Configuring and Building Palacios/Linux

1/3/2011

1. Check out the Palacios repository

The central Palacios repository is directly accessible from *newskysaw.cs.northwestern.edu* and *newbehemoth.cs.northwestern.edu*. The central repository is approximately 1 hour ahead of the externally visible repository that can be accessed from v3vee.org. All internal developers have read access to the directory. Each developer must create their own local version of the repository, this is done with *git clone*.

If you are logging at *newskysaw*, do `git clone /home/palacios/palacios`. And if you are logging in *newbehemoth*, do `git clone /home-remote/palacios/palacios`. Students in the EECS 441 class should be using newbehemoth. On any other machine, you can clone the repository via ssh, provided you have a *newskysaw* account:

```
git clone
ssh://you@newskysaw.cs.northwestern.edu//home/palacios/palacios
```

No matter how you clone, the clone command creates a local copy of the repository at *`./palacios/`*.

Note that both *newskysaw* and *newbehemoth* have all the tools installed that are needed to build and test Palacios and Kitten. If you develop on another machine, you will need to set those tools up for yourself. This isn't hard and the tools are all free. See the technical report for what tools you will need.

When you first clone the repository, you will get the *master* branch, which exists for historical reasons. All current development work is done in the *devel* branch of the repository. The developer can access this branch via:

```
[andrewlxia@newbehemoth palacios]$ git checkout --track -b devel ori-
gin/devel
```

or

```
[andrewlxia@newbehemoth palacios]$ /opt/vmm-tools/bin/checkout_branch
devel
```

Now check your current branch, make sure it is in *devel* branch:

```
[andrewlxia@newbehemoth palacios]$ git branch
* devel
  master
```

And do pull to make sure all your local code is updated:

```
[andrewlxia@newbehemoth palacios]$ git pull
Already up-to-date.
```

Finally, to visualize the development history in the repository:

```
[andrewlxia@newbehemoth palacios]$ gitk
```

You will need X11 working in order for gitk to function. You can get a text view by running `git log`.

2. Check out Host Linux codebase w/ Palacios backend code

Palacios is independent of host operating systems and can be embedded into several OSes, such as GeekOS, Kitten, Minix, and Linux. Here we use Linux as its host OS. To embed Palacios inside a Linux host, there is code that resides in the Linux kernel to implement the interfaces that Palacios uses to access host OS services and that the host OS uses to access Palacios. This code currently resides in a Linux kernel code branch (for long term, this code will be separated from a specific Linux version and turned into a dynamic loadable kernel module). You can check out the needed code as part of a local branch of Linux that we use. To do so on *newbehemoth*:

```
[andrewlxia@newbehemoth eecs441]$ git clone /home-remote/palacios-linux/linux-2.6.32.y.git
```

The Palacios interface code is almost entirely under *arch*/x86/palacios

```
[andrewlxia@newbehemoth linux-2.6.32.y]$ ls arch/x86/palacios/
Kconfig   palacios-dev.c   palacios-mm.h
palacios-ringbuffer.c  palacios-socket.c palacios-vnet.c
Makefile   palacios-file.c  palacios-packet.c
palacios-ringbuffer.h  palacios-stream.c  palacios-vnet.h
palacios.c  palacios-file.h  palacios-packet.h
palacios-ringbuf.h   palacios-stream.h
palacios-console.c  palacios.h   palacios-queue.c
palacios-serial.c   palacios-vm.c
palacios-console.h  palacios-mm.c
palacios-queue.h   palacios-serial.h   palacios-vm.h
```

3. Configure Palacios

You can find a detailed manual of getting and building Palacios and Kitten from scratch in the Palacios website (http://www.v3vee.org/palacios/manual/node7.html, some are not updated with devel branch). The methodology for building with Linux is very similar. Here we only give the specific requirements related to the procedure of embedding Palacios inside the Linux host. To configure Palacios, change to the "palacios/" directory and type the following:

```
[andrewlxia@newbehemoth palacios]$ make menuconfig
```

or

```
[andrewlxia@newbehemoth palacios]$ make xconfig
```

Make sure you configure the following settings:

```
   Target Configuration -> Target Host OS -> Linux 2.6
   Target Configuration -> Supported host OS features -> Host support
for multiple threads
   Target->Supported Host Features->Host Support for VM Console is
needed for compilation with Linux to complete
```

Don't forget to include the devices that your virtual machine requires. Specifically, to make Palacios runs smoothly on Linux host, we need several additional devices which are not enabled in the default configuration:

```
    CGA -> Curses Virtual Console
    Serial Port
    Stream based character frontend
```

When you have configured the components you want to build into Palacios, save the configuration and close the window. To build Palacios type the following:

```
[andrewlxia@newbehemoth palacios]$ make
```

or

```
[andrewlxia@newbehemoth palacios]$ make all
```

Once the Palacios static library has been built you can find the library file, `libv3vee.a`, in the Palacios root directory. This library contains Palacios with all the features and devices you have configured, with open bindings for the Palacios backend that the Linux embedding will use. The library will then be statically linked with the Linux kernel.

4. Build or fetch a Palacios-runnable guest image

Palacios extends Linux with the ability to run virtual machine images. A VM image contains a description of the hardware of the VM, and blobs that represent disk content, such as CD ROM images.

For testing, we will create (or use) a tiny Linux setup as the guest. The guest Linux will boot in text mode to a command prompt. The guest Linux is essentially a bootable CD-ROM (a simple “LiveCD”) containing a normal Linux kernel, and a RAM disk image with a simple filesystem on it.

Note that there is an opportunity for confusion here. We will also be building a “host Linux” bootable CD for testing (Section 5). The host Linux will also boot in text mode to a command prompt. However, the host Linux will contain a Linux kernel that is augmented with Palacios, and its RAM disk image will contain the guest Linux image and command-line user-level tools for starting and interacting with VMs.

The Palacios/Linux setup boils down to a Palacios-augmented Linux kernel and the user-level tools for using Palacios to run VM images from the command line. As such, it should be possible to integrate it with any distribution. The purpose of the simple host Linux image we create is to give you a fast-to-build and simple test environment for Palacios and the purpose of the simple guest Linux image is to give you a guest that is fast to boot for use in testing.

4.1 Building a Guest OS Image

To run Linux as the guest OS on Palacios, first you have to build your own bootable guest OS file. Here is a link for brief introduction on how to do that: `http://www.v3vee.org/palacios/guest_build_manual/`. We already have built a Linux guest with minimal tools in it, which you can copy from `newbehemoth.cs.northwestern.edu:/home/andrewlxia/eecs441/guest_os.iso` and use it now. This image is also included in the host Linux RAM disk image that we will give you access to, and describe, in the next section. As your project expands, you may need to put in more tools and libraries into the guest, or create your own specialized guest.

4.2 Configure your Virtual Machine

Before you can run your VM on Palacios with the guest OS you just built, you need to configure your virtual machine. Palacios uses XML-based configuration file to specify the VM hardware and media. Under `"palacios/utils/guest_creator"`, there is a guest creator utility that helps users to building a Palacios-runnable guest image file with VM configuration embedded in it.

Change to the `"palacios/utils/guest_creator"` directory and build the guest creator utility:

```
[andrewlxia@newbehemoth guest_creator]$ make
```

You will get the `"build_vm"` utility:

```
[andrewlxia@newbehemoth guest_creator]$ file build_vm
 build_vm: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dy-
 namically linked (uses shared libs), for GNU/Linux 2.6.9, not
 stripped
```

The guest configuration file is written in XML. A sample configuration file is provided: `"default.xml"`. Make a copy of the default configuration file named `"myconfig.xml"` and edit the configuration elements that you are interested in (if a device is included in the guest configuration file, it must be configured in the section Configuring and building Palacios or the guest will not boot). Of particular importance is the `"files"` element, which includes your guest OS bootable image file:

```
<files>
 <file id="boot-cd" filename="guest.iso"></file>
</files>
```

When you finish editing the guest configuration, save the configuration file. The guest image consists of the guest configuration file and the Linux ISO image. Build the guest image with the guest creator utility:

```
[andrewlxia@newbehemoth guest_creator]$./build_vm myconfig.xml -o
guest.img
```

5. Configure and build the host Linux

Palacios is embedded in Linux by statically linking the libv3vee.a library with a Linux kernel that has the interface code described earlier. The combination of the two augments the Linux kernel by providing system calls to create, run, and destroy virtual machines.

For those familiar with Linux, this section describes how to do a "make isoimage", creating a simple Linux LiveCD whose initrd contains Palacios-augmented kernel, the Palacios userspace utilities, and the simple guest image described in Section 4.

For those not familiar with Linux, here is a bit more detail on what is going on. After compiling the Linux kernel and linking it with the Palacios library, we are left with a kernel image. The kernel image needs to be embedded in a distribution in order to be useful. The distribution does two core things: it includes a boot loader that can boot the kernel (copy it to memory, and jump to it, at machine startup time), and it contains a root file system ('/') with programs that users can interact with. Here, we will create a distribution in the form of a CD ROM image. The image will contain a simple boot loader, the kernel, and the contents of a RAM disk filesystem. When we boot the CD ROM, the boot loader will copy the kernel and the RAM disk contents into memory, and then jump to the kernel's entry point, passing it a pointer to where the RAM disk has been placed in memory. The kernel will initialize the machine, treat the RAM contents as a filesystem and mount it in on '/', it will then start the first user process (/etc/init), which will start a bash shell that we can interact with (/bin/bash).

5.1 Building the ramdisk file system for host Linux

To run the basic host Linux test environment, an initial ramdisk filesystem is needed. This filesystem (the "initrd") will contain a set of basic utilities and the guest Linux image. You can copy an example from Lei's directory:

```
[andrewlxia@newbehemoth eecs441]$ cp
/home/andrewlxia/eecs441/initrd.tar.gz .
[andrewlxia@newbehemoth eecs441]$ tar xzf initrd.tar.gz
```

This will create an initrd subdirectory, the important part of which is initrd/initramfs. This directory contains all the files and directories that will be placed in the RAM disk image when you build the kernel distribution.

You will also need one special file to be created, initrd/initramfs/dev/console, which is the device file for the console. Since this is a device file and owned by root, you will need Lei to do this for you, as you don't have the permissions. He only needs to do this once.

If you would like, you can set up your initrd any way you would like. A base initrd comes with Linux. Some important things to remember if you do so are the following, described from the perspective of the initrd/initramfs directory:

```
[andrewlxia@newbehemoth initramfs]$ mkdir -p proc sys var/log
```

Edit the ``init_task'' script and uncomment these lines:

```
#mknod /dev/tty0 c 4 0
#mknod /dev/tty1 c 4 1
#mknod /dev/tty2 c 4 2
```

Create the ``console'' device. If you have sudo or root access it is possible to create this device manually:

```
[andrewlxia@newbehemoth initramfs]$ sudo mknod dev/console c 5 1
[andrewlxia@newbehemoth initramfs]$ sudo chmod 0600 dev/console
```

If you do not have sudo or root access it is still possible to create the "console" device indirectly through the kernel build. Change to the "initrd/" directory and create a file called "root_files". Add the following line:

```
nod /dev/console 0600 0 0 c 5 1
```

The "root_files" file is used when building the Linux kernel as described in the section Configuring and building the Linux kernel. Finally, create any additional directories and copy any additional files that you need.

After this, you need to build the v3vee user control binaries and then copy the five binaries to your init ram disk file system:

```
[andrewlxia@newbehemoth eecs441]$ cd linux-2.6.32.y/usr/v3_ctrl/
[andrewlxia@newbehemoth v3_ctrl]$ cp v3_mem v3_monitor v3_serial
v3_stop v3_cons v3_ctrl ../../../initrd/initramfs/bin/
```

Also, do not forget to copy your guest OS image file to the init file system:

```
[andrewlxia@newbehemoth palacios]$ cp guest_os.img ../initrd/initramfs/
```

5.2 Build Host Linux:

To configure the Linux kernel, Use:

```
[andrewlxia@newbehemoth palacios]$ make menuconfig
```

Or

```
[andrewlxia@newbehemoth palacios]$ make xconfig
```

To embed Palacios in Linux, make sure to enable the following options in your Linux configuration:

```
General Setup -> Initial RAM filesystem and RAM disk (initramfs/initrd)
support

Initramfs source file(s):  ../initrd/initramfs

Virtualization -> Palacios -> Include Palacios virtual machine monitor
```

```
Path to pre-built Palacios tree: ../palacios
```

After you finish the configuring, save it, and do a make:

```
[andrewlxia@newbehemoth linux-2.6.32.y]$ make isoimage
```

This will build a bootable iso image file in a subdirectory. This file is the bootable CD ROM containing the Palacios-augmented Linux kernel and the RAM disk image containing the Palacios user tools and the simple guest image.

```
[andrewlxia@newbehemoth linux-2.6.32.y]$ file arch/x86/boot/image.iso
arch/x86/boot/image.iso: ISO 9660 CD-ROM filesystem data 'CDROM
' (bootable)
```

6. Run Palacios/Linux on Qemu

We commonly test Palacios using the Qemu x86 system emulator, which contains an implementation of the AMD SVM extensions. Note that the host image can also be used for testing on real hardware using our PXE network-booting setup. Finally, you can burn the image.iso file to a real CD and you should be able to boot machines from it.

You can boot the host Linux image file on Qemu using syntax like this:

```
[andrewlxia@newbehemoth linux-2.6.32.y]$ qemu-system-x86_64 -m 2046  -
smp 1 -serial file:serial.out -cdrom arch/x86/boot/image.iso
```

This means “create an emulated x86_64 PC with a single processor, 2 GB of RAM, a serial port, and a CD ROM drive; then insert our CD ROM image into the drive and attach the serial port to a local file; then start the machine”.

After the image boots on Qemu, you can follow the demo in the following video to set up memory for Palacios, and start a Palacios VM and load your guest OS.

The Demo video:

```
http://www.cs.northwestern.edu/~lxi990/frame/eecs441/palacios-linux.wmv
```

The `setup.sh` script used in the video to release contiguous physical memory segments for Palacios is like this (depending on how many physical memory the machine has and how many you want to release for use of Palacios):

```
echo offline > /sys/devices/system/memory/memory2/state
echo offline > /sys/devices/system/memory/memory3/state
echo offline > /sys/devices/system/memory/memory4/state
echo offline > /sys/devices/system/memory/memory5/state
echo offline > /sys/devices/system/memory/memory6/state
echo offline > /sys/devices/system/memory/memory7/state
echo offline > /sys/devices/system/memory/memory8/state
```

```
echo offline > /sys/devices/system/memory/memory9/state
v3_mem 268435456 1073741824
```

Final comments

This document is an incomplete manual for configuring and running Palacios in Linux host, and I will keep updating it. For any question about this document or beyond but related to Palacios configuring/building process, please contact Lei Xia (lxia@northwestern.edu).