Running virtualized native drivers in User Mode Linux

V. Guffens, G. Bastin

UCL/CESAME (Belgium)

USENIX’05 / Freenix track
Anaheim, USA, June 10-15, 2005
Outline

- Overview of User Mode Linux (UML)
- A wifi layer for UML
  - Principle
  - Architecture
- Applications
  - wireless setup emulator
  - Teaching tool
PART I

Overview of User Mode Linux
User Mode Linux

- Mainly developed by Jeff Dike
- Started in February 1999 (Registered at sourceforge in November 1999)
- UML architecture is described in papers found on the UML kernel home page (OLS’01, OLS’02)
- Integrated in the official Linux 2.6.9 tree
Overview of User Mode Linux

- UML runs as a process

Diagram:
- UML
- Hardware: CPU, disks, networks, terminal, ...
- Generic kernel
- Drivers
- Architecture Layer
- ps, netscape, ls
Overview of User Mode Linux

- Virtualised kernel, new Linux architecture

![Diagram showing the architecture of User Mode Linux](image)
Overview of User Mode Linux

- No attempt to run an unmodified OS

```
Hardware : CPU, disks, networks, terminal, ...
```

```
Drivers | Architecture Layer
-------|---------------------
```

```
Generic kernel
```

```
UML architecture | UML Drivers
```

```
ls  ps  netscape
```

```
ls  ps  netscape
```

```
Generic kernel
```

```
UML
```

```
```
Achieving virtualization

- System calls: 2 modes
  - Tracing thread mode (tt)
  - Separate Kernel Address Space (skas), requires a patch on the host

- Hardware is emulated
  - UML block devices associated with a file on the host which contains a filesystem
  - Interrupts are replaced by signals
  - Network devices use a hub daemon on the host OR ethertap, ...
System calls

- **tt mode**
  - One process on the host per process on UML + tracing thread process
  - use `ptrace` syscall to intercept the UML syscall, nullify it run the syscall handler in UML kernel

- **skas mode**
  - Only 4 process/UML on the host
  - use a `/proc/mm` interface on the host to change address space
Hardware emulation, example

```
uml-gw:~# cat /proc/interrupts

    CPU0
0:  2147 SIGVTALRM timer
2:   40 SIGIO console
3:    0 SIGIO console-write
4:  4906 SIGIO ubd
9:    0 SIGIO mconsole
10:   0 SIGIO winch, winch
11:   38 SIGIO write sigio
```
PART II

A wifi layer for UML
UML-wifi Principle

- UML is used to create virtual machines
- Machines are interconnected through a simulator server
Needed components 1/2

Wireless Network interface with wireless extension in UML

- Write a specific UML driver similar to what exists now in UML
- Use an existing wireless driver and virtualize it
Needed components 1/2

Wireless Network interface with wireless extension in UML

- Write a specific UML driver similar to what exists now in UML
- Use an existing wireless driver and virtualize it

Use hostap driver
Needed components 2/2

**Physical layer**

- Forward the packets from nodes to nodes
- Drops the packets when needed (probability loss model)
- Graphical display
  - Visualise what happens in the network
  - Easily create an arbitrary topology
The hostap driver (Jouni Malinen)

- Supports multiple hardware type
  - PCI, PCMCIA → Add a UML layer
- Supports a host AP mode in software
Inserting the hostap driver in UML

- unresolved PCI related symbols in hostap_pci

$ nm hostap_pci.o | grep pci | grep U
  U pci_disable_device
  U pci_enable_device
  U pci_register_driver
  U pci_restore_state
  U pci_save_state
  U pci_set_power_state
  U pci_unregister_driver
+ writew and readw
Inserting the hostap driver in UML

- No PCI bus in UML
- Add a new virtual bus in UML: netbus
- Replace the PCI-dependent code of the hostap driver
Inserting the hostap driver in UML

$ nm hostap_uuml.o | grep netbus | grep U
  U netbus_finish_interrupt
  U netbus_read_interrupt
  U netbus_readw
  U netbus_recv
  U netbus_register_device
  U netbus_request_irq
  U netbus_send
  U netbus_unregister_device
  U netbus_writew
Interconnection with UML

Hostap driver can be inserted in UML

Need to act on an emulated device
Device emulation

- Simulator server written as a tcp server in QT/C++
- Each device is an object instantiated when a connection is established
- Driver may read and write word in device memory, status register is emulated
- A second tcp connection is used to send interrupt requests
Physical layer emulator

- Each device has a 2D physical position \((x, y)\)
- Empirical and theoretical models are available for path lost against distance
- Packet error rate may be calculated with Signal-to-Noise ratio depending on the digital modulation
- Include a mobility model
Simulator architecture

- UML machine
  - hostap driver
  - tcp client
- Virtual wireless card
- Wireless network
- Physical layer emulation
- Network visualisation window
- Simulation server
Demo

See the video

Visualisation toolbar

Mobile nodes

Fixed nodes
Applications
A testbed environment 1/4

- Testing Ad hoc On-Demand Distance Vector Routing
- Setup runs for hours (with mobile nodes)
- Connectivity is broken
A testbed environment 2/4

Troubleshooting

Route Table at uml6

<table>
<thead>
<tr>
<th>IP</th>
<th>Seq</th>
<th>Hop Count</th>
<th>Next Hop</th>
</tr>
</thead>
<tbody>
<tr>
<td>192.168.0.2</td>
<td>1</td>
<td>1</td>
<td>192.168.0.2</td>
</tr>
<tr>
<td>192.168.0.5</td>
<td>1</td>
<td>1</td>
<td>192.168.0.5</td>
</tr>
<tr>
<td>192.168.0.6</td>
<td>1</td>
<td>0</td>
<td>192.168.0.6</td>
</tr>
</tbody>
</table>
A testbed environment 3/4

- Troubleshooting

Route Table at uml3
----------------------------------------------
<table>
<thead>
<tr>
<th>IP</th>
<th>Seq</th>
<th>Hop Count</th>
<th>Next Hop</th>
</tr>
</thead>
<tbody>
<tr>
<td>192.168.0.7</td>
<td>1</td>
<td>1</td>
<td>192.168.0.7</td>
</tr>
<tr>
<td>192.168.0.6</td>
<td>1</td>
<td>1</td>
<td>192.168.0.6</td>
</tr>
<tr>
<td>192.168.0.5</td>
<td>1</td>
<td>1</td>
<td>192.168.0.5</td>
</tr>
<tr>
<td>192.168.0.3</td>
<td>1</td>
<td>0</td>
<td>192.168.0.3</td>
</tr>
</tbody>
</table>
----------------------------------------------
A testbed environment 4/4

Troubleshooting

- transmission power of UML3 > UML6
- Classical asymmetric link problem
Development and testing

- All the software available in Linux may be used in a wireless environment
- Easily develop and test new solution
- example: name resolution in MANET (draft-engelstad-manet-name-resolution-01)
- Uses a modified proxy dns server (dnrd)
A teaching tool 1/2

- Study the interactions between a driver and the kernel (ex. sending a command)

```
enqueue(struct cmd)
cmd_issue()
```

```
driver
prism2_interrupt()
dequeue(cmd)
cmd-&gt;callback()
```

```
outw
interrupt
```

```
card
acknowledge
```
A teaching tool 2/2

(1) writew(0x10,0x4)  
write param0 in PARAM0_OFF(0x4)

(2) writew(0x0,0x8)  
write parma1 in PARAM1_OFF(0x8)

(3) writew(0x10b,HFA384X_CMD_OFF)  
write the command in command register

the frame + header previously stored in card memory at address param0:param1 (0x1000) is sent

(4) Interrupt evStat=0x18,inten=0xe09f  
sent the interrupt to the processor

(5) readw(0x60)  
read the event status register

(6) readw(0x64)  
verify if interrupt enable

(7) writew(0x10,HFA384X_EVACK_OFF)  
acknowledge cmd

(8) Interrupt evStat=0x8,inten=0xe09f  
acknowledge alloc

(9) writew(0x8,HFA384X_EVACK_OFF)
Conclusion and future work

Conclusion
- Highly realistic simulations
- Good teaching and development tool

Future work
- Add more features (AP mode, rate limitation, ...)
- More stable code
Conclusion and future work

- Conclusion
  - Highly realistic simulations
  - Good teaching and development tool

- Future work
  - Add more features (AP mode, rate limitation, ...)
  - More stable code

Thank you!
Conclusion and future work

- Conclusion
  - Highly realistic simulations
  - Good teaching and development tool

- Future work
  - Add more features (AP mode, rate limitation, ...)
  - More stable code

Thank you!