id and password.

6) Try some Linux commands on the telnet console, such as `ls` or `pwd`, for example.

7) Enter `exit` to quit the `telnet` program.

2.1.7 Transferring Files with FTP

FTP is another frequently used network feature. Your ARM Cortex-A9 MPcore Linux system is also pre-configured with an FTP server.

1. Launch the FTP application and experiment with its different functionalities.
   1) Launch the FTP application from your host by executing:

      ```
      [host] $ ftp 192.168.10.2
      Connected to 192.168.10.2.
      220 Operation successful

      Name (192.168.10.2:petalinux):
      ```

   2) Press <Enter> at the name prompt when you see messages similar to the following:

      ```
      230 Operation successful
      Name (192.168.10.2:petalinux):
      230 Operation successful
      Remote system type is UNIX.
      Using binary mode to transfer files.
      ```

      You should now be able to see the FTP prompt:

      `ftp>`

      You can now transfer files to and from the ARM Cortex-A9 MPcore system. If you are sending files to the ARM Cortex-A9 MPcore system, the home directory of FTP in the ARM Cortex-A9 MPcore system is `/var/ftp`. You can `get` and `put` files to that directory only.

   3) Enter `bye` to quit ftp.
   4) Close the terminal.

2.1.8 Using NFS

Network File System (NFS) is a long-supported capability of Linux (and thus embedded Linux). It allows a remote file system to be mounted over the network and used as though it were physically on the local host. In the context of cross-compiled embedded Linux systems, this can be invaluable.

NFS is very useful when you are debugging your application. Instead of rebuilding and downloading an entire image every time you make a change to your application, you can simply mount your development directory onto the ARM Cortex-A9 MPcore system. When you recompile your application, the new version is immediately available to run on the target.

1. Determine the LiveUSB partitions names and mount the second partition.

    1) In the dashboard, enter **Disk**.

    2) Select **Disk Utility**.

    3) Select the **LiveUSB** device.

    4) Select the 2nd partition and note its name. In the figure below it shows **casper-rw** partition.

    5) Click **Mount Volume**. (Fig. 2.2 and 2.3)

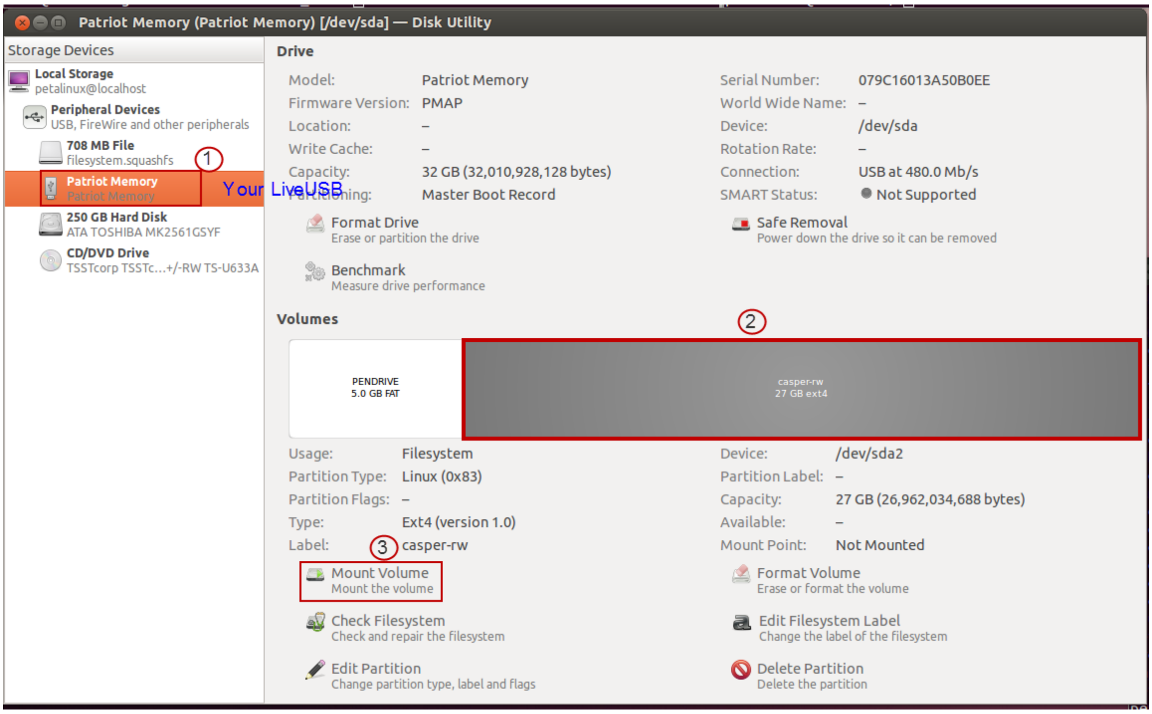


图 2.2: Determining the LiveUSB device's partition names and mounting the 2nd partition

    After mounting, you should see the mount point as `/media/casper-rw`.

    6) Close the Disk Utility application.

2. To allow your ARM Cortex-A9 MPcore system to mount a remote file system from your host, the host must be configured to allow it. This is specified in the `/etc/exports` file. Verify that the host is properly configured.

    1) Open a new terminal.

    2) Enter the following command:

    ```
    [host] $ df
    ```

    Note: Observe that `/dev/sdb2` (in this case) is mounted as `/media/casper-rw`

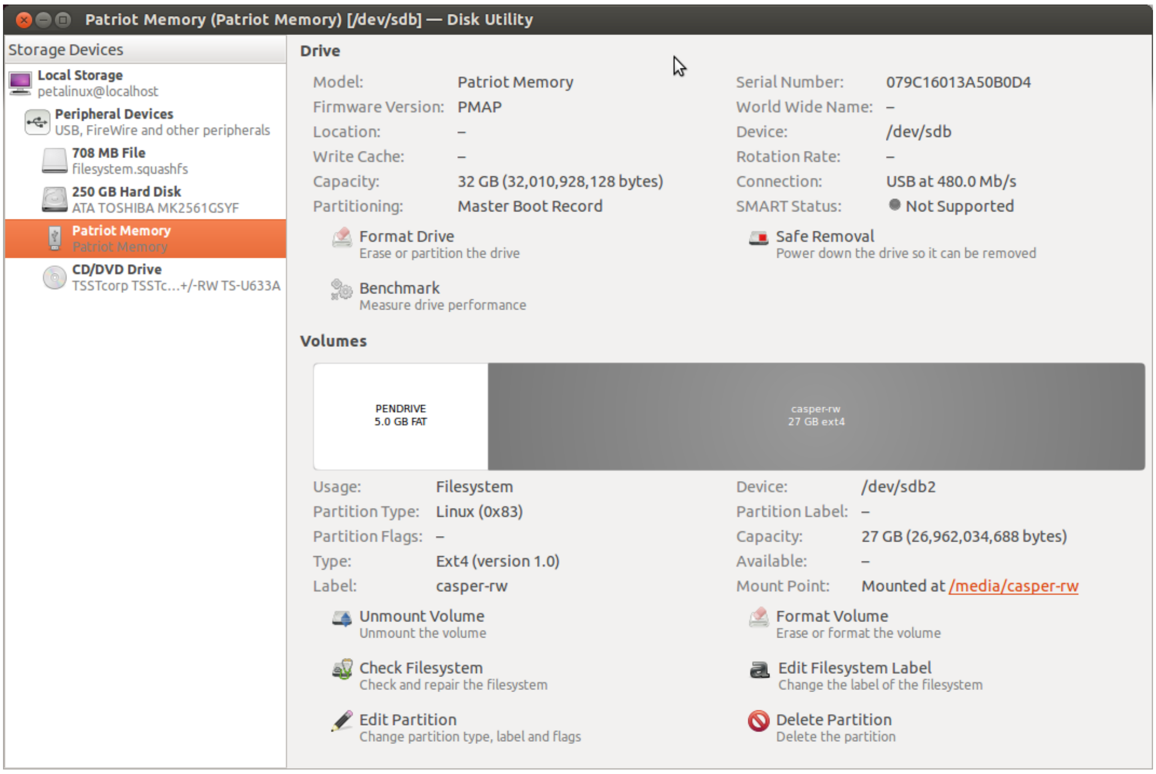


图 2.3: casper-rw mounted as `/media/casper-rw`

on the host machine. This may be different for your system.

3) Examine the contents of the `/etc/exports` file by executing:

   `[host] $ cat /etc/exports`

4) Find the following line in the `/etc/exports`

```
/home/petalinux 192.168.*.* (rw,sync,no_root_squash,no_subtree_check)
```

This says that the directory `/home/petalinux` can be exported to the machine with IP address `192.168.*.*` (IP address from 192.168.0.1 to 192.168.255.255) and that it can be mounted with read-write permission.

However note that you do not have `/home/petalinux` mounted

Because you have `/media/casper-rw` mounted, edit the `/etc/exports` file (you will have to use `sudo` command) and change the line to read as:

```
/media/casper-rw/home/petalinux 192.168.*.*
(rw,sync,no_root_squash,no_subtree_check)
```

This says that the directory `/media/casper-rw/home/petalinux` can be exported to the machine with IP address `192.168.*.*` (IP address from 192.168.0.1 to 192.168.255.255) and that it can be mounted with read-write permission.

5) Restart the NFS server on host:

   `[host] $ sudo /etc/init.d/nfs-kernel-server restart`

   This command will stop running the NFS service if there is an NFS service running and then restart it.

   The following is the output on the host from this command:

```
Stopping NFS kernel daemon [ OK ]
Unexporting directories for NFS kernal daemon··· [ OK ]
Exporting directories for NFS kernel daemon··· [ OK ]
Starting NFS kernel daemon: [ OK ]
```

   If you want to change the shared folder, you should:

   - Edit the `/etc/exports` file
   - Restart the NFS server by running:

     `[host] $ sudo /etc/init.d/nfs-kernel-server restart`

   Now, the host allows your ARM Cortex-A9 MPcore system to NFS mount to its `/home/petalinux` directory.

3. Scroll the QEMU console back and take a closer look at the bootup output.

   You should see when the network device driver is initialized, when the Linux networking stack is configured, and, towards the end, when the `portmap` application is run. This `portmap` application is required for NFS mount.

   Mount the file system on the desktop PC on the ARM Cortex-A9 MPcore system

   1) Log in to the QEMU system.
   2) Run the following command in the QEMU console:

```
# mount -o port=2049,nolock,proto=tcp -t nfs \
192.168.10.1:/media/casper-rw/home/petalinux /mnt
```

   This command tells `mount` that:

   - You want to mount a file system of NFS type (`-t nfs`).
   - The host of this file system has IP address 192.168.10.1.
   - The directory on the host that you want to mount is `/home/petalinux` (that is, your home directory).
   - You want this file system to be mounted underneath the local `/mnt` directory (this is known as the "mount point").

4. Change into the `/mnt` directory on the ARM Cortex-A9 system.

Experiment with making changes to the `myapp` application that you used in the earlier lab. For example, change `printf("Hello, Petalinux World!\n")` to `printf("Hello, Welcome to the Xilinx workshop!\n")`. Rebuild it on the host and run it again on the ARM Cortex-A9 MPcore system over the NFS mount.

1) Execute the following:

```
# cd /mnt
# ls
...
```

   Does it all seem strangely familiar? It should—it is the home directory on your desktop machine. You have read/write access, so be careful. Deleting a file on this mounted NFS drive means that it is deleted from your desktop, and vice versa.

2) To see how NFS mounting can be useful on your host machine, return to the `myapp` application from the earlier lab by executing the following command in the Gtk-Term window:

```
# cd /mnt/emblnx/labs/lab3/Avnet-Digilent-ZedBoard-2014.2/build/\
linux/rootfs/apps/myapp
```

3) Run the hello application directly over the network by running:
   `# ./myapp`
4) Open a new terminal.
5) Change to the `myapp` directory:
   `[host]$ cd ~/emblnx/labs/lab3/Avnet-Digilent-ZedBoard-2014.2/components/apps/myapp`
6) Try making some changes to the `myapp.c` file (to the `printf` statement, for example).
   `[host] $ gedit myapp.c`
7) Change the first `printf` statement to `printf("Hello, Welcome to the XUP workshop!\n")`.
8) Change to the PetaLinux project directory.
   `[host]$ cd ~/emblnx/labs/lab3/Avnet-Digilent-ZedBoard-2014.2`
9) Build the application only.

```
[host] $ petalinux-build -c rootfs/myapp -x clean
[host] $ petalinux-build -c rootfs/myapp
```

10) Run `myapp` again on the ARM Cortex-A9 MPcore system by running the following command:

    ```
    # ./myapp
    ```

    The output of the application should reflect the changes.

    Any changes made on the host to the application can be tested on the ARM Cortex-A9 MPcore system immediately over the NFS mount.

2.1.9 Navigating the Web Page on HTTP

More and more embedded systems and applications are becoming web-enabled, allowing for remote control, management, and monitoring. In this step, you will experiment with the PetaLinux uWeb demo and httpd

1. Launch a web browser on the host machine and explore the default placeholder page that is installed on the ARM Cortex-A9 MPcore Linux system.

    1) Exit the existing QEMU Linux by pressing `<Ctrl-a>` and then `<x>` and restarting QEMU by running the following command:

    ```
    [host] $ petalinux-boot --qemu --prebuilt 3 --kernel --
    qemu-args "-redir tcp:10080:10.0.2.15:80"
    ```

    At the bottom of the boot-up messages, you can see the uWeb server has been started during boot.

2. The web demo self-contains the uWEB server. There is another httpd server built into the ARM Cortex-A9 MPcore Linux system, which is a BusyBox `httpd` server.

    In the rest of this lab, you will try this BusyBox `httpd` server and experiment with a simple CGI application.

    Log in to the ARM Cortex-A9 MPcore console and start the `httpd` server.

    1) Log in to the system.

    2) Run the following command:

    ```
    # httpd -p 8080 -h /home/httpd
    ```

    The above command binds the `httpd` server to port 8080 and uses `/home/httpd` as the `httpd` home directory.

    3) On your host machine, change the URL in your web browser to:

    ```
    http://localhost:10080
    ```

    This time, you will see a home page. This home page is located in `/home/httpd` in the ARM Cortex-A9 MPcore Linux system.

4) Explore the `httpd` home directory by running the following command in the ARM Cortex-A9 MPcore Linux:

```
# ls /home/httpd
```

The directory should list:

```
cgi-bin css img javascript source
```

The `cgi-bin/` directory is for CGI applications.

5) Press `<Ctrl+a>` and then `<x>` to shut down QEMU.

2.1.10 Building the Web-Enabled Application

Web serving embedded applications becomes a lot more useful when the web interface can be used to control the device,or monitor sensor inputs. In this step, you will build and experiment with a simple web-enabled application on the ARM Cortex-A9 MPcore system.

This step will be performed on the hardware board, not QEMU.

1. In this step you will build a web-enabled application. A sample CGI application to control the on/off of the LEDs on the board is provided.

   Build this program and run it step by step.

   1) Make sure that you are in the PetaLinux project location; i.e., `~/emblnx/labs/lab4/Avnet-Digilent-ZedBoard-2014.2`.

   2) Enter the following command to create a new user application inside the PetaLinux project:

   ```
   [host] $ petalinux-create -t apps --name cgi-leds
   ```

   The new application you have created can be found in the `<project-root>/components/apps/cgi-leds` directory, where `<project-root>` is `~/emblnx/labs/lab4/Avnet-Digilent-ZedBoard-2014.2`.

2. Copy the cgi-leds source from the `sources/lab4/cgi-leds` directory.

   1) Change to the newly created application directory:

   ```
   [host] $ cd <project-root>/components/apps/cgi-leds
   ```

   2) Copy the `cgi-leds` application related files from the `sources/lab4/cgi-leds` directory:

   ```
   [host] $ cp ~/emblnx/sources/lab4/cgi-leds/* ./
   ```

   The main application is composed of `cgi_leds.c`, `led.cgi.c`, and `led-gpio.c`. The other files are for a small CGI library. You can find them in the `cgi-leds` project. If you open the `Makefile`, you will notice that the target application name is set to `led.cgi`.

3. Select the new application to be included in the build process. The application is not enabled by default.

   1) Make sure that you are in the project directory; i.e., `~/emblnx/labs/lab4/Avnet-Digilent-ZedBoard-2014.2`.
   2) Launch the rootfs configuration menu by entering the following command:
      `[host] $ petalinux-config -c rootfs`
   3) Press the Down Arrow key to scroll down the menu to **Apps**.
   4) Press `<Enter>` to go into the Apps sub-menu.
      The new application **cgi-leds** is listed in the menu.
   5) Scroll to **cgi-leds** and press `<Y>` to select the application.
   6) Exit the menu and select `<Yes>` to save the new configuration.
      It will take a few seconds for the configuration changes to be applied. Wait until you return to the shell prompt on the command console.

4. Build the image.

   1) Enter the following command to build the image:
      `[host] $ petalinux-build`
      Let the build process to complete and the image be created.

5. Make sure that the `BOOT.BIN` file located in SD card is copied from the pre-built directory.

   1) Make sure that the pre-built `BOOT.BIN` file is located in the SD card.
      If you have performed the "Build and Boot an Image" lab or "Application Development and Debugging" lab as your last lab, there is no need to perform any changes to the SD card.
   2) If not, copy the `BOOT.BIN` file from the `~/emblnx/sources/lab1/SDCard` directory to the SD card.

6. To download the image, run the `DHCP` server on the host.

   1) Run the `DHCP` server:
      `[host] $ sudo service isc-dhcp-server restart`

7. Power up the board and set the serial port terminal.

   1) Power ON the board.

2) Ensure that `/dev/ttyACM0` is set to read/write access:

   `# sudo chmod 666 /dev/ttyACM0`

3) In the dashboard, in the Search field, enter the serial port.

4) Select the **Serial port terminal** application.

   You can reset the board (BTN7) to see the booting info once again.

8. Boot the new embedded Linux image over the network.

   1) Watch the booting process in the GtkTerm window.

   2) Press any key to stop auto-boot when you see the autoboot message in the GtkTerm window.

      If you did not see the "DHCP client bound to address" message during uboot bootup, you will need to run `dhcp` to obtain the IP address:

      `U-Boot-PetaLinux> dhcp`

   3) Set the TFTP server IP to the host IP by running the following command in the u-boot console:

      `U-Boot-PetaLinux> set serverip 192.168.1.1`

   4) Download and boot the new image using TFTP by executing this command in the u-boot console:

      `U-Boot-PetaLinux> run netboot`

      This command will download the `image.ub` file from `/tftpboot` on the host to the main memory of the ARM Cortex-A9 MPcore system and boot the system with the image.

   5) Watch the GtkTerm window.

9. Run the `led.cgi` program.

   1) Once the board reboots, log in and start the `httpd` service:

      `# httpd -p 8080 -h /home/httpd`

   2) Point the web browser on the host back to the board:

      `http://<IP of the board>`

      The IP address of the board will be shown in the end of the boot messages.

      For example:

      ```
      Sending select for 192.168.1.5...
      Lease of 192.168.1.5 obtained, lease time 864000
      ```

      From the above messages, you can see that the board's IP is assigned as 192.168.1.5.

Again, the index page will display.

3) Modify the URL to include the path to the new `led.cgi` application (Fig. 2.4):

```
http://<IP of the board>:8080/cgi-bin/led.cgi
```

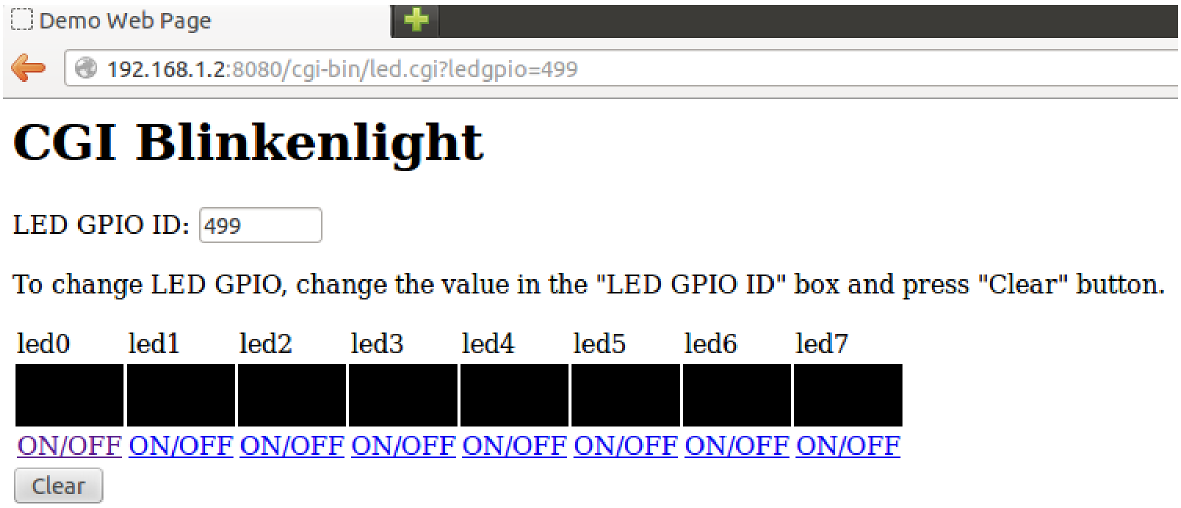


图 2.4: `led.cgi` application

4) Enter the following command to display the ID numbers of the various available GPIOs:

```
# ls /sys/class/gpio
```

Note that ID number 243 corresponds to the LEDs. The ID number may vary depending on which SD card image you have used.

5) In the browser, enter **243** in the LED GPIO ID field.

6) Click **ON/OFF** in the web page and watch what happens on the board and the web page. (Fig. 2.5)

7) Click clear button to turn OFF all the LEDs.

8) Once you are done, power off the board.

2.1.11 Conclusion

In this lab, you have learned how to:

- Use `telnet` to log in to the Linux system
- Use `ftp` to transfer files
- Use NFS to mount your development system onto the Linux target
- Execute a Linux application directly over the NFS mount, instead of updating and down-

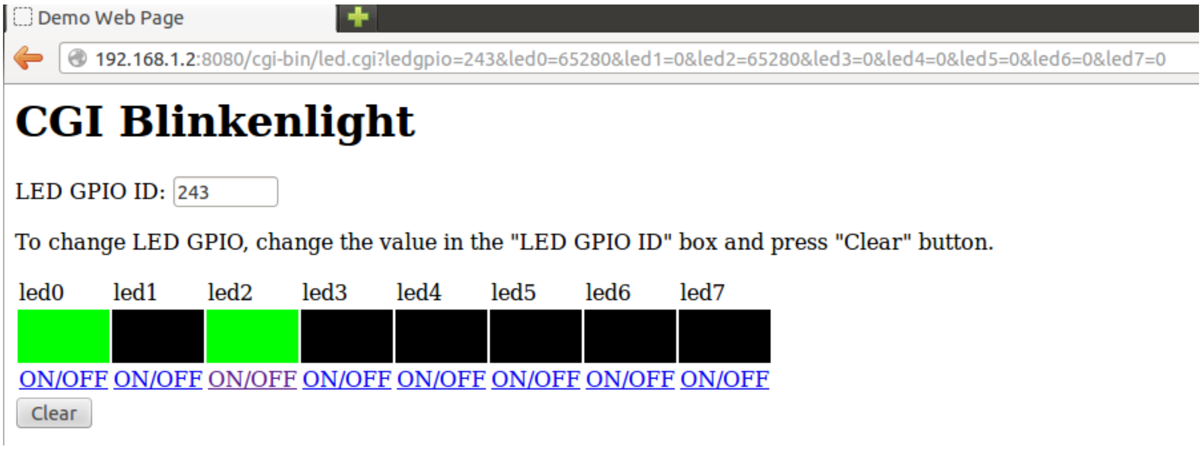


图 2.5: Providing the LED GPIO ID and turning ON/OFF the LEDs

loading an entirely new image file

- Create and modify simple static HTML pages so that they can be served by the embedded web server
- Describe how simple web-enabled applications run on the Linux target

2.1.12 Completed Solution

If you want to run the solution then copy `BOOT.bin` from the `labsolution/lab4/SDCard` directory onto a SD card. Place the SD card in the ZedBoard. Set ZedBoard in the SD Card boot mode. Connect the ZedBoard to the host machine using Ethernet cable.

Run the following command to start DHCP server on the host:

```
[host] $ sudo service isc-dhcp-server restart
```

Copy the `image.ub` file from the `labsolution/lab4/tftpboot` directory into `/tftpboot` directory.

Power ON the board. Set the terminal session. Interrupt the boot process when autoboot message is shown. Set the serverip address using the following command in the target board terminal window:

```
#set serverip 192.168.1.1
```

Run the netboot command:

```
#run netboot
```

Login into the system and test the lab.

2.2 Linux 网络编程实验

2.2.1 实验目的

- 深入理解 TCP/IP 协议模型
- 熟悉并学会 Linux 的 Socket 套接字编程方法
- 熟悉使用 TCP 协议编程的基本过程
- 熟练掌握基于客户/服务器模式应用程序的编写方法

2.2.2 实验原理

- 本程序通过建立基于 TCP 协议的服务器与客户端的通讯，使客户端向服务器发送字符串，服务器将接收到的字符串打印出来。
- 服务器代码流程图如图 2.6 所示。

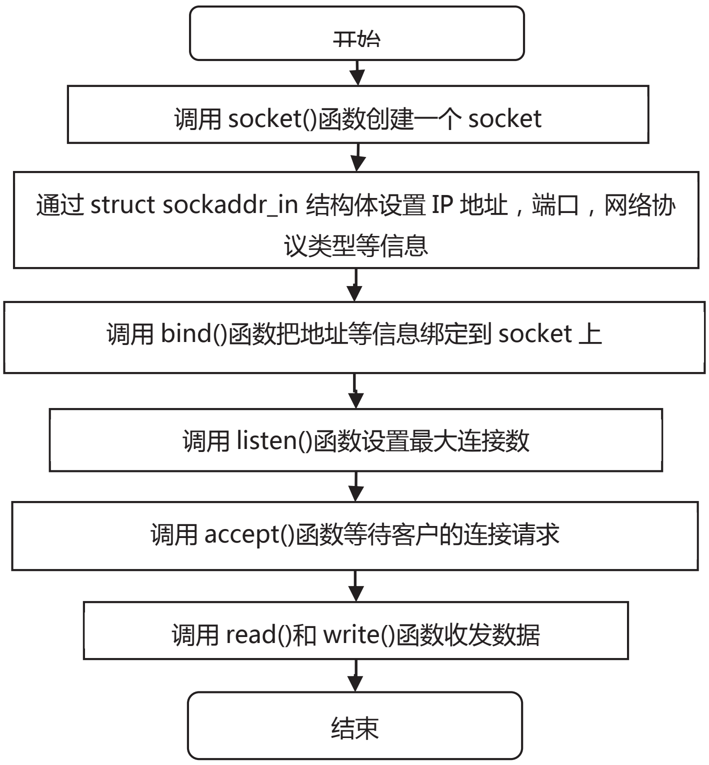


图 2.6: 服务器代码流程图

- 客户端代码流程图如图 2.7 所示

2.2.3 步骤与现象

- 打开终端并进入实验代码目录，即输入以下命令：

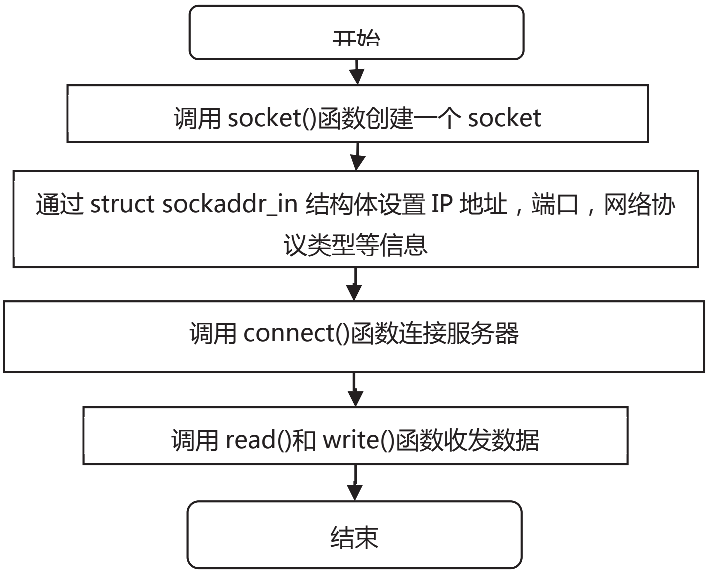
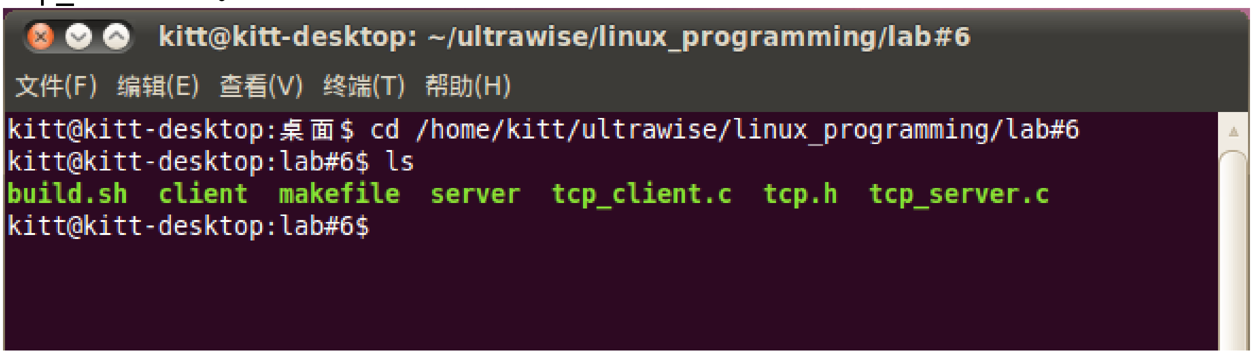


图 2.7: 客户端代码流程图

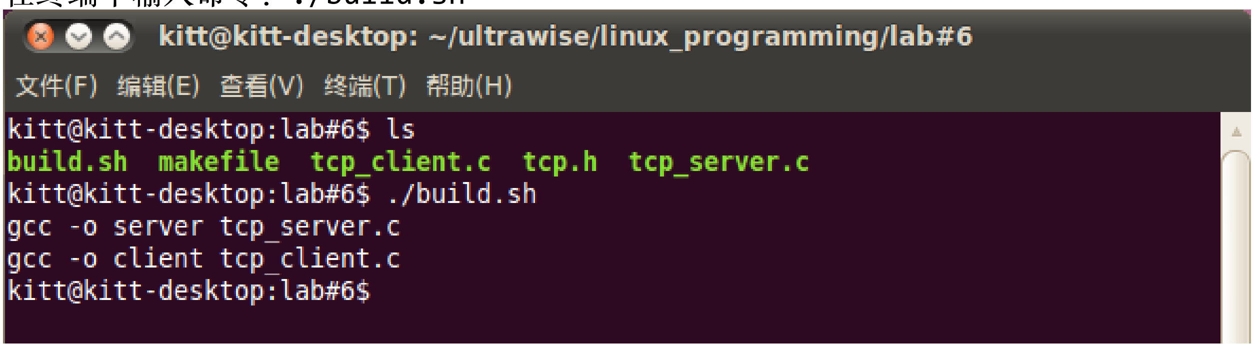
cd /home/zynq/linux_programming/lab#6

如下图所示，进入目录后可以看见实验参考代码 tcp_client.c、tcp.h 和 tcp_server.c。

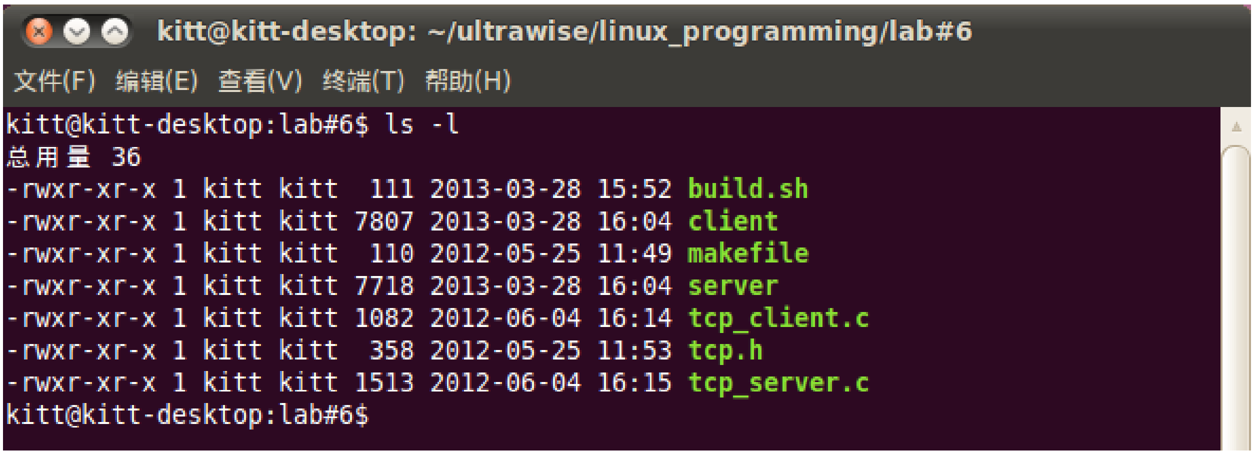


- 编译应用程序

在终端中输入命令：./build.sh

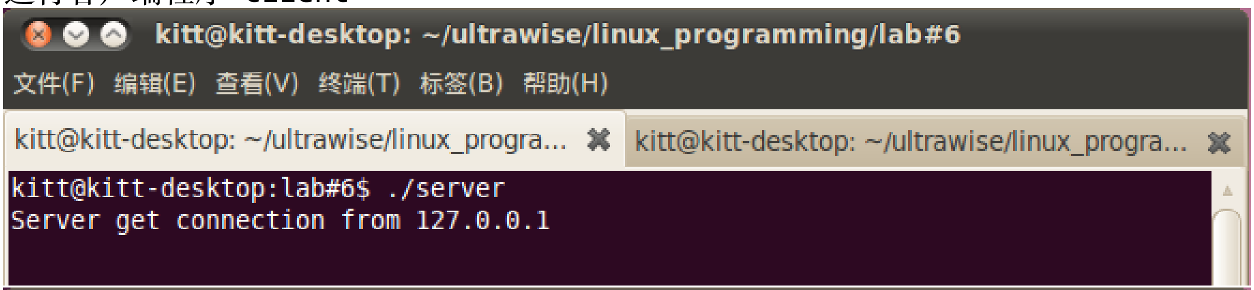


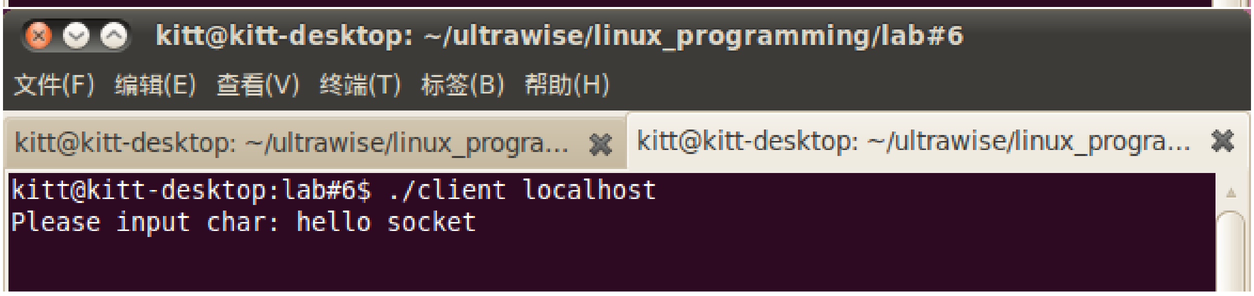
用 gcc 命令编译 tcp_client.c 和 tcp_server.c 成可执行文件 client 与 server



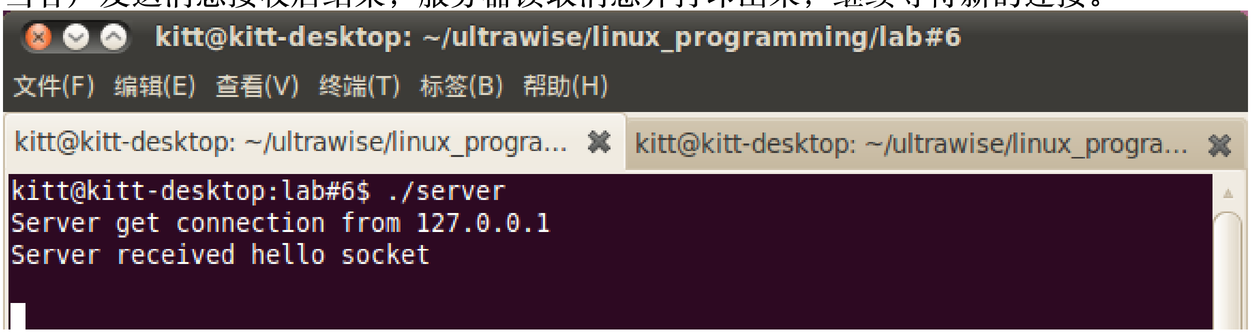
- 运行应用程序

  打开两个终端，先在一个终端中运行服务器程序 `server`，然后在另一个终端中运行客户端程序 `client`





从运行情况可以看出，在没有客户连接上来时服务器程序阻塞在 `accept` 函数上，等待连接。当有客户程序连接上来时，阻塞在 `read` 函数上，等待读取消息。当客户发送消息接收后结束，服务器读取消息并打印出来，继续等待新的连接。



2.2.4 关键代码分析

- 服务器代码分析

```
int main(int argc,char**argv)
{
        int sockfd,client_fd;
        int addr_len = sizeof(struct sockaddr_in);
        struct sockaddr_in server_addr;
        struct sockaddr_in client_addr;
        char buf[BUF_SIZE];
        int bytes;
        //服务器端开始建立 socket 描述符
        if((sockfd=socket(AF_INET,SOCK_STREAM,0))==-1)  {
                fprintf(stderr,"socket error:%s\n",strerror(errno));
                exit(1);
        }
        //服务器端填充 sockaddr 结构体
        bzero(&server_addr,addr_len);//初始化，置 0
        server_addr.sin_family = AF_INET;//IPV4 网络协议
        server_addr.sin_port = htons(SERVER_PORT);//设置端口号
        //设置服务器运行在和 IP 的主机上
        server_addr.sin_addr.s_addr = htonl(INADDR_ANY);
        //绑定 socket 描述符到 IP 地址
        if(bind(sockfd,(struct sockaddr*)&server_addr,addr_len)==-1)  {
                fprintf(stderr,"bind error:%s\n",strerror(errno));
                exit(1);
        }
        //设置运行连接的最大客户端数
        if(listen(sockfd,5)==-1){
                fprintf(stderr,"listen error:%s\n",strerror(errno));
                exit(1);
        }

        while(1)
      {
                //服务器阻塞，直到客户程序建立连接
                if((client_fd = accept(sockfd, (struct sockaddr*)&client_addr,
                    &addr_len))==-1) {
                        fprintf(stderr,"accept error:%s\n",strerror(errno));
                        exit(1);
                }
                //将网络地址转化成字符串并打印出来
                fprintf(stderr, "Server get connection from %s\n",
```

```
                inet_ntoa(client_addr.sin_addr));
        if((bytes = read(client_fd,buf,BUF_SIZE))==-1) {
                fprintf(stderr,"read error:%s\n",strerror(errno));
                exit(1);
        }
        //在接收字符的最后加上终止符
        buf[bytes]='\0';
        printf("Server received %s\n",buf);
        //关闭这个通讯连接
        close(client_fd);
    }
    close(sockfd);
    return 0;
}
```

- 客户端代码分析

```
int main(int argc,char**argv)
       {
        int sockfd;
        int addr_len = sizeof(struct sockaddr_in);
        struct sockaddr_in client;
        struct hostent *host;
        char buf[BUF_SIZE];
        int bytes;
        if(argc!=2)  {
                fprintf(stderr,"Usage:%s hostname \a\n",argv[0]);
                exit(1);
        }
        //使用 hostname 查询主机的名字
        if((host=gethostbyname(argv[1]))==NULL)  {
                fprintf(stderr,"Gethostname error\n");
                exit(1);
        }
        //客户程序开始建立 socket 描述符
        if((sockfd=socket(AF_INET,SOCK_STREAM,0))==-1)  {
                fprintf(stderr,"socket error:%s\n",strerror(errno));
                exit(1);
        }
        //客户端程序填充服务器的资料
        bzero(&client,addr_len);//初始化，置 0
        client.sin_family=AF_INET;//IPV4 网络协议
        client.sin_port=htons(SERVER_PORT);//设置端口号
        client.sin_addr=*((struct in_addr *)host->h_addr);//设置 IP 地址
```

```
        //客户程序发起连接请求
        if(connect(sockfd,(struct sockaddr*)&client,addr_len)==-1)  {
                fprintf(stderr,"connect error:%s\n",strerror(errno));
                exit(1);
        }
        //连接成功
        printf("Please input char: ");
        //发送数据
        fgets(buf,BUF_SIZE,stdin);
        write(sockfd,buf,strlen(buf));
        //结束通讯
        close(sockfd);
        return 0;
}
```