Thursday, 12 December 2002
Umeet 2002

Ismael Briones Vilar
ismak@inkatel.com
http://www.inkatel.com

An Introduction to User Mode Linux

I - Introduction

This is a short introduction to User Mode Linux (aka UML). Basically is a mix of the documentation of the project, that you can find in the project's web (***http:/user-mode-linux.sourceforge.net***).
I am going to explain what is UML, some descriptions of how people are using User-Mode Linux, its capabilities and how you can install and run UML.

1.1 - Run Linux inside itself

We can read a brief description of what is UML in the web of the project (http://user-mode-linux.sourceforge.net):

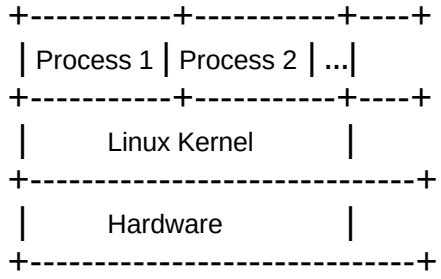
"*User-Mode Linux is a safe, secure way of running Linux versions inside itself. It gives you a virtual machine that may have more hardware and software virtual resources than your actual, physical computer. Disk storage for the virtual machine is entirely contained in a single file on your physical machine. You can provide to the virtual machine only the hardware access you want it to have an with properly limited access, nothing you can do on the virtual machine can damage your real computer, or its software.*"

That sound fun. We can have a running linux inside our linux, and we can do what we want on it, because it's like another system. UML allows you to run a Linux kernel as a user process under a normal Linux kernel.

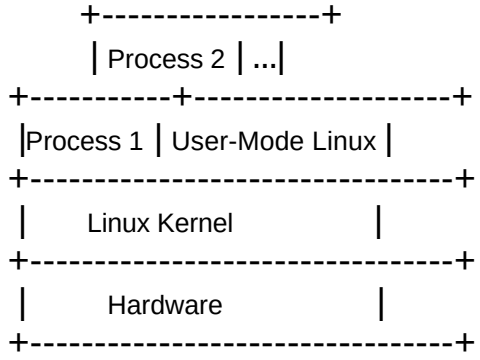
Basically, uml is the port of the Linux kernel to Linux system call interface rather than to a hardware interface. What does it mean?. It treat Linux as a platform to which the kernel can be ported, like platforms such as Intel, Alpha, Mips, etc... All of the devices accesibles inside the virtual machine are virtual and UML support the full range of devices suported by a Linux box, like:

- Console and serial lines.
- Block devices.
- Network devices.

Normaly, the Linux Kernel talks straight to the hardware (network card, video card, sound card....) and any program that run in the system ask the kernel to operate the hardware.



The UML kernel doesn't talk to the hardware, it talks to a 'real' Linux Kernel (the kernel of the system where we are running our UML) like any other program. That means that programns can run inside UML as if they were running in a normal kernel, like so:



Some of the benefits of UML are:

- If the user mode linux crash, the system running uml is still fine and this system can't be damage by an uml crash.
- The user mode linux can run as non-root user
- uml run like another process, so you can debug it
- you can use it for testing new aplications, new kernels, new distributions (run different distributions simultaneously)

II - What are people using it for?

virtual hosting

We can use uml for Virtual hosting. UML provides us with a complet Linux System, so we can run anything than can be run in our host. An example of a virtual hosting is **usermodelinux.org**, administrated by David Coulson, which is running in a user mode linux system.

Kernel development and debugging

It is a very good method of kernel debugging. If our uml system crash, the system running uml is still fine, our system will not be damaged.
We can use some debug programs like gdb, gprof and gcov. The development of drivers is more efficient, because we haven't to reboot the machine, wich decrease the development time.

Process debugging

UML can be used to debug user-level processes. If we want to debug some process, we can launch UML, set a breakpoint on the system call, and run the program.

Safely playing with the latest kernels

We can use UML to safely probe the latest kernel, so if the kernel contains any bugs, like file corruption bugs, it can't hurt any important data outside of the UML

Trying out new distributions

The filesystem of the UML is completely contained inside a file, so we don't need to use an entire disk partition for it. We can find a number of ready-to-go root filesystems loaded with various distributions in the web of UML (***http://sourceforge.net/project/showfiles.php?group_id=429***) or we can make our filesystem from the scratch. Later we will see an example with the debian distribution.

Education

It's usefull for student than need a dedicated machine, to teach OS development, network administration, and more general system administration.

Experimental development

Our virtual machine can run with more devices than the physical system running UML, so we can have a virtual system with more memory, mode devices and with more processors. We can development and testing of hardware capabilities even when we don't have the relevant hardware

Poking around inside a running system

We have a complete OS running outside UML, so we can use it to "look inside" this kernel, that are impossible for a native kernel.

As a secure sandbox or jail

Processes running inside uml have no access to the system running uml, so some malicious programs running inside uml can not damage our real system.

Virtual networking

We can use the network in a running uml. We can setup a virtual network if we want to test experimental services. Later I will explain how to setup a uml with network support.

As a test environment

Testing some software requires booting the machine, so with uml we can avoid this loss of time. We can automate this testing. There is a a small perl module implementing a UML object which provides methods to boot a virtual machine, log in to it, run commands, and shut it down. We can find it in the download page (http://sourceforge.net/project/showfiles.php?group_id=429)

Disaster recovery practice

If you want to know what happen whe you execute rm -rf / ;-) , or you want to practice recovering from a disaster, we can use uml as a practice box. It can be fun :-)

A Linux environment for other operating systems

UML only runs on Linux right now, but there are some projects to port uml to other OS, so we can have an entire linux environment in other OS

Now that we know what is user mode linux and how i can use it, i am going to explain some practical examples of uml. How to install, from scratch, a running uml based on Debian

III - A Practical Example

We need a UML kernel and a root filesystem to boot it on. We can get the kernel installing the .rpm or .deb package, or patching the kernel and compiling our UML kernel.
The .rpm and .deb packages also provide a set of userspace tools, kernel modules, and documentation.

Source Installation

If you want to build UML from source, you have to download the patch an apply it to the appropiate Linux source kernel:

```
cd linux-2.4.19
patch -p1 < uml-patch-2.4.19-37
```

Compiling the user mode kernel is just like compiling any other kernel. This is the process. After patching the kernel we have to compile it:

make menuconfig ARCH=um (you can use make xconfig and make config)

The default kernel configuration works as well, so you don't have to change anything. If you can change something. Its probably nothing will be damaged. Now we have to compile our new kernel:

make linux ARCH=um

After compilation you have the user mode kernel, named linux, in the top directory of your source tree.

Now we are going to compile and install the kernel modules. We compile the modules in the same way as a native kernel with the exception of the 'ARCH=um':

make modules ARCH=um

You can install them by using ftp, uploading these to our UML, or we can mount our root filesystem and copy them in the appropiate directory:

mount root_fs_file mount_point -o loop
make modules_install INSTALL_MOD_PATH=`pwd`/mnt ARCH=um
umount mnt

this will use the kernel build process to install our modules in our uml root filesystem. If we want to use the uml utilities, we have to get the source code and compie it.

Debian installation

Ok, now i am going to explain the easy way of getting a running uml, with debian of course ;-)

First, the woody version of debian has the next uml packages:

user-mode-linux
user-mode-linux-doc
uml-utilities

We have to install the user-mode-linux package:

apt-get install user-mode-linux (the uml-utilities package will be installed automatically by package dependencies)

We said that the root uml filesystem will be a file, so we are going to create the appropriate file:

dd if=/dev/zero of=root_fs_woody bs=1M count=100

By this way we are creating a file of 100 megas size. We can put the size we want.

This file may have an ext2 filesystem. We use *mke2fs* to create the ext2 filesystem:

/sbin/mke2fs -Fq root_fs_woody

Ok, we have our root filesystem file, now we are going to mount it and create our filesystem structure:

mkdir /mnt/uml
mount root_fs_woody /mnt/uml -o loop

We get the base system package of our Debian version (i have use the latest Debian version: woody):

wget ftp://ftp.debian.org/debian/dists/woody/main/disks-i386/base-images-current/basedebs.tar

Install the base system in our file:

debootstrap --unpack-tarball /PATH/basedebs.tar woody /mnt/uml/

Now we have to configure some files of our uml system:

cd /mnt/debinst

vi ***etc/fstab***

```
# /etc/fstab: static file system information.
#
# file system    mount point   type    options              dump pass
/dev/ubd0        /             ext2    defaults             0    0
proc             /proc         proc    defaults             0    0
```

vi ***etc/inittab***
we comment the above lines, so only an xterm will be launched when the system boot.

```
1:2345:respawn:/sbin/getty 38400 tty1
#2:23:respawn:/sbin/getty 38400 tty2
#3:23:respawn:/sbin/getty 38400 tty3
#4:23:respawn:/sbin/getty 38400 tty4
#5:23:respawn:/sbin/getty 38400 tty5
#6:23:respawn:/sbin/getty 38400 tty6
```

We create an empty source list:

touch etc/apt/sources.list

OK, now we can launch our uml kernel:

linux ubd0=root_fs_woody devfs=nomount rw

If we want with cdrom support:

linux ubd0=root_fs_woody ubd2r=/dev/cdrom devfs=nomount rw

What happen with the network support?. It is the last part of the conference.

User Mode Linux Networking

We need the TUN/TAP device module, sou we need to recompile the kernel (not the uml kernel) with this module supported.

When we have TUN/TAP device as module, we insert this in the kernel:

insmod tun

Create the TUN/TAP interface:

tunctl -u uiduser

where *uiduser* is the uid of the user running the uml.

Configure the interface:

ifconfig tap0 **ip_system** *netmask* **netmask_system** *broadcast* **broadcast_system**

where **ip_system, netmask_system** and **broadcast_system** are the network parameters of our real system (the system running uml)

Now we configure the routing table. We have to allow forwarding and manually create an ARP address mapping entry for the uml host:

bash -c 'echo 1 > /proc/sys/net/ipv4/ip_forward'
route add -host **uml_ip** *dev tap0*
bash -c 'echo 1 > /proc/sys/net/ipv4/conf/tap0/proxy_arp'
arp -Ds **uml_ip** *eth0 pub*

uml_ip must be the ip of the uml network interface.

/dev/net/tun device must have write permissions for the user running uml:

*chgrp **gid** /dev/net/tun*
chmod 660 /dev/net/tun

Our system is ready to run uml with network support:

linux ubd0=root_fs_woody devfs=nomount rw eth0=tuntap,tap0

Now you have to configure your network parameters in the uml system (ip, gateway, dns,...) and you will have a running uml with network support.

The only thing to do now is to enjoy with it ;-).