CS 423
Operating System Design: The Kernel Abstraction

Professor Adam Bates
Goals for Today

- **Learning Objectives:**
  - (Learn how to use the vSphere console)
  - Understand the Kernel/Process Abstraction
  - Gain a working knowledge of Mode/Context switches

- **Announcements:**
  - C4 weekly summaries! **Due Jan 25.** Week 3 is out.
  - HW0 is available on Compass! **Due Jan 25**
  - MP0 is available on Compass! **Due Jan 28**
  - **TA Office Hours change:**
    - 0207 @ ????

**Reminder:** Please put away devices at the start of class
I thwarted your C3 restriction... we cool?

Responsible for textbook?
Process concept

- A process is the OS abstraction for executing a program with limited privileges

Dual-mode operation: user vs. kernel

- Kernel-mode: execute with complete privileges
- User-mode: execute with fewer privileges

Safe control transfer

- How do we switch from one mode to the other?
Process Abstraction

Process: an **instance** of a program that runs with limited rights on the machine

- Thread: a sequence of instructions within a process
  - Potentially many threads per process (for now, assume 1:1)
- Address space: set of rights of a process
  - Memory that the process can access
  - Other permissions the process has (e.g., which system calls it can make, what files it can access)
How can we permit a process to execute with only limited privileges?
How can we implement execution with limited privilege?

- Execute each program instruction in a simulator
- If the instruction is permitted, do the instruction
- Otherwise, stop the process
- Basic model in Javascript and other interpreted languages
How can we implement execution with limited privilege?

- Execute each program instruction in a simulator
- If the instruction is permitted, do the instruction
- Otherwise, stop the process
- Basic model in Javascript and other interpreted languages

Ok... but how do we go faster?
Thought Experiment

How can we implement execution with limited privilege?
• Execute each program instruction in a simulator
• If the instruction is permitted, do the instruction
• Otherwise, stop the process
• Basic model in Javascript and other interpreted languages

Ok... but how do we go faster?
• Run the unprivileged code directly on the CPU!
A Model of a CPU

Branch Address

Select PC

New PC

Program Counter

CPU Instructions Fetch and Execute

opcode

+4
A CPU with Dual-Mode Operation

Handler PC

Select PC

New PC

Program Counter

CPU Instructions Fetch and Execute

Branch Address

New Mode

Select Mode

Mode

opcode
Privileged instructions
  • Available to kernel
  • Not available to user code
Limits on memory accesses
  • To prevent user code from overwriting the kernel
Timer
  • To regain control from a user program in a loop
Safe way to switch from user mode to kernel mode, and vice versa
Examples?

What should happen if a user program attempts to execute a privileged instruction?
How/when do we switch from user to kernel mode?

1. Interrupts
   - Triggered by timer and I/O devices
2. Exceptions
   - Triggered by unexpected program behavior
   - Or malicious behavior!
3. System calls (aka protected procedure call)
   - Request by program for kernel to do some operation on its behalf
   - Only limited # of very carefully coded entry points
How does the OS know when a process is in an infinite loop?
Hardware Timer

Hardware device that periodically interrupts the processor
  • Returns control to the kernel handler
  • Interrupt frequency set by the kernel
    Not by user code!
  • Interrupts can be temporarily deferred
    Not by user code! Interrupt deferral crucial for implementing mutual exclusion
How/when do we switch from kernel to user mode?

1. New process/new thread start
   • Jump to first instruction in program/thread
2. Return from interrupt, exception, system call
   • Resume suspended execution (return to PC)
3. Process/thread context switch
   • Resume some other process (return to PC)
4. User-level upcall (UNIX signal)
   • Asynchronous notification to user program
What is the CPU’s behavior defined by at any given moment?
What is the CPU’s behavior defined by at any given moment?
CPU State

What is the CPU’s behavior defined by at any given moment?
What is the CPU’s behavior defined by at any given moment?
What is the CPU’s behavior defined by at any given moment?
What defines the **STATE** of the CPU?

- **Program Counter**: Determines the next instruction to be executed.
- **Code Segment Offset**: Contains the program instructions.
- **Data Segment Offset**: Contains data used by the program.
- **Stack Segment Offset**: Contains the stack pointer.
- **Stack Pointer**: Points to the current stack location.
- **Data Operand**: Contains operands for instructions.
- **Operand**: Contains data for operations.
- **Current Instruction**: Represents the instruction currently being executed.
- **OpCode**: Specifies the operation to be performed.
- **Heap**: Stores data not on the stack.
- **Stack**: Stores data temporarily during program execution.

**Registers**
What’s the **STATE** of a real CPU?
The Context Switch

Save State (Context)

Load State (Context)
The state for processes that are not running on the CPU are maintained in the Process Control Block (PCB) data structure.
The Context Switch

Save State (Context)

Load State (Context)
Note: In thread context switches, heap is not switched!

**Save State (Context)**

**Load State (Context)**

Program Counter

Program instructions

Code Segment

Offset

Program Counter

OpCode Operand

Stack Segment

Stack Pointer

Stack

Data Segment

Data Operand

Heap

Registers

Register

OpCode Operand

Code Segment

Offset

Program Counter

Stack Segment

Stack Pointer

Stack

Program instructions

Stack

28
The Context Switch

Note: In thread context switches, heap is not switched!

Save State (Context)

Load State (Context)
So who does the context switch, and when??
Thread Context Switch

Note: In **thread** context switches, heap is not switched!

Solution 1: An Interrupt

Save State (Context)

Load State (Context)
CTX Switch: Interrupt

Running Thread

- Program Counter
- Code Segment Offset
- Stack Segment
- Stack Pointer
- Program instructions
- Stack

Registers

- Code Segment Offset
- Program Counter
- Stack Segment
- Stack Pointer
- Program instructions
- Stack
Interrupt

Save PC on thread stack
Jump to Interrupt handler
Save PC on thread stack
Jump to Interrupt handler

Handler
- Save thread state in thread control block
  (SP, registers, segment pointers, ...)

Thread Control Block
Save PC on thread stack
Jump to Interrupt handler

Handler
- Save thread state in thread control block
  (SP, registers, segment pointers, ...)
- Choose next thread
- Load thread state from control block
Save PC on thread stack
Jump to Interrupt handler

Handler
- Save thread state in thread control block (SP, registers, segment pointers, …)
- Choose next thread
- Load thread state from control block
- Pop PC from thread stack (return from handler)
Save PC on thread stack
Jump to Interrupt handler

Handler
- Save thread state in thread control block
  (SP, registers, segment pointers, ...)
- Choose next thread
- Load thread state from control block
- Pop PC from thread stack (return from handler)

Where does it return?
Save PC on thread stack
Jump to Interrupt handler

Handler
- Save thread state in thread control block (SP, registers, segment pointers, ...)
- Choose next thread
- Load thread state from control block
- Pop PC from thread stack (return from handler)

Where does it return?
What are some examples of context switches due to interrupts?

- **Clock Interrupt**: Task exceeds its time slice
- **I/O Interrupt**: Waiting processes may be preempted
- **Memory Fault**: CPU attempts to access a virtual memory address that is not in main memory. OS may resume execution of another process while retrieving the block, then moves process to ready state.
Thread Context Switch

Save State (Context)

Load State (Context)

Note: In thread context switches, heap is not switched!

Global Variables

Solution 2: Voluntary yield()
Running Thread

Program instructions

Code Segment
Offset

Program Counter

Stack Segment

Stack Pointer

Stack

Registers

Code Segment
Offset

Program Counter

Stack Segment

Stack Pointer

Stack

Registers
yield()

Save PC on thread stack
Jump to yield() function
Save PC on thread stack
Jump to yield() function

yield()
- Save thread state in thread control block
  (SP, registers, segment pointers, ...)

Thread Control Block
CTX Switch: Yield

- Save thread state in thread control block (SP, registers, segment pointers, ...)
- Choose next thread

Save PC on thread stack
Jump to yield() function

Program instructions
Code Segment
Offset
Program Counter
Stack Segment
Stack
Stack Pointer
Register

Program instructions
Code Segment
Offset
Program Counter
Stack Segment
Stack
Stack Pointer
Register

Thread Control Block

yield()
Save PC on thread stack
Jump to yield() function

yield()
- Save thread state in thread control block
  (SP, registers, segment pointers, ...)
- Choose next thread
- Load thread state from control block
Save PC on thread stack
Jump to yield() function

yield()
- Save thread state in thread control block
  (SP, registers, segment pointers, ...)
- Choose next thread
- Load thread state from control block
- Pop PC from thread stack (return from handler)
CTX Switch: Yield

Save PC on thread stack
Jump to yield() function

yield()
- Choose next thread
- swapcontext()
Where is the Scheduling Policy?

Save PC on thread stack
Jump to yield() function

yield()
- Choose next thread
- swapcontext()
Where is the Scheduling Policy?

Save PC on thread stack
Jump to yield() function

yield()
- NextThreadID = scheduler()
- swapcontext()

Maintains a sorted queue of ready threads