Lecture 5

User-Mode Linux

Jeff Dike

Operating Systems Practical

November 7, 2012

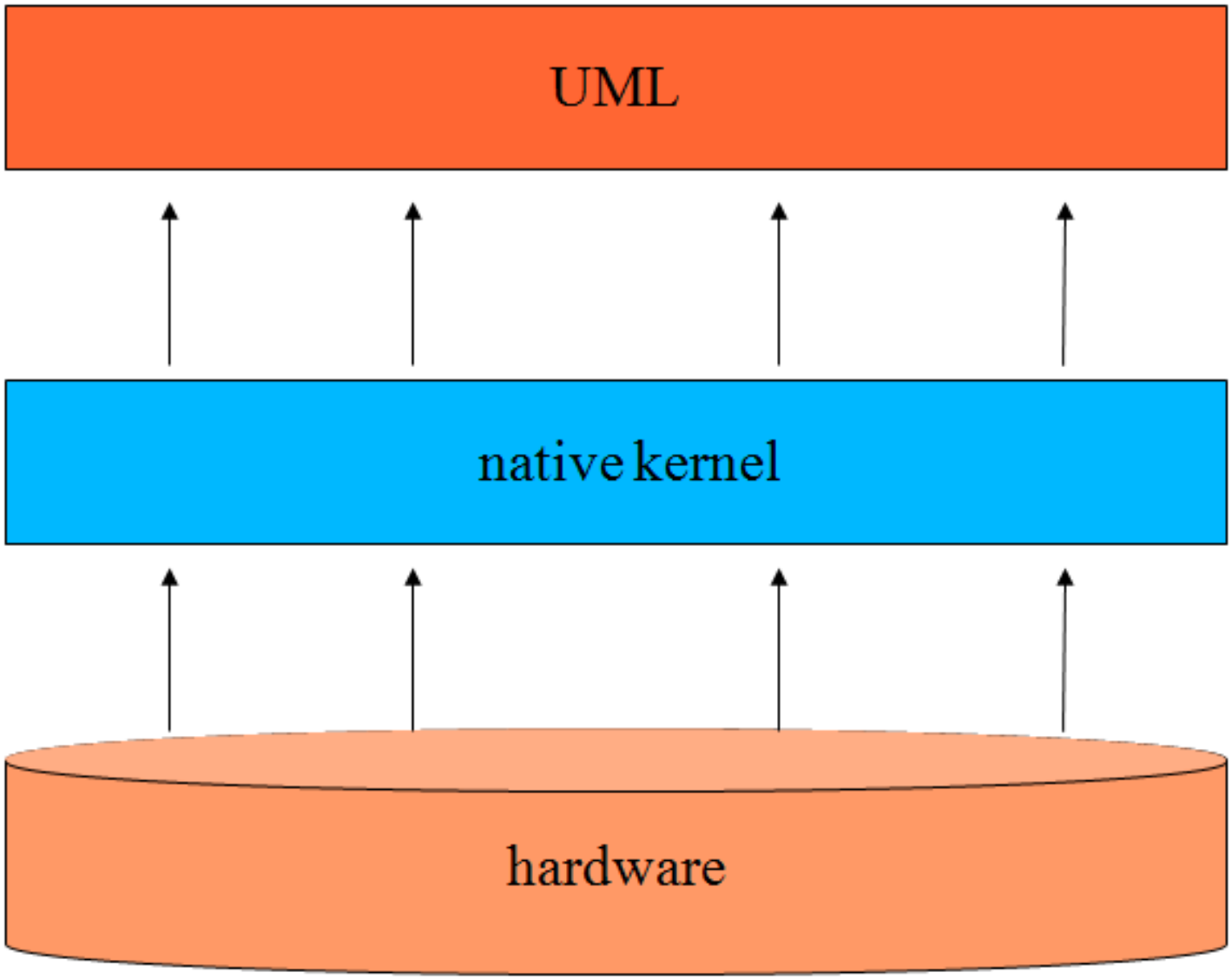
Contents

Outline

UML

- A linux kernel port on Linux
- A virtual machine in user-space
- The simulated hardware is built on top of the native kernel services
- The UML kernel is ported on top of the native's kernel system calls
- The associated code is in the arch interface (arch/um/)
- There are no drivers
- Processes run in a closed environment

UML (2)



Devices

- console
  - the main console is the one in which the UML kernel was started
  - subsequent consoles run inside an xterm
- block devices
  - emulated through files
- serial links
  - emulated through pseudo-terminals (/dev/pts/0)
- networking
  - daemon used to send Ethernet frames between virtual machines
  - can link the virtual device to the real one

- implemented using the arch interface
- the entire code is a separate architecture named “um”:

```
# ls /usr/src/linux/arch/um/
Kconfig          Kconfig.x86_64      Makefile-skas     [...]
Kconfig.char     Makefile            Makefile-tt       [...]
Kconfig.debug    Makefile-i386       Makefile-x86_64   os-Linux
Kconfig.i386     Makefile-ia64       config.release    scripts
Kconfig.net      Makefile-os-Linux   defconfig         sys-i386
Kconfig.scsi     Makefile-ppc        drivers           sys-ia64
```

- user space code need to be able to run unmodified in the virtual machine
- system calls are interpreted and run on the virtual machine
- UML runs in user space as a process

Intercepting system calls

- uses ptrace - controlled execution (gdp uses ptrace)
- one thread uses ptrace to control the other threads and processes
- the thread is notified by a system call from another thread
  - the arguments are gathered
  - redirects to kernel code running in user space for execution

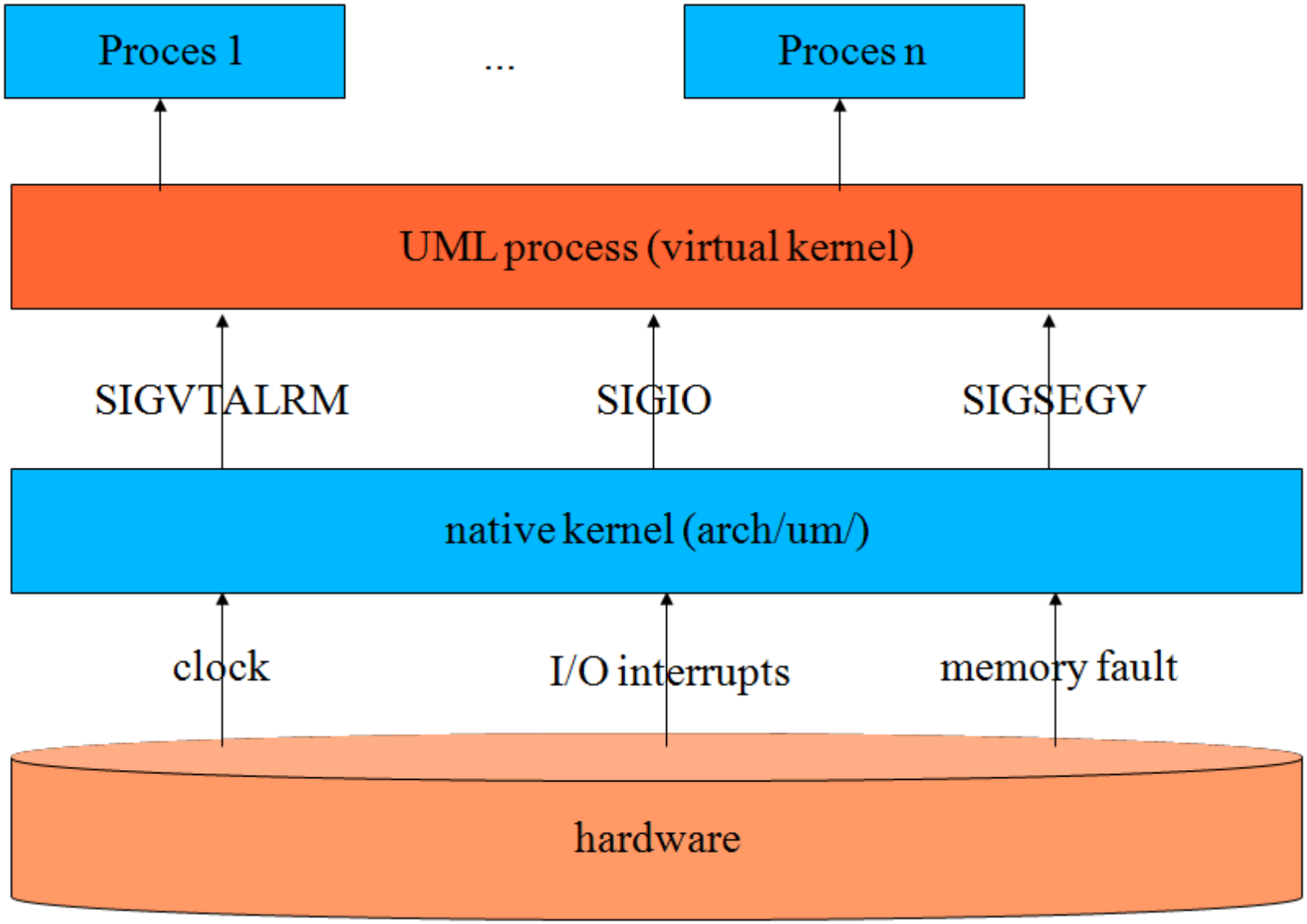
Trap

- traps are used to switch from user mode to kernel mode
- on physical systems are generated by hardware components
  - the result is forcing the processor to jump to a certain address in kernel space
- traps are implemented using Linux signals
  - SIGALRM/SIGVTALRM - clock
  - SIGIO - hardware interrupts
  - SIGSEGV - memory faults
  - the user-space kernel declares handlers for these signals

Trap (2)

- signal handlers need to run in kernel-mode (in the UML process)
  1. need to use a kernel stack
  2. need to deactivate the interception of system calls

Trap (3)


Proces 1
...
Proces n
UML process (virtual kernel)
SIGVTALRM
SIGIO
SIGSEGV
native kernel (arch/um/)
clock
I/O interrupts
memory fault
hardware

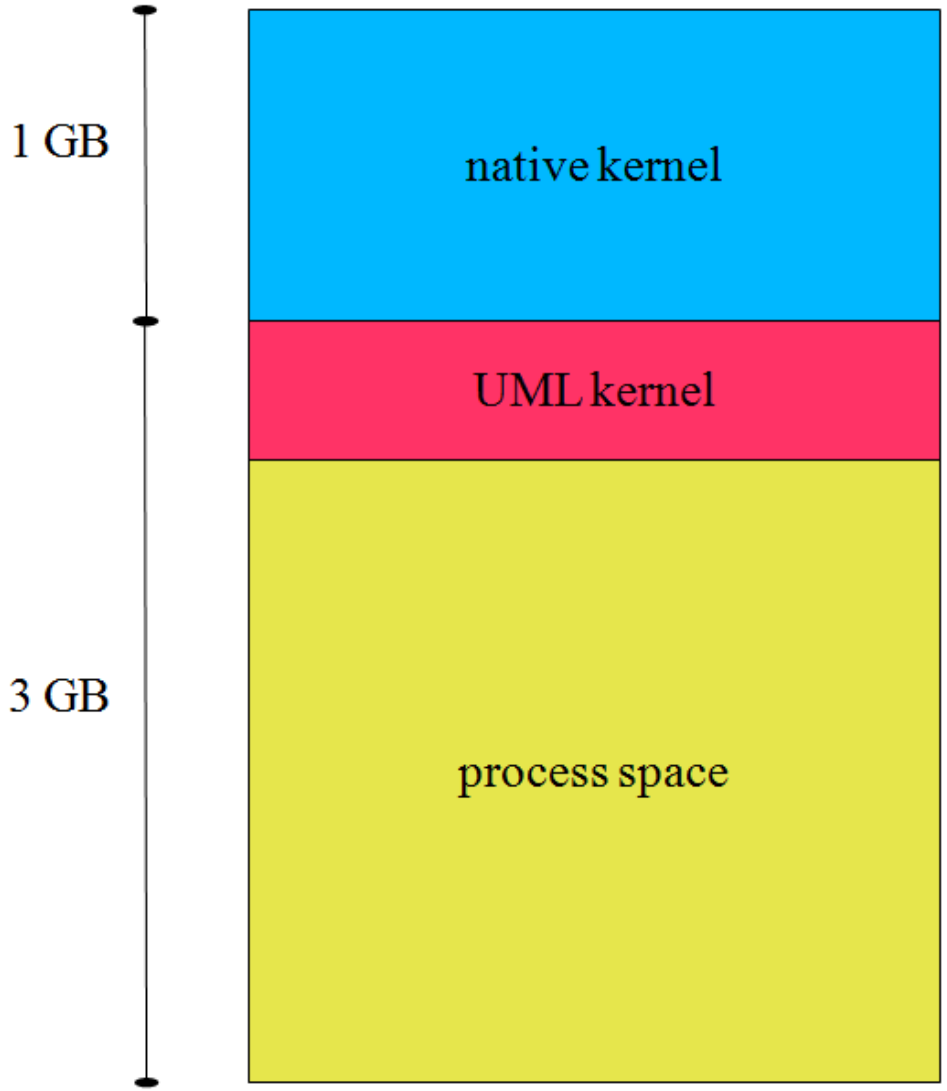
Kernel address space

- when a process enters kernel mode, it automatically changes the address space
- the UML problem?
  - the kernel and the process co-exist in the same address space - the UML process address space
- solution:
  - placing the UML kernel in a memory area that is not likely to be accessed (0xa0000000 – 0xbfffffff)
  - mmaps a file in each process space

Process address space

- Each process on the virtual machine has a real process on the physical machine associated with it
- All processes share kernel data
  - mmap a file with kernel data in the address space of each andevery process (shared segment)
- context changes are implemented using real (native) context changes
- what preempts a process?
  - clock interrupt on the native kernel
  - SIGVTALRM on the UML kernel

Address space



Initialization and shutdown

- example:
  ./linux-2.6.19-rc5 ubda=FedoraCore5-x86-root_fs mem=128M
- arguments are sent through a buffer
- init memory, start idle thread
- the monitoring thread starts intercepting
- start_kernel, mem_init, paging_init
- register and init drivers
- on shutdown, all processes and threads are killed

Processes

- for creating a new process, the generic code in the kernle calls the specific architecture code from /arch/
- in the UML case, a new process is created on the host system
- the monitoring thread is used (MT)
  - new process/thread executes init operation (handlers for SIGVTALRM, SIGIO, SIGSEGV, etc.)
  - after init it sends itself a SIGSTOP
  - the MT detects the stopping of the process and ends the system call and returns a return value specific to 'fork'
- the process is killed on the host system and the memory is freed

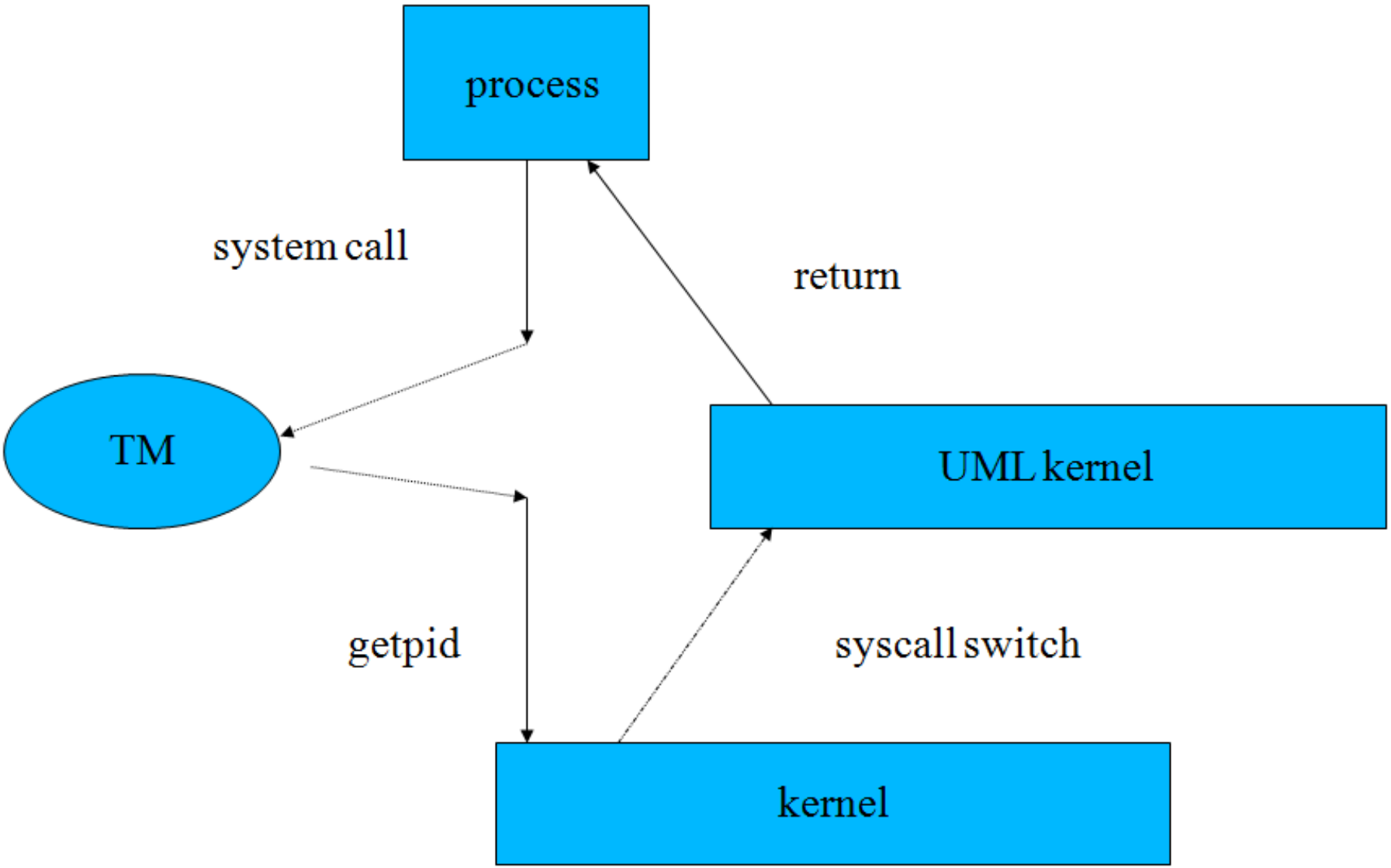
System calls

- The virtualization of the system calls is done through MT
  - system calls are redirected to the virtual kernel
  - the system call is mapped to a “getpid” on the host system

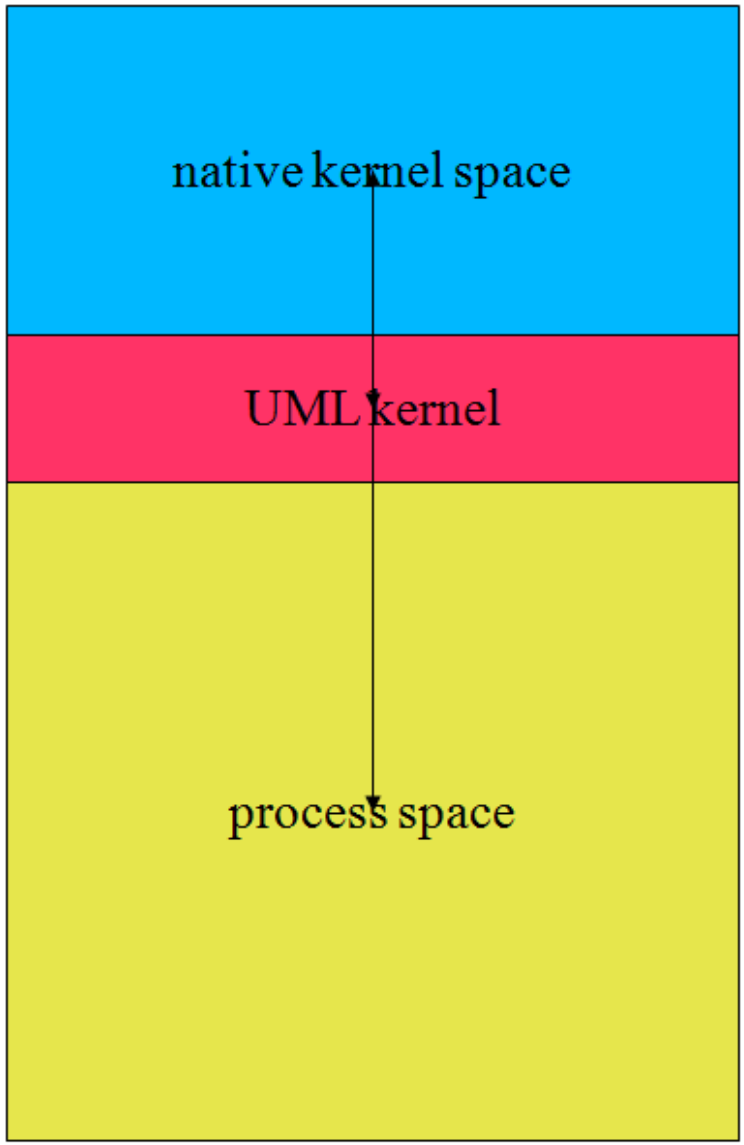
System calls (2)

- how to call the system call switch on the kernel stack?
  1. creating an execution context that positions the process at the beginning of the switch statement
  2. use a signal at the return from the kernel; the handler for this signal is the execution of the system call switch statement
- the MT is notified at the end of the system call in the MT
- the MT stores the return value in the specialized register
- the process continues to execute user level code

System calls (3)



System calls (4)



Context switch

- in the case of a process switch - a process calls schedule
- a new process is chosen and the architecture dependent code is called
- the MT is notified from the arch/um/ code
- the MT stops the process and starts the new one

Context switch (2)

- after re-planning, some pages can be swapped - but the mapping exists
- the pages are stored in a circular buffer
- after re-planning this buffer is checked and the address space is updated

Delivering signals

- delivered signals are stored in a queue in the process' task_struct
- the queue is inspected upon every exit from kernel mode
- the signal is delivered to the process running on the host kernel through SIGUSR2
- the SIGUSR2 handler executes the actual signal handler

Memory faults

- what is demand paging?
- a memory fault causes the delivery of SIGSEGV to the UML process
- the handler checks the nature of the fault: user-mode fault or kernel-mode fault
- if the page is valid - it is mapped
- otherwise SIGSEGV is sent to the user process or kernel panic
- exception: sending of a invalid pointer from user-space or kernel space
  - checks the address of the instruction that generated the fault

IRQ

- copied from i386
- for i386: the interrupt routine is called through do_irq
- for um: interrupts are simulated through SIGIO
  - the routine is chosen through the file descriptor that is associated with the device

The result

- a Linux virtual machine runs on a Linux host operating system
- native applications run un-modified on UML
- has the advantage of using the latest kernel (over other virtualization techniques)
- from 2.6 up, the um "architecture" is included in the kernel
- from 2.6 up SKAS (separate kernel address space) is used instead of MT

Applications

- kernel debugging
- isolation
- prototiping (testing on a virtual system before launching on the physical system)
- multiple environments on the same physical system

Outline

Keywords

- Linux kernel
- user-mode
- context switch
- address space
- system calls
- traps
- memory faults
- IRQ

Outline

Resources

- http://www.usenix.org/publications/library/proceedings/als00/2000papers/papers/full_papers/dike/
- https://www.usenix.org/publications/library/proceedings/als01/full_papers/dike/
- http://user-mode-linux.sourceforge.net/
- http://user-mode-linux.sourceforge.net/old/UserModeLinux-HOWTO.html
- http://www.coherenthosting.com/prj/uml/henrique/pool_h01/

Outline

Questions

?