Institut Eurécom [1]
Department of Mobile Communications
2229, route des Crêtes
B.P. 193
06904 Sophia-Antipolis
FRANCE

Research Report RR-08-211

Practical and Unified Process for developing the Future Mobile Internet with Simultaneous Access (MISA)

February 26, 2008

Huu Nghia NGUYEN
Prof. Christian BONNET

Tel: (+33) 04.93.00.82.38
Fax: (+33) 04.93.00.26.27
Email : {Huu-Nghia.Nguyen,Christian.Bonnet}@eurecom.fr

[1] Institut Eurécom's research is partially supported by its industrial members: BMW, Bouygues Télécom, Cisco Systems, France Télécom , Hitachi Europe, SFR, Sharp, STMicroelectronics, Swisscom, Thales

Table of contents

Abstract

Future Mobile Internet has to cope with multi-interface mobile nodes and multi-access networks that provide simultaneous use of access technologies for wireless bandwidth aggregation and load balancing. However working in such a multi-homing & mobility environment requires cost, time and efforts; a new process based on virtualization should be considered. When it comes to virtualization, there's not just one way to do it; this document explores the ideas behind each virtualization technologies for virtual machines and describes in details the User Mode Linux (UML) approach that can be adapted easily to different purpose: virtual networking, distributed application development, driver or kernel development. We then propose a practical and unified process for developing the future Mobile Internet with Simultaneous Access (MISA) using UML and Mobile IPv6 for Linux (MIPL).

Keywords: *User Mode Linux, UML, B3G, Multi-homing, Mobility, Mobile IPv6 for Linux, MISA*

Introduction

This document describes in details the User Mode Linux (UML) approach that can be adapted easily to different purposes: virtual networking, distributed application development, driver or kernel development. We also explore the ideas behind each virtualization technology for virtual machines. A practical and unified process for developing the Mobile Internet with Simultaneous Access (MISA) is also proposed for both UML and real testbed.

1. Abbreviations & Terminology

Guest OS/kernel/file system	The OS/kernel/file system for the virtual machine
Host OS/kernel/file system	The OS/kernel/file system for the real host on which the virtual machine is running
OS	Operating System
VM	Virtual Machine
UML	User Mode Linux

2. Classification of Virtualization Technologies

This section introduces you to three of the most common methods of virtualization in Linux and identifies their relative strengths and weaknesses.

2.1. Full virtualization

Full virtualization, otherwise known as native virtualization, is an interesting method of virtualization. This model uses a virtual machine (hypervisor) that mediates between the guest operating systems and the native hardware of the host machine (see Figure 1). Certain protected instructions must be trapped and handled within the hypervisor because the underlying hardware isn't owned by an operating system but is instead shared by it through the hypervisor.

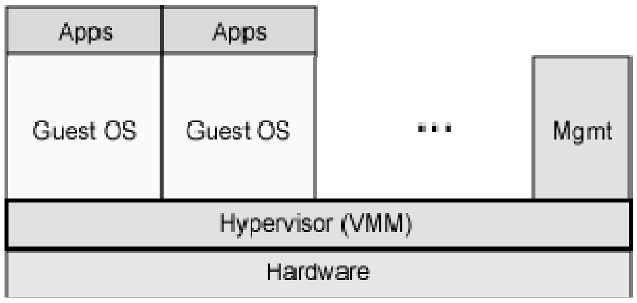

Figure 1. Full virtualization uses a hypervisor to share the underlying hardware

Advantages	Performance is less than bare hardware because of the hypervisor mediation. The biggest advantage of full virtualization is that any operating system can run unmodified on the guest machine.
Disavantages	The only constraint is that the operating system must support the underlying hardware (virtualized hardware device)
Examples	VMware, Microsoft VirtualPC, QEMU, Xen (extended to full)

Of all these products, VMWare has been being the leader for the virtualization because it supports a wide range of OS for both guest and host machines. It has been widely used for deploying and testing complex applications in enterprises. However, full virtualization totally isolates the guest OS from the host OS; therefore it costs same or more time and efforts for developing new protocols on the virtual machine (comparing to a real environment).

2.2. Paravirtualization

Paravirtualization is another popular technique that has some similarities to full virtualization. This method uses a hypervisor for shared access to the underlying hardware but integrates virtualization-aware code into the guest OS itself (see Figure 2). This approach obviates the need for any recompilation or trapping because the guest OS themselves cooperate in the virtualization process.

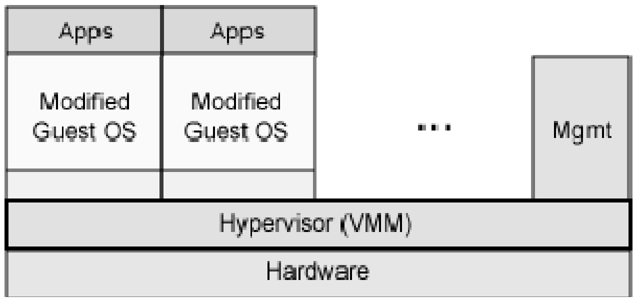

Figure 2. Paravirtualization integrates virtualization-aware code into the guest OS

Advantages	Paravirtualization offers performance near that of an unvirtualized system.
Disavantages	Paravirtualization requires the guest operating systems to be modified for interacting with the hypervisor (except UML). It can only support linux OS.
Examples	Xen, User Mode Linux (A new architecture named ‘um’ is created so there is no modification in kernel modules)

Note 1: Intel has contributed modifications to Xen to support their VT-x (formerly Vanderpool) architecture extensions. These technologies, while differing quite substantially in their implementation and instruction sets, are managed by a common abstraction layer in Xen and enable unmodified guest operating systems to run within Xen virtual machines, starting with Xen 3.0. Hardware assisted virtualization offers new instructions to support direct calls by a paravirtualized guest/driver into the hypervisor, typically used for I/O or other so-called hypercalls. "Hardware" accesses are under complete control of the hypervisor.

Note 2 : User Mode Linux is considered as a paravirtualization solution. However, it integrates a new virtualization architecture into the guest OS to avoid modifying existing kernel modules. In this sense it can be considered as a full virtualization for linux systems.

2.3. Operating system-level virtualization

The final technique, operating system-level virtualization, uses a different technique than those covered so far. This method uses a single kernel for both the host and the guests. To improve the performance, all guest machines share the same scheduler and kernel modules with the host machine. To ensure the independent servers from one another, each guest (private server) has separated virtual environment and this require modifying the kernel at every kernel modules: ieee802.11, IPv4, IPv6, UDP, TCP, SCTP…

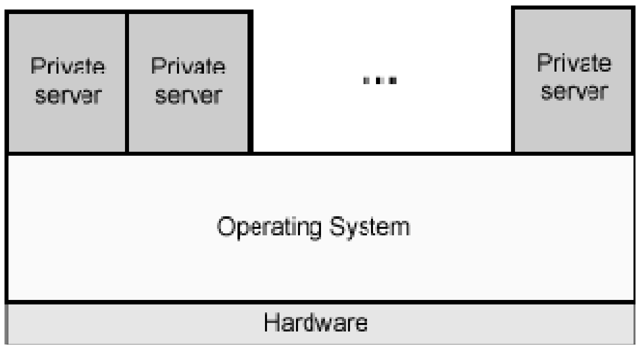

Figure 3. Operating system-level virtualization isolates servers

Advantages	Native performance
Disavantages	Operating system-level virtualization requires changes to the operating system kernel (even kernel modules)
Examples	OpenVz

Developing a new protocol with this virtualization technology requires using the notion of Virtual Environment in the source code. A module created for running with this virtualization technology can not run with the real environment; therefore a double effort must be done.

2.4. Comparison of UML, VMware (KVM, , QEmu), OpenVz

Name	Modification of Guest OS	Device Driver	Method of operation	Best Use	Guest OS speed relative to Host OS
VMware	No	A set of virtualized device: can not add new hardware device.	Full	Experiment of existing protocols, apps. Development of apps.	Near native
KVM	No	Depends on QEMU.	Full	Experiment of existing protocols, apps. Development	Near native

				of apps.	
QEMU	No	A set of virtualized device. Need to modify QEMU source for extension	Full	Experiment of existing protocols, apps. Application development	Near native
User Mode Linux	Yes (add a new branch ARCH=um)	Easy to create new virtual device & device driver.	Para, extensible to Full	Kernel development, Application development. Virtual Networking,	Near native Our test show the perfornace is fine for many VMs with TLS (Thread Local Storage) support Some documents consider slow as they didn't have TLS support at that moment
OpenVZ	Yes (modify all kernel modules for virtual environment)	Share with the host OS (Linux)	OS-level	Virtualized Server Isolation, User space applications development	Native

3. Basic User Mode Linux (UML)

User-Mode Linux is a safe, secure way of running Linux versions and Linux processes. Run buggy software, experiment with new Linux kernels or distributions, and poke around in the internals of Linux, all without risking your main Linux setup. User-Mode Linux gives you a virtual machine (the guest) that may have more hardware and software virtual resources than your actual, physical computer. Disk storage for the virtual machine is entirely contained inside a single file on your physical machine. Nothing you do on the virtual machine can change or damage your real host computer, or its software.

UML has two major components: a guest kernel and a guest root file system. The guest kernel is basically a version of the mainline linux kernel provided with a new architecture named *um* ,i.e. user mode architecture, while the guest root file system is a single file emulating an actual linux file system that may reside on your hard disk. The easiest way to start experimenting with UML is to download a pre-compiled guest kernel and a matching guest root file system (Fedora, Ubuntu, Debian, Gentoo…) Both of these can be downloaded from the UML homepage:

http://user-mode-linux.sourceforge.net/old/dl-sf.html

http://uml.nagafix.co.uk/

http://www.dit.upm.es/vnumlwiki/index.php/Download

From the command line:

```
wget http://downloads.sourceforge.net/vnuml/linux-2.6.16.27-bs2-xt-1m.tar.bz2

wget http://downloads.sourceforge.net/vnuml/root_fs_tutorial-0.5.1.bz2

bzcat  root_fs_tutorial-0.5.1.bz2 > rootfs.img

tar xjf linux-2.6.16.27-bs2-xt-1m.tar.bz2

linux-2.6.16.27-bs2-xt-1m/linux-2.6.16.27-bs2-xt-1m ubd0=rootfs.cow,rootfs.img eth0=tuntap,,,
```

The guest virtual machine should start up and prompt for username and password. You can logon using root/xxxx as username/password. To turn off the guest virtual machine, use the *halt* command. Note that you can optimize the performance by applying the Separated Kernel Address Space (skas patch) to the host kernel. If you have any problem of running UML, refer to Appendix A.

There also exist some tools for working with UML:

UML Utilities	This package contains userspace utilities for use with User-mode Linux, including uml_mconsole, uml_moo, uml_switch, uml_net and tunctl.
VNUML	http://www.dit.upm.es/vnumlwiki/index.php/Main_Page VNUML (Virtual Network User Mode Linux) is an open-source general purpose virtualization tool designed to quickly define and test complex network simulation scenarios based on the great User Mode Linux (UML) It has been developed with the partial support from the European Commission under the Euro6IX IST research project

4. Unified Process for developing MISA

Future Mobile Internet has to cope with multi-interface mobile nodes and multi-access networks that provide simultaneous use of access technologies for wireless bandwidth aggregation and load balancing. In this section, we propose a practical and unified process for developing the future Mobile Internet with Simultaneous Access (MISA) using UML and Mobile IPv6 for Linux (MIPL).

MIPL releases, up to 1.1, are for Linux kernel 2.4 series. Releases 0.x and 1.x implement the whole MIPv6 inside the kernel. Later releases (2.x) are exclusively for 2.6 kernels. MIPL 2.0 is a complete rewrite of the protocol, implementing only the absolutely necessary infrastructural support for MIPv6 in the kernel and moving most functionality to user space. Enabling Mobile IP feature in linux requires you to recompile the linux kernel.

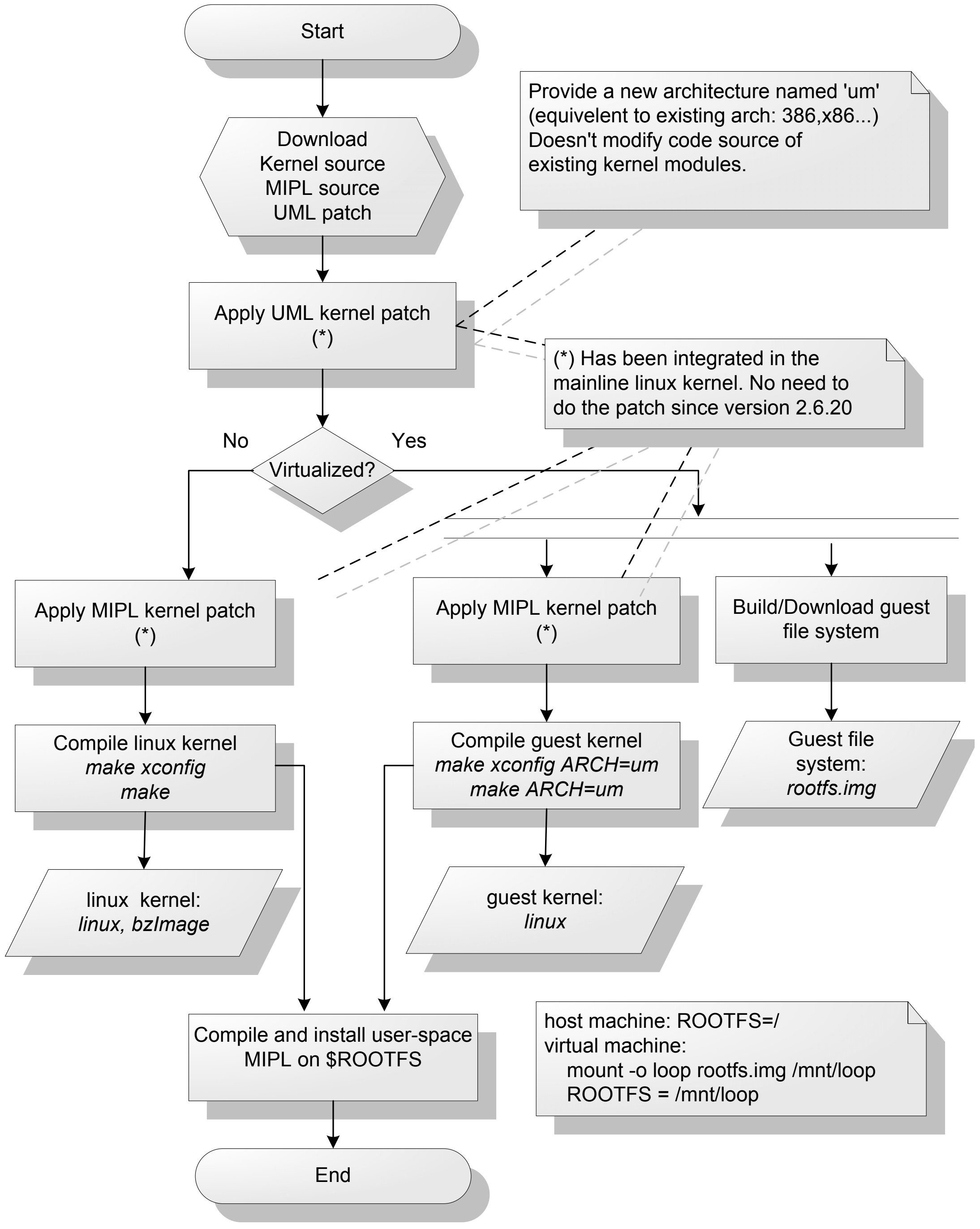
Start
Download
Kernel source
MIPL source
UML patch
Provide a new architecture named 'um' (equivelent to existing arch: 386,x86...) Doesn't modify code source of existing kernel modules.
Apply UML kernel patch (*)
(*) Has been integrated in the mainline linux kernel. No need to do the patch since version 2.6.20
No
Yes
Virtualized?
Apply MIPL kernel patch (*)
Apply MIPL kernel patch (*)
Build/Download guest file system
Compile linux kernel
make xconfig
make
Compile guest kernel
make xconfig ARCH=um
make ARCH=um
Guest file system:
rootfs.img
linux kernel:
linux, bzImage
guest kernel:
linux
Compile and install user-space MIPL on $ROOTFS
host machine: ROOTFS=/
virtual machine:
mount -o loop rootfs.img /mnt/loop
ROOTFS = /mnt/loop
End

Figure 4. Unified Process for Mobile IP in UML/real testbed

The above figure shows a practical and unified process for developing/deploying Mobile IP in UML as well as in real testbed. This process reduces the cost of equipments but also the time and effort for developing/deploying/debugging. The process stays the same if you want to develop a new kernel module.

UML has been integrated into the mainline Linux kernel. So starting from the linux kernel source, we apply the UML kernel patch to add a new architecture for UML (ARCH=um) to the linux kernel source tree. From now on, the developer can work on this kernel source for both real machine and virtual machines. To develop Mobile IP, we apply MIPL patch, add new functionalities to MIPL, develop new protocols, and compile it. If we want to deploy in the real test bed, just compile this linux kernel source in the x86/sparc/mips architecture, otherwise, we create a guest kernel running in UML architecture by compiling the kernel source with 'ARCH=um' option in the *make* command line.

The guest root file system is a normal file that can be mount directly to the host file system using '-o loop' option. This allows developers to work with the guest file system without the need of turning on the virtual machine.

Copy On Write (COW) is another interesting feature when playing with UML as it allow different virtual machines to run on the same guest root file system and save the disk space by storing the differences in .cow files.

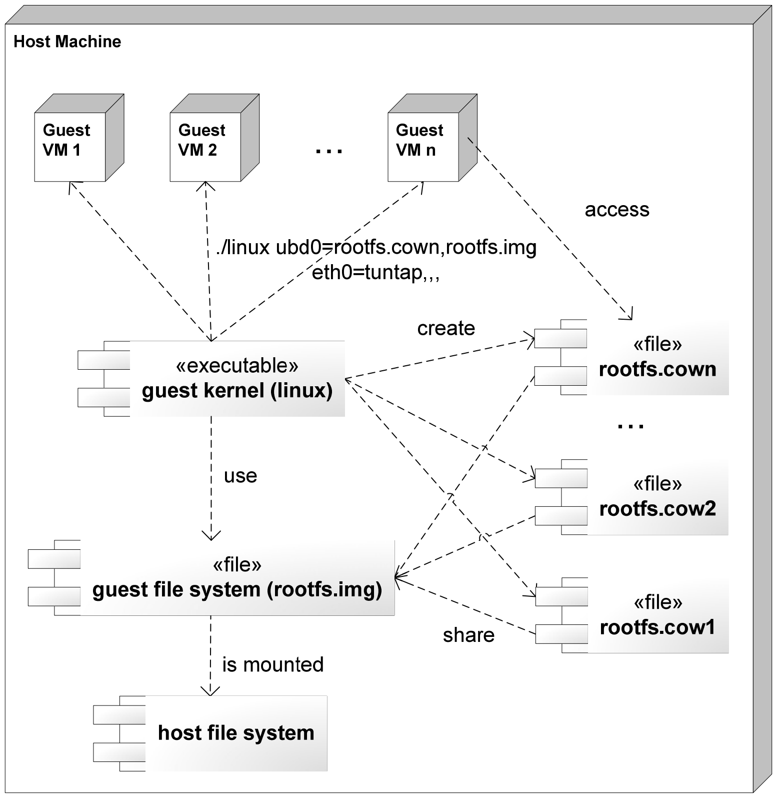

Figure 5. The guest file system is shared by different Virtual Machine

From the figure, the two main components of UML are the guest kernel (linux) and the guest file system (rootfs.img). Using COW, these components are considered as a template and each command *linux ubd0=rootfs.cown,rootfs.img* corresponds to a guest virtual machine having access to roofs.cown (read/write) and rootfs.img (read only). This allow to save disk space, to create and run as many virtual machines as we want just with a script of 3 lines using the *for* loop ☺.

5. Use case: Compiling guest kernel supporting MIPv6

5.1. Preparing for the guest kernel

Download all the necessary files to the local host before installing your host and UML kernels. In our setup, we assume to use the kernel 2.6.16 for compatibility with the newest MIPL 2.0.2

Linux kernel	www.eu.kernel.org
Guest UML kernel patches	http://www.user-mode-linux.org/~blaisorblade/patches/guest/
MIPL user space	http://www.mobile-ipv6.org/software/download/mipv6-2.0.2.tar.gz
MIP kernel patch	http://www.mobile-ipv6.org/software/download/mipv6-2.0.2-linux-2.6.16.patch.gz

```
mkdir ~/uml

cd ~/uml

wget http://www.eu.kernel.org/pub/linux/kernel/v2.6/linux-2.6.16.tar.gz

wget http://www.user-mode-linux.org/~blaisorblade/patches/guest/uml-2.6.16-bs2/uml-2.6.16-bs2.patch.bz2

wget http://www.user-mode-linux.org/~blaisorblade/patches/skas3-2.6/skas-2.6.16-v9-pre9/skas-2.6.16-v9-pre9.patch.bz2

wget http://www.mobile-ipv6.org/software/download/mipv6-2.0.2.tar.gz

wget http://www.mobile-ipv6.org/software/download/mipv6-2.0.2-linux-2.6.16.patch.gz
```

5.2. Compiling the guest kernel

Compiling UML kernel is just like compiling any kernel, except that you have to include ‘ARCH = um’ in the command line. For the real deployment, just remove ‘ARCH=um’ from the command line (you may need to issue make mrproper when switching between um, i386, x86, mips, sparc architecture). Let's go through the steps:

1. Unpack file(s)

```
tar -xzf linux-2.6.16.tar.gz
```

2. Apply the uml kernel patch

```
cd ~/uml/linux-2.6.16

bzcat ~/uml/uml-2.6.16-bs2.patch.bz2 | patch -p1 --dry-run --verbose
```

3. Run your favorite config (can be xconfig, menuconfig or config)

```
make xconfig ARCH=um

make config ARCH=um

make menuconfig ARCH=um
```

Set the following options in UML-Specific options. You can leave the remaining options unchanged

```
CONFIG_MODE_TT=N

CONFIG_STATIC_LINK=Y

CONFIG_HOSTFS=M

CONFIG_HIGHMEM=Y

X86_GENERIC=Y
```

Note: If the host is configured with a 2G/2G address space split rather than the usual 3G/1G split, then the packaged UML binaries will not run. They will immediately segfault.

4. Compile the kernel in UML architecture. If you have problems while compiling, refer Appendix A for some common errors. The result is a file called `linux' in the top directory of your source tree.

```
make linux ARCH=um
```

5. You may notice that the final binary is pretty large. This is almost entirely symbol information. The actual binary is comparable in size to a native kernel. You can run that huge binary, and only the actual code and data will be loaded into memory, so the symbols only consume disk space unless you are running UML under gdb. You can strip UML to see the true size of the UML kernel

```
strip linux
```

5.3. Building/downloading the guest file system

We recommend to use a pre-build UML root filesystem (Fedora, Ubuntu, Debian, Gentoo…) downloaded from http://uml.nagafix.co.uk/ and make it compatible with VNUML if you want to use VNUML for creating virtual testbed as in http://www.dit.upm.es/vnumlwiki/index.php/Create-rootfs.

```
cd ~/uml

wget http://uml.nagafix.co.uk/FedoraCore6/FedoraCore6-x86-root_fs.bz2
```

```
bzcat  FedoraCore6-x86-root_fs.bz2 > rootfs.img
```

You can also make your own file system as shown in Appendix B

Note: To speed up the performance, mount the *rootfs.img* to */mnt/loop* and comment the line *'/sbin/start_udev'* in */mnt/loop/etc/rc.d/rc.sysinit* of the guest file system to disable udev file system. You may need to run MAKEDEV to create tty devices in /dev directory.

5.4. Enabling Mobile IP in the guest kernel

If your UML machine runs well, it's time to enable the Mobile IP feature. Follow the instruction in http://www.tldp.org/HOWTO/Mobile-IPv6-HOWTO/ to enable Mobile IP feature, as if you are working with a normal linux kernel, then recompile the kernel.
Note: The HOWTO document is decicated to MIPL 1.x and is out of date. Only use it as reference

5.4.1. Kernel Space

Unpack file(s) and apply the MIPL patch to the kernel

```
cd ~/uml

gunzip mipv6-2.0.2-linux-2.6.16.patch.gz

cd ~/uml/linux-2.6.16

patch -p1 --dry-run < ~/uml/mipv6-2.0.2-linux-2.6.16.patch
```

The --dry-run option checks that the patch will apply correctly. If you get any failed hunks, you should *not* proceed. If everything went fine do

```
patch -p1 < ~/uml/mipv6-2.0.2-linux-2.6.16.patch.
```

Now your kernel tree is ready for configuration. Run your favorite make xconfig ARCH=um. The MIPv6 options are under "Networking Options". The following options should be present in *".config"*:

```
CONFIG_EXPERIMENTAL=y
CONFIG_SYSCTL=y
CONFIG_PROC_FS=y
CONFIG_MODULES=y
CONFIG_NET=y
CONFIG_NETFILTER=y
CONFIG_UNIX=y
CONFIG_INET=y

CONFIG_IPV6=y
CONFIG_IPV6_TUNNEL=y
CONFIG_IPV6_ADVANCED_ROUTER=y
CONFIG_IPV6_MULTIPLE_TABLES=y
CONFIG_IPV6_SUBTREES=y
CONFIG_IPV6_MIP6=y
CONFIG_IPV6_MIP6_DEBUG=y
```

Compile kernel & modules.

```
make ARCH=um
```

Install new modules to guest file system (rootfs.img)

```
su
mount –o loop ~/uml/rootfs.img /mnt/loop
export UMLROOT=/mnt/loop
make modules_install ARCH=um INSTALL_MOD_PATH=$UMLROOT
```

5.4.2. User Space

Su back as normal user; unpack the file(s) and compile as any normal user space application.

```
tar -xzf mipv6-2.0.2.tar.gz
cd mipv6-2.0.2
export CPPFLAGS="-isystem $HOME/uml/linux-2.6.16/include"
export UMLROOT=/mnt/loop
./configure --with-builtin-crypto --enable-vt --prefix=$UMLROOT/usr/local
make
su
make install
```

Note: repair the file config.sub. Search for "maybe_os" and modify | *linux-gnu** | to | *linux-gnu** | *linux-oldld** | if you can not run configure

5.5. *Creating a virtual testbed*

Here is the scenario, written for VNUML, based on the tutorial for Linux Mobile IPv6 Howto 1.x (http://www.tldp.org/HOWTO/Mobile-IPv6-HOWTO/)

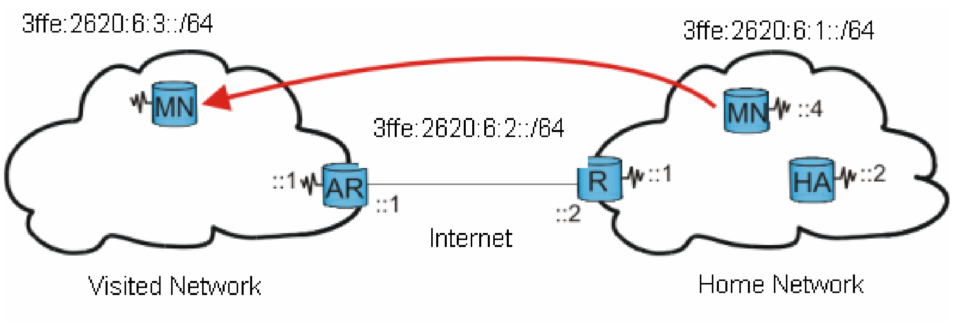

Figure 5. Mobile IPv6 Testbed

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE vnuml SYSTEM "/usr/local/share/xml/vnuml/vnuml.dtd">

<!--
VNUML Mobile IPv6 Scenario
Authors: Nguyen Huu Nghia, Christian Bonnet
This scenario is based on the tutorial for Linux Mobile IPv6 Howto for 1.x
http://www.tldp.org/HOWTO/Mobile-IPv6-HOWTO/
-->
<vnuml>
  <global>
    <version>1.8</version>
    <simulation_name>mip6</simulation_name>
    <automac/>
    <vm_mgmt type="none" />
    <vm_defaults exec_mode="mconsole">
                <filesystem type="cow">/home/nguyenhn/MIPL/fc6/root_fs.fc6</filesystem>
                <kernel>/home/nguyenhn/MIPL/linux-2.6.16.8/linux</kernel>
      <console id="0">xterm</console>
      <!--xterm>gnome-terminal,-t,-x</xterm-->
      <console id="1">pts</console>
      <console id="2">pts</console>
      <console id="3">pts</console>
      <console id="4">pts</console>
      <console id="5">pts</console>
    </vm_defaults>
  </global>
  <net name="Inter" mode="virtual_bridge" />
  <net name="Home" mode="virtual_bridge" />
  <net name="Visit" mode="virtual_bridge" />
  <!-- Mobile Node -->
  <vm name="MN">
              <if id="0" net="Home">
                          <ipv6>3ffe:2620:6:1::4/64</ipv6>
              </if>
  </vm>
  <!-- Home Agent -->
  <vm name="HA">
              <if id="0" net="Home">
                          <ipv6>3ffe:2620:6:1::2/64</ipv6>
              </if>
              <route type="ipv6" gw="3ffe:2620:6:1::1">::/0</route>
  </vm>
  <!-- Router -->
  <vm name="R">
              <if id="0" net="Inter">
                          <ipv6>3ffe:2620:6:2::2/64</ipv6>
              </if>
              <if id="1" net="Home">
                          <ipv6>3ffe:2620:6:1::1/64</ipv6>
              </if>
              <route type="ipv6" gw="3ffe:2620:6:2::1">::/0</route>
  </vm>
```

```
        <!-- Access Router-->
        <vm name="AR">
                <if id="0" net="Inter">
                        <ipv6>3ffe:2620:6:2::1/64</ipv6>
                </if>
                <if id="1" net="Visit">
                        <ipv6>3ffe:2620:6:3::1/64</ipv6>
                </if>
                <route type="ipv6" gw="3ffe:2620:6:2::2">3ffe:2620:6:1::/0</route>
        </vm>
       </vnuml>
```

TODO: Configure radvd, mipd (sripts are downloadable from my site)

TODO: Simulate the mobility of the MN using brctl (scripts are downloadable from my site)

http://www.eurecom.fr/~nguyenhn/uml/scripts/

References

[1] http://user-mode-linux.sourceforge.net/old/dl-sf.html

[2] http://uml.nagafix.co.uk/

[3] http://www.dit.upm.es/vnumlwiki/index.php/Main_Page

[4] http://uml.jfdi.org/uml/Wiki.jsp?page=BuildingRootFileSystems

[5] http://www.tldp.org/HOWTO/Bootdisk-HOWTO/buildroot.html

[6] http://www.ibm.com/developerworks/library/l-linuxvirt/index.html

[7] http://www.ibm.com/developerworks/linux/library/l-linux-kvm/

[8] http://www.tldp.org/HOWTO/Mobile-IPv6-HOWTO/

Appendix A – Known bugs while compiling and running

- Problem about the threads.h

  ln –s $HOME/uml/linux-2.6.16/include/linux arch/um/include/linux

- Apply patches jmpbuf and no-syscallx downloadable from

  http://www.eurecom.fr/~nguyenhn/uml/patches/

- Apply skas patch for host kernel and recompile the host kernel with the patch. If you want to use the same kernel source as the guest. This not only optimize the performance of UML and but also solve the problem of hanging after 'VFS: Mounted root...'
- Solution for other running problem can be found at http://user-mode-linux.sourceforge.net/problems.htm

Appendix B – Building UML root file system

Using Yellow dog Updater Modifier (YUM)

There are many ways to build a UML root file system. We present here a convenient way to create file system is using yum (Yellow dog Updater, Modified) to create a CHROOT environment. Here are some tools found on the UMLWiki page for building such a root file systems

https://lists.dulug.duke.edu/pipermail/yum/2005-January/005950.html

https://lists.dulug.duke.edu/pipermail/yum/2005-February/006055.html

http://uml.jfdi.org/uml/Wiki.jsp?page=Yum3

1. First of all, you need to install yum and configure yum in your host machine. Then create the following files:

vi yum.fc6.conf

```
[main]
cachedir=/var/cache/yum
debuglevel=2
logfile=/var/log/yum.log
exclude=*-debuginfo
gpgcheck=0
obsoletes=1
reposdir=/dev/null

[base]
name=Fedora Core 6 - i386 - Base
mirrorlist=http://fedora.redhat.com/download/mirrors/fedora-core-6
enabled=1

[updates-released]
```

```
name=Fedora Core 6 - i386 - Released Updates
mirrorlist=http://fedora.redhat.com/download/mirrors/updates-released-fc6
enabled=1
```

vi yum3.sh

```
#!/bin/bash
UMLROOT=/mnt/loop
echo " --> Setting up chroot env in $UMLROOT"

if ~[ `id -u` = "0" ] ; then
        mkdir -p \
                $UMLROOT/etc \
                $UMLROOT/dev \
                $UMLROOT/proc \
                $UMLROOT/sys \
                $UMLROOT/var/tmp \
                $UMLROOT/var/cache/yum
        touch $UMLROOT/etc/fstab
        mknod $UMLROOT/dev/null c 1 3
        chmod 666 $UMLROOT/dev/null
        mount --bind /proc $UMLROOT/proc
        mount --bind /sys $UMLROOT/sys
        rpm --root $UMLROOT --import http://fedora.redhat.com/about/security/4F2A6FD2.txt
        rpm --root $UMLROOT -Uvh --nodeps --force
http://download.fedora.redhat.com/pub/fedora/linux/core/6/x86_64/os/Fedora/RPMS/fedora-release-
6-4.noarch.rpm
        yum -y -c ./yum.fc6.conf -C --installroot=$UMLROOT groupinstall "Base"
        umount $UMLROOT/proc
        umount $UMLROOT/sys
else
        echo " *** Sorry, you must be root to setup a chroot environment"
        exit 1
fi
```

2. Create an file system image and install the file system

```
dd if=/dev/zero of=rootfs.img bs=1M count=1 seek=700

mke2fs rootfs.img

su

mkdir –p /mnt/loop

mount rootfs.img /mnt/loop

./yum3.sh
```

3. Then configure the guest file system: /etc/inittab & /etc/fstab & /dev

4. Install UML modules with

```
cd ~/uml/linux-2.6.16
make modules_install ARCH=um INSTALL_MOD_PATH=$UMLROOT
```

Note: to resize a root file system, use the following command sequence:

```
e2fsck –f  rootfs.img
dd if=/dev/zero of=rootfs.img bs=1M count=1 seek=<newsize> conv=notrunc
resize2fs –p rootfs.img
e2fsck –f rootfs.img
```

Using linux installation disk

You can also build your UML rootfs from a linux installation disk as shown in

http://uml.jfdi.org/uml/Wiki.jsp?page=BuildingRootFileSystems

http://www.tldp.org/HOWTO/Bootdisk-HOWTO/buildroot.html

1. First create the empty file that we will use for our disk image. Decide how large you want your disk image to be. Example: If you want a 500Mb disk

```
dd if=/dev/zero of=~/disk.img bs=1M count=1 seek=500
```

2. Now you can use and initrd file from any linux distribution installer you want as it on a real disk.

```
./linux mem=128M ubd0=initrd ubd1= disk.img ubd2r=/dev/cdrom rw
```

3. After the installation you will load the disk image as for real device.

```
./linux mem=128M ubdb= disk.img root=/dev/ubdb1
```

If you experience trouble with hardisk names try fake_ide fake_hd on the command line.

```
./linux mem=128M ubdb= disk.img root=/dev/ubdb1 fake_ide fake_hd
```