University of BRISTOL

Computer Systems B

COMS20012

Introduction to Operating Systems and Security

bristol.ac.uk

University of BRISTOL

OS structure

bristol.ac.uk

What is an OS?

Protection Boundary

Kernel
File System Networking
Device Drivers Processes
Virtual Memory

Hardware/Software Interface

Protection Boundaries

- Multiple privilege levels
- Different software can run with different privileges
- Processors provide at least two different modes
  - User space: how “regular” program run (e.g. steam)
  - Kernel mode: How the kernel run
- The mode determine a number of things
  - What instructions may be executed
  - How addresses are translated
  - What memory locations can be accessed (through translation)

Example Intel

- Four modes
  - Ring 0: most privileged run the kernel here
  - Ring 1/2: ignored in most environment. Can run less privileged code (e.g. third party device drivers or virtual machine monitors etc.)
  - Ring 3: where “normal” processes live
- Memory is divided in segments
  - Each segment has a privilege level (0 to 3)
  - Processor maintain a current privilege level (CPL) generally the CPL of the segment containing the instruction currently executing
  - Can read/write in segment when CPL >= segment privilege
  - Cannot directly call code in segment where CPL < segment privilege

Example MIPS

- Standard two modes processor
  - User mode: access CPU registers; flat uniform virtual memory address space
  - Kernel mode: can access memory mapping hardware and special registers

Changing Protection Level

- How do we transfer control between applications and kernel?
- When do we transfer control between applications and kernel?

How?

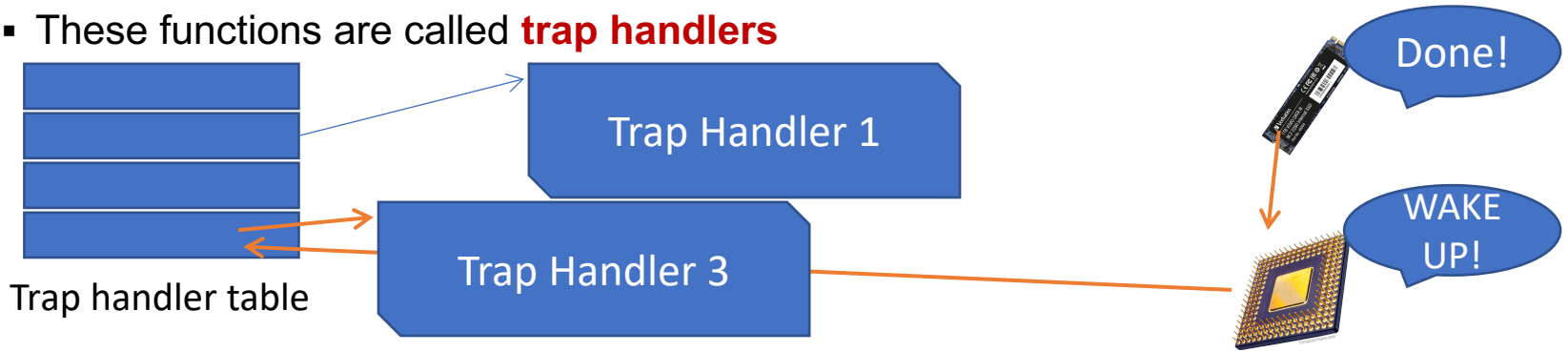
- Fundamental concept **trap**
- There is different types of trap which relate to the **when**

When?

- Sleeping beauty approach
  - Wait for something to happens to wake up the kernel
  - What might that be?
    - System calls: an application wants the operating to do something on its behalf (e.g. access some hardware)
    - Exceptions: an application does unintentionally something it should not (e.g. divide by zero, read an address it should not etc.)
    - Interrupts: An asynchronous event (e.g. I/O completion)
- Alarm clock approach
  - Set a timer that generate an interrupt when it finishes

Trap

- Each type of trap is assigned a number. For example
  - System call: 1
  - Timer interrupt: 2
  - Disk interrupt: 3
- The kernel build a table that **map trap number to function** addresses to be executed
- These functions are called **trap handlers**



Example MIPS

- MIPS has 5 distinct traps, and their addresses are hardwired
  - One each for: reset, NMI (non-maskable interrupt), fast-TLB loading and debug
  - Sys161 does not support NMI and debug
  - One for everything else (software must then do further dispatch)
- Vital information are set in specific registers. For example:
  - The EPC (program counter) tell you which instruction caused an exception
  - Cause register indicate the source of the trap
    - Interrupt
    - Exception
    - System call
    - And what specific type it was
  - Status register indicate
    - The mode the processor was when it happened (i.e. user/kernel)
    - States about which kind of interrupt are enabled

Example Intel

- Hardware register
  - Traditionally called PIC (Program Interrupt Controller)
  - … then APIC (advanced PIC)
  - … more recently LAPIC (local APIC) one per core
- Remember
  - X86 has multiple protection level (4)
  - Cannot call code in a different level
  - This needs to be handled
- Interrupt Descriptor Table (IDT)
  - Point to special objects called gate
  - Raises the CPL
  - When returning from a gate drops CPL to is original level
- First 32 gates reserved for hardware defined traps
- Remaining available to software using the INT (interrupt) instruction.
- For system calls Linux uses a single desginated INT instruction
  - Dispatch via software in the same we saw for MIPS

University of BRISTOL

Thank you

bristol.ac.uk