Lab 4: Networking and TCP/IP

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.

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

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.

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.

General Flow for this Lab

Logging In Using Telnet Step 1

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.

1-1. Change the path to the project directory.

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

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

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

1-2. Use the `petalinux-create` command to create a new embedded Linux platform and choose the platform.

1-2-1. 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`.

1-2-2. Change the directory to the PetaLinux project:

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

1-3. Telnet to the ARM Cortex-A9 processor system using QEMU.

1-3-1. 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
```

1-3-2. Press **y** to continue.

1-3-3. Set the IP address of the target board to 192.168.10.2 using the command

```
#ifconfig eth0 192.168.10.2
```

1-3-4. 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.

The following is the output in the `telnet` console on the host:

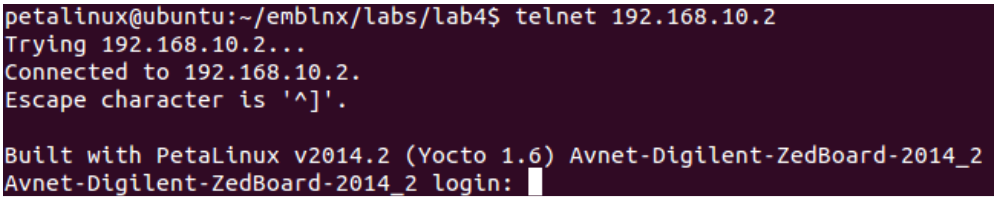

Figure 1. Telnet console on the host

1-3-5. Log in using `root` as the login id and password.

1-3-6. Try some Linux commands on the telnet console, such as `ls` or `pwd`, for example.

1-3-7. Enter `exit` to quit the `telnet` program.

Transferring Files with FTP — Step 2

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

2-1. Launch the FTP application and experiment with its different functionalities.

2-1-1. 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):
```

2-1-2. 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.

2-1-3. Enter **bye** to quit ftp.

2-1-4. Close the terminal.

Using NFS Step 3

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.

3-1. Determine the LiveUSB partitions names and mount the second partition.

3-1-1. In the dashboard, enter **Disk**.

3-1-2. Select **Disk Utility**.

3-1-3. Select the **LiveUSB** device.

3-1-4. Select the 2nd partition and note its name. In the figure below it shows **casper-rw** partition.

3-1-5. Click **Mount Volume**.

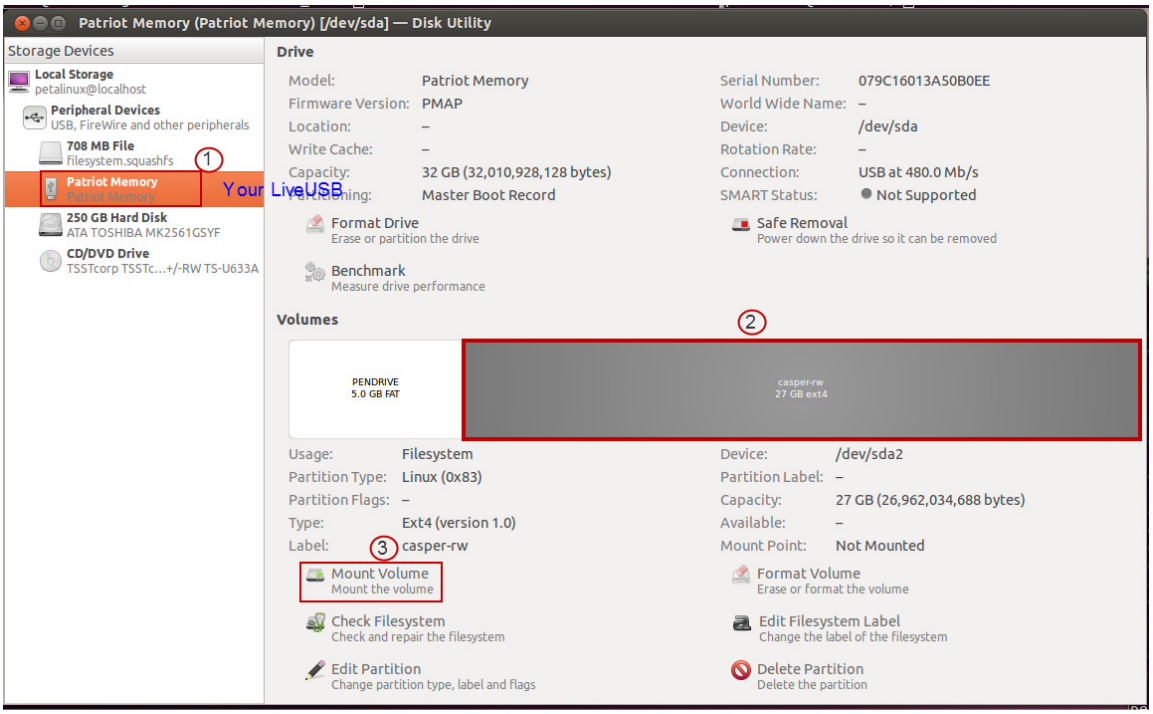

Figure 2. Determining the LiveUSB device's partition names and mounting the 2nd partition

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

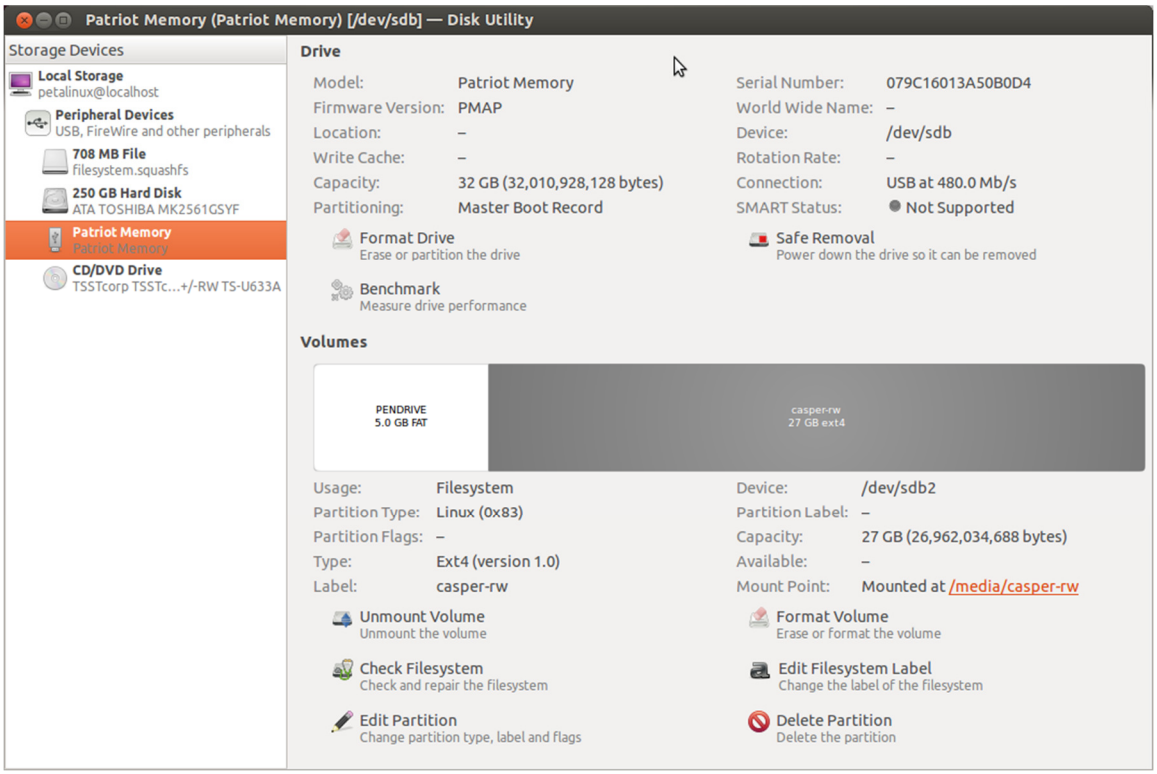

Figure 3. casper-rw mounted as /media/casper-rw

3-1-6. Close the Disk Utility application.

3-2. 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.

3-2-1. Open a new terminal.

3-2-2. Enter the following command:

```
[host] $ df
```

Note: Observe that `/dev/sdb2` (in this case) is mounted as `/media/casper-rw` on the host machine. This may be different for your system.

3-2-3. Examine the contents of the /etc/exports file by executing:

```
[host] $ cat /etc/exports
```

3-2-4. 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.

3-2-5. 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.

3-3. 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

3-3-1. Log in to the QEMU system.

3-3-2. 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").

3-4. 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.

3-4-1. 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.
```

3-4-2. 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 GtkTerm window:

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

3-4-3. Run the hello application directly over the network by running:

```
# ./myapp
```

3-4-4. Open a new terminal.

3-4-5. Change to the `myapp` directory:

```
[host]$ cd ~/emblnx/labs/lab3/Avnet-Digilent-ZedBoard-
2014.2/components/apps/myapp
```

3-4-6. Try making some changes to the `myapp.c` file (to the `print` statement, for example).

```
[host] $ gedit myapp.c
```

3-4-7. Change the first `printf` statement to `printf("Hello, Welcome to the XUP workshop!\n")`.

3-4-8. Change to the PetaLinux project directory.

```
[host]$ cd ~/emblnx/labs/lab3/Avnet-Digilent-ZedBoard-2014.2
```

3-4-9. Build the application only.

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

3-4-10. 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.

Navigating the Web Page on HTTP — Step 4

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

4-1. 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.

4-1-1. 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.

4-2. 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.

4-2-1. Log in to the system.

4-2-2. 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.

4-2-3. 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.

4-2-4. 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.

4-2-5. Press <**Ctrl + a**> and then <**x**> to shut down QEMU.

Building the Web-Enabled Application Step 5

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.

5-1. 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.

5-1-1. Make sure that you are in the PetaLinux project location; i.e., `~/emblnx/labs/lab4/Avnet-Digilent-ZedBoard-2014.2`.

5-1-2. 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`.

5-2. Copy the cgi-leds source from the `sources/lab4/cgi-leds` directory.

5-2-1. Change to the newly created application directory:

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

5-2-2. 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**.

5-3. Select the new application to be included in the build process. The application is not enabled by default.

5-3-1. Make sure that you are in the project directory; i.e., `~/emblnx/labs/lab4/Avnet-Digilent-ZedBoard-2014.2`.

5-3-2. Launch the rootfs configuration menu by entering the following command:

```
[host] $ petalinux-config -c rootfs
```

5-3-3. Press the Down Arrow key (↓) to scroll down the menu to **Apps**.

5-3-4. Press <**Enter**> to go into the Apps sub-menu.

The new application **cgi-leds** is listed in the menu.

5-3-5. Scroll to **cgi-leds** and press <**Y**> to select the application.

5-3-6. 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.

5-4. Build the image.

5-4-1. Enter the following command to build the image:

```
[host] $ petalinux-build
```

Let the build process to complete and the image be created.

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

5-5-1. 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.

5-5-2. If not, copy the `BOOT.BIN` file from the `~/emblnx/sources/lab1/SDCard` directory to the SD card.

5-6. To download the image, run the `DHCP` server on the host.

5-6-1. Run the `DHCP` server:

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

5-7. Power up the board and set the serial port terminal.

5-7-1. Power ON the board.

5-7-2. Ensure that `/dev/ttyACM0` is set to read/write access:

```
# sudo chmod 666 /dev/ttyACM0
```

5-7-3. In the dashboard, in the Search field, enter the serial port.

5-7-4. Select the **Serial port terminal** application.

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

5-8. Boot the new embedded Linux image over the network.

5-8-1. Watch the booting process in the GtkTerm window.

5-8-2. 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
```

5-8-3. 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
```

5-8-4. 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.

5-8-5. Watch the GtkTerm window.

5-9. Run the `led.cgi` program.

5-9-1. Once the board reboots, log in and start the `httpd` service:

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

5-9-2. 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.

5-9-3. Modify the URL to include the path to the new `led.cgi` application:

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

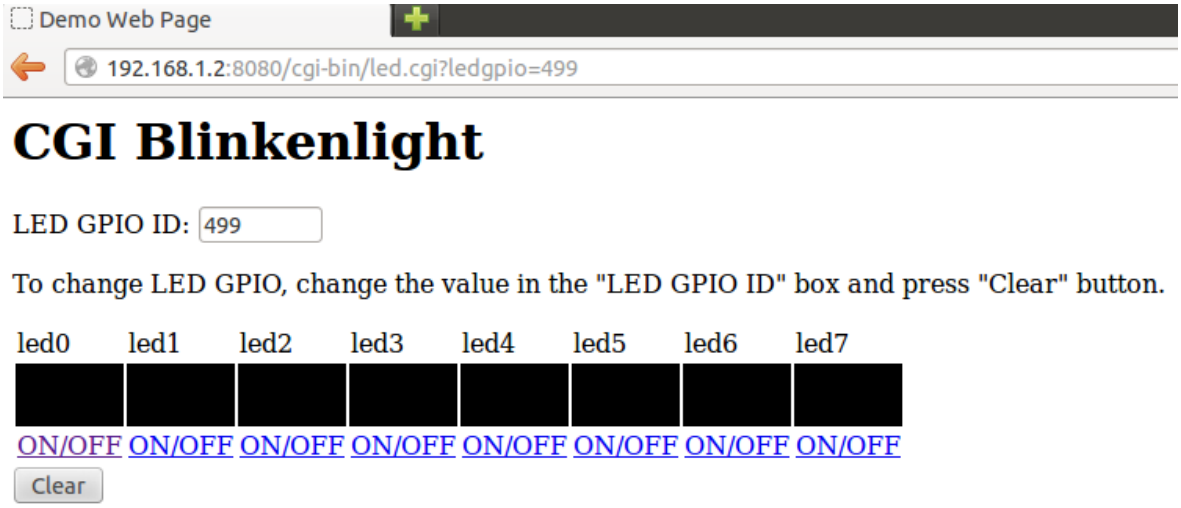

Figure 4. led.cgi application

5-9-4. 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.

5-9-5. In the browser, enter **243** in the LED GPIO ID field.

5-9-6. Click **ON/OFF** in the web page and watch what happens on the board and the web page.

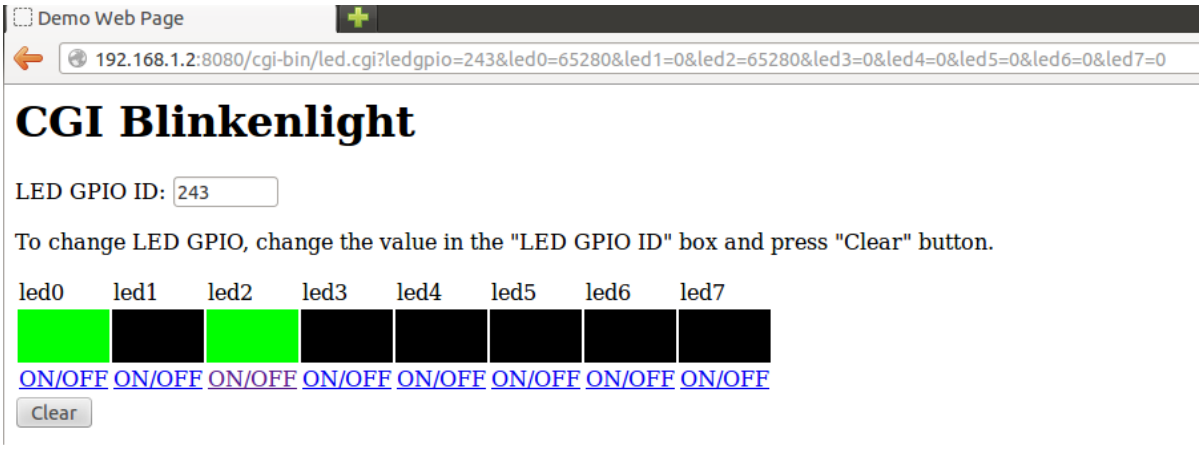

Figure 5. Providing the LED GPIO ID and turning ON/OFF the LEDs

5-9-7. Click clear button to turn OFF all the LEDs.

5-9-8. Once you are done, power off the board.

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 downloading 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

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.