UNIVERSITY of WISCONSIN-MADISON
Computer Sciences Department

CS 537
Introduction to Operating Systems

Andrea C. Arpaci-Dusseau
Remzi H. Arpaci-Dusseau

Processes

Questions answered in this lecture:

What is a process?

How does the dispatcher context-switch between processes?

How does the OS create a new process?

What is a Process?

Process: An execution stream in the context of a process state

Execution stream

1) stream of executing instructions
2) Running piece of code
3) Sequential sequence of instructions
4) "thread of control"

Process state

- Everything that the running code can affect or be affected by
- Registers
  - General-purpose, floating point, status, program counter, stack pointer
- Address space
  - Everything process can address through memory
  - Represented by array of bytes
  - Heap, stack, and code

Processes vs. Programs

A process is different than a program

- Program: Static code and static data
- Process: Dynamic instance of code and data

No one-to-one mapping between programs and processes

- Can have multiple processes of the same program
  Example: many users can run "ls" at the same time
- One program can invoke multiple processes
  Example: "make" runs many processes to accomplish its work

Processes vs. Threads

A process is different than a thread

Thread: "Lightweight process" (LWP)

- An execution stream that shares an address space
- Multiple threads within a single process

Example:

- Two processes examining memory address 0xffa84264 see different values (i.e., different contents)
- Two threads examining memory address 0xffa84264 see same value (i.e., same contents)

System Classifications

All systems support processes, but the number varies

Uniprogramming: Only one process resident at a time
- Examples: First systems and DOS for PCs
- Advantages: Runs on all hardware
- Disadvantages: Not convenient for user and poor performance

Multiprogramming: Multiple processes resident at a time (or, multitasking)
- Note: Different than multiprocessing
  - Multiprocessing: Systems with multiple processors
- Examples: Unix variants, WindowsNT
- Advantages: Better user convenience and system performance
- Disadvantages: Complexity in OS

Multiprogramming

OS requirements for multiprogramming
- Mechanism
  - to switch between processes
  - to protect processes from one another
- Policy
  - to decide which process to schedule

Separation of policy and mechanism
- Reoccuring theme in OS
- Policy: Decision-maker to optimize some workload performance metric
  - Which process when?
  - Process scheduler: future lecture
- Mechanism: Low-level code that implements the decision
  - How?
  - Process Dispatcher: today's lecture

Dispatch Mechanism

OS runs dispatch loop

```
while (1) {
    run process A for some time-slice
    stop process A and save its context
    load context of another process B
}
```

context-switch

Question 1: How does dispatcher gain control?

Question 2: What execution context must be saved and restored?

Q1: Hardware for Multiprogramming

Must differentiate application process and OS

Hardware support
- Bit in status word designates whether currently running in user or system mode
- System mode (or privileged or supervisor) allows functionality
  - Execution of special instructions (e.g., access hardware devices)
  - Access to all of memory
  - Change stack pointer
- Usage
  - Applications run in user mode
  - OS runs in system mode

Q1: Entering system mode

How does OS get control?

1) Synchronous interrupts, or traps
   - Event internal to a process that gives control to OS
   - Examples: System calls, page faults (access page not in main memory), or errors (illegal instruction or divide by zero)
2) Asynchronous interrupts
   - Events external to a process, generated by hardware
   - Examples: Characters typed, or completion of a disk transfer

How are interrupts handled?

- Each type of interrupt has corresponding routine (handler or interrupt service routine (ISR)
- Hardware saves current process and passes control to ISR

Q1: How does Dispatcher run?

Option 1: Cooperative Multi-tasking

- Trust process to relinquish CPU through traps
  - Trap: Event internal to process that gives control to OS
  - Examples: System call, page fault (access page not in main memory), or error (illegal instruction or divide by zero)
- Disadvantages: Processes can misbehave
  - By avoiding all traps and performing no I/O, can take over entire machine
  - Only solution: Reboot!
- Not performed in modern operating systems

Q1: How does Dispatcher run?

Option 2: True Multi-tasking

- Guarantee OS can obtain control periodically
- Enter OS by enabling periodic alarm clock
  - Hardware generates timer interrupt (CPU or separate chip)
  - Example: Every 10ms
- User must not be able to mask timer interrupt
- Dispatcher counts interrupts between context switches
  - Example: Waiting 20 timer ticks gives 200 ms time slice
  - Common time slices range from 10 ms to 200 ms

Q2: Process States

Each process is in one of three modes:

- Running: On the CPU (only one on a uniprocessor)
- Ready: Waiting for the CPU
- Blocked (or asleep): Waiting for I/O or synchronization to complete