Advanced virtualization techniques for FAUmachine

Hans-Jörg Höxer  Volkmar Sieh  Martin Waitz

Department of Computer Science 3: Computer Architecture
Friedrich-Alexander-University Erlangen-Nuremberg

September 2004
Outline

1. Introduction
2. Just In Time Compiler
3. Host Kernel Support
Outline

1. Introduction
2. Just In Time Compiler
3. Host Kernel Support
Many Different Virtualization Projects

- Commercial: VMware, Virtual PC, Simics, ...
- Open Source: bochs, plex86, QEMU, PearPC, FAUmachine, ...
- partial virtualization: UML, VServer, ...
Motivation: Fault injection

UMLinux started as a user mode Linux (different to UML)

Moved to a hardware simulator with minimal changes in the guest system

Now called FAUmachine
Goals of FAUmachine

- Complete simulation of a PC
- Simulator runs in user mode
- No need to patch host kernel
- Efficient
- No performance penalty
- Privileged instructions and privilege level changes need special care
- Examples: Hardware support in S390, vm86
Memory: Mapped Files

- Files to represent the physical memory
- Process’ address space to represent virtual memory
- mmap(2) to simulate MMU
- Only 3GB are available in Linux
Peripherals: Simulated

- Hardware is represented by software
- Input/output is mapped to function calls
- Simulated hardware can interact with the host system:
  - hard disk content is stored in a file
  - video signal is displayed in a window
  - sound is sent to real sound card
Differences Between User And Kernel Mode

- Different memory mappings
  - Only the kernel can access all the physical memory
- Some instructions are only available in kernel mode
  - All hardware access
  - Processor configuration
- Some instructions behave differently on i386
Virtualization of User Mode Code

- Code consists of unprivileged instructions
- Simulator has to handle user/kernel mode transitions
- Traps either provoke a signal or a real host system call
  - Can be detected by ptrace(2) or a special kernel extension
Virtualization of Kernel Mode Code

- Code contains many privileged instructions
- Those cannot be executed in user mode
- A JIT compiler is used to generate code that can be executed directly
Outline

1. Introduction
2. Just In Time Compiler
3. Host Kernel Support
Direct execution of kernel code not possible in user mode
C implementation of every instruction
Simulator works on a shadow copy of the CPU state

\[ \text{inb imm}_8 \Rightarrow \text{regs->al} = \text{host_bus_inb}(	ext{instp->imm}_8); \]
Switching Between Simulation And Direct Execution

- Direct execution is not possible all the time
- Simulation is slow

Solution:
- Only use simulation when it is necessary
- Switch back to direct execution as soon as possible

Problems:
- Real CPU state and the shadow copy have to stay in sync
- How/when to activate the simulator?
pushf/popol

- available both in user and kernel mode
- but with different semantics

```
0 0 0 0 0 0 0 0 0
0 0 ID VIP VIF AC VM RF
0 NT IOPL OF DF IF TF
SF ZF 0 AF 0 PF 0 CF
```
### pushf/popf

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>ID</td>
<td>VIP</td>
<td>VIF</td>
<td>AC</td>
<td>VM</td>
<td>RF</td>
</tr>
<tr>
<td>0</td>
<td>NT</td>
<td>IOPL</td>
<td>OF</td>
<td>DF</td>
<td>IF</td>
<td>TF</td>
<td></td>
</tr>
<tr>
<td>SF</td>
<td>ZF</td>
<td>0</td>
<td>AF</td>
<td>0</td>
<td>PF</td>
<td>0</td>
<td>CF</td>
</tr>
</tbody>
</table>

- available both in user and kernel mode
- but with different semantics
- some bits only available to kernel
Detecting Instructions That Need To Be Simulated

- No hardware support to detect problematic instructions on i386
- Every instruction has to be checked before it is executed
- But: every instruction has to be checked only once
- The result can be stored in a cache
Cache

- Executable code in the cache
- Problematic instructions are replaced with special simulator calls
- Cache is filled instruction by instruction by a JIT compiler
- A special “compile-next-instruction” call is appended to the cached code
Code Transformation

```
+----------------+    +----------------+    +----------------+
| normal code    |    | problematic instr. |    | normal code    |
+----------------+    +----------------+    +----------------+
```

```
+----------------+    +----------------+
| normal code    |    | call sim         |
+----------------+    +----------------+
```

```
+----------------+
| normal code    |
+----------------+
```

```
+----------------+
| save CPU state |
| simulate instr.|
| restore CPU state|
+----------------+```
Cache Lines

- Cache is split into several cache lines
- Direct mapping between original and cached code inside of each cache line
- Hash tables to map real address to cache line
Execution in a separate cache influences Instruction Pointer (%eip)
→ call and ret have to be simulated, too
Layout of code is changed
→ Jump targets may have to be represented using more bits
1. Introduction
2. Just In Time Compiler
3. Host Kernel Support
Handling of a System Call

- System calls in the guest user mode code will be executed directly, too.
- The simulator has to intercept these system calls and redirect them to the guest kernel.
Redirection of a System Call

- A signal is delivered instead of executing the system call
- The signal handler of the simulator fakes the system call in the guest system
- The simulator code residing in the CPU process still has to be able to execute system calls to the host kernel
- System calls coming from the simulator address space must not be redirected
A special process ("tracer") uses PTRACE_SYSCALL to trace system calls executed by the CPU process.

It gets notified on system call entry and exit.

If the system call is coming from a guest process:
- System call entry: redirect system call number to getpid(2)
- System call exit: restore system call number, send signal to CPU process

Total: four context switches, several system calls.
Conversion of a system call to a signal is trivial in kernel space.

Only system calls from a guest process inside a FAUmachine CPU process have to be converted.

New system call to register FAUmachine CPU processes.

The address range of the simulator is given as parameter.
Address Space

- Real machine has a 4GB address space
- User space processes only have a 3GB address space
- 4G patch has two problems:
  - extra TLB flush on every system call → slowdown for all processes.
  - needs a fixed 16MB area for switching
- We are working on a conditional 4GB patch with a dynamic switching zone
  ⇒ Should allow us to efficiently run unmodified kernels in the future
Performance

- Benchmark: Kernel compilation
- JIT compiler has a performance impact
- Host kernel support can increase performance

![Bar Chart]

- ptrace
- kernel support

Hans-Jörg Höxer, Volkmar Sieh, Martin Waitz
Advanced virtualization techniques for FAUmachine
Conclusion

- Direct execution to increase speed
- JIT to convert kernel to user mode code
- Host kernel support can increase performance

More information is available on faumachine.org.
Thank you!

Questions?
Thank you!

Have a nice time at LinuxKongress!
Cache Line Usage

Hans-Jörg Höxer, Volkmar Sieh, Martin Waitz