CS 550:

Advanced Operating Systems

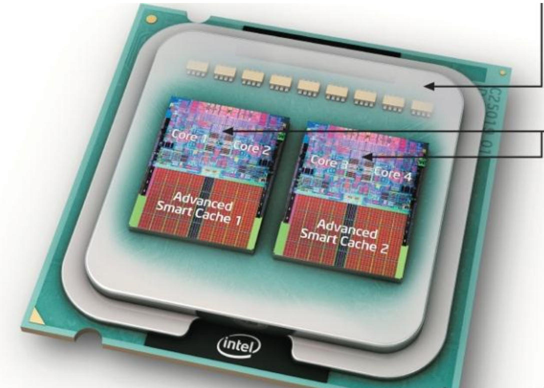
Processes and Threads

Ioan Raicu
Computer Science Department
Illinois Institute of Technology

CS 550
Advanced Operating Systems
February 10th, 2011

Review: Multicore everywhere!

- Multicore processors are taking over, *many*core is coming
- The processor is the “new transistor”
- This is a “sea change” for HW designers and especially for programmers

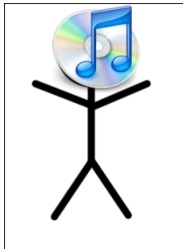


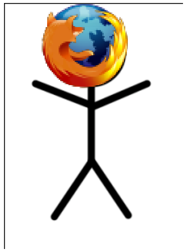
Outline for Today

- Motivation and definitions
- Processes
- Threads
- Synchronization constructs
- Speedup issues
  - Overhead
  - Caches
  - Amdahl’s Law

How can we harness (many | multi)cores?

- Is it good enough to just have multiple programs running simultaneously?
- We want per-program performance gains!





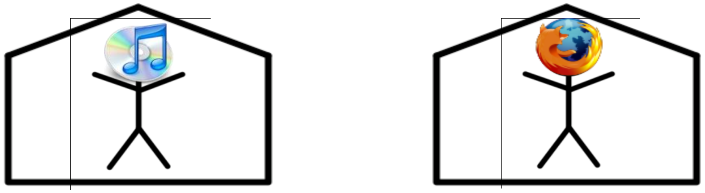
Crysis, Crytek 2007

Multiprogramming/Timesharing Systems

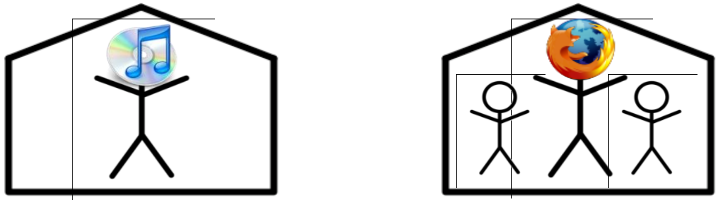
- Goal: to provide interleaved execution of several processes to give an illusion of many simultaneously executing processes.
- Computer can be a single-processor or multi-processor machine.
- The OS must keep track of the state for each active process and make sure that the correct information is properly installed when a process is given control of the CPU.
- Many resource allocation issues to consider:
  - How to give each process a chance to run?
  - How is main memory allocated to processes?
  - How are I/O devices scheduled among processes?

Definitions: threads v.s. processes

- A *process* is a “program” with its own address space.
  - A process has at least one thread!



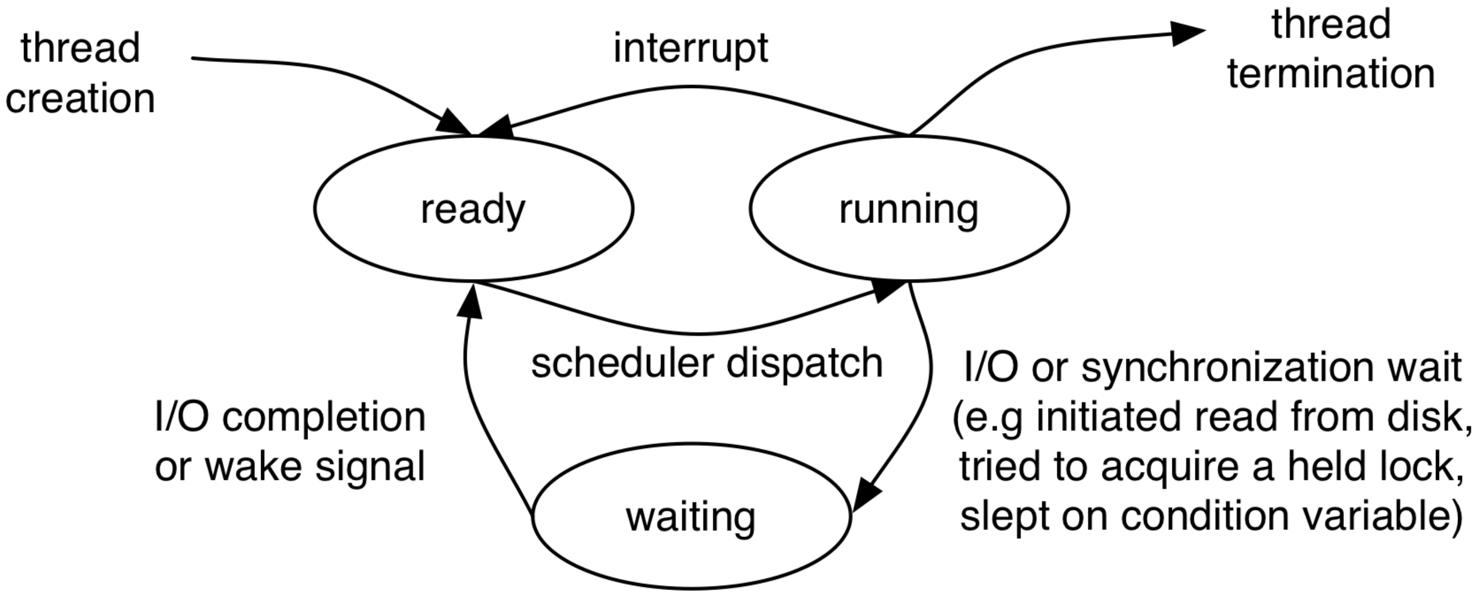
- A *thread of execution* is an independent sequential computational task with its own control flow, stack, registers, etc.
  - There can be many threads in the same process sharing the same address space



  - There are several APIs for threads in several languages. We will cover the PThread API in C.

How are threads *scheduled?*

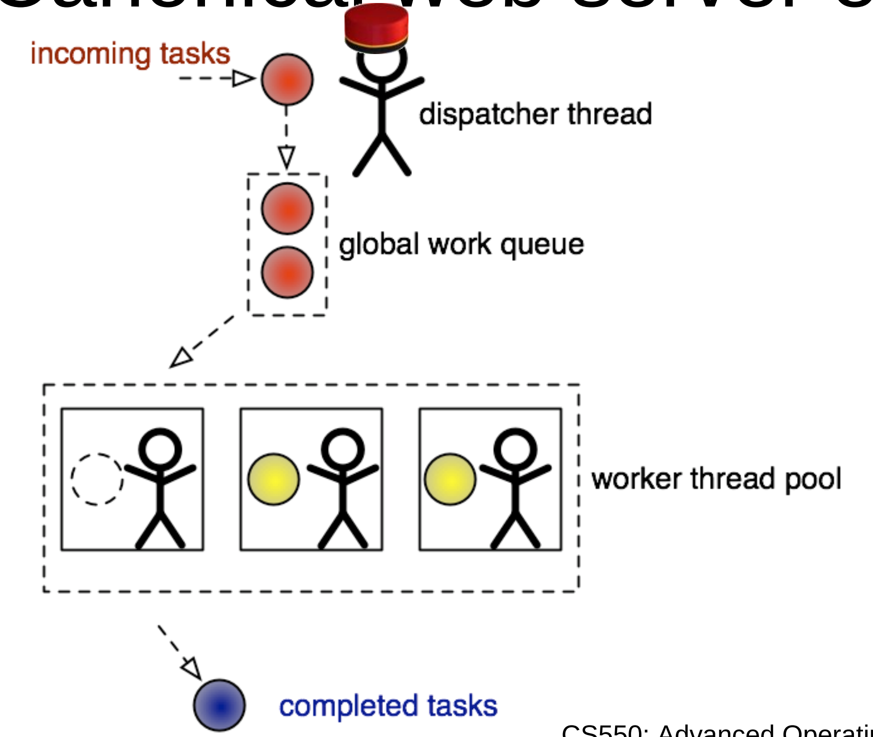
- Threads/processes are run sequentially on one core or simultaneously on multiple cores
    - The operating system schedules threads and


thread creation
interrupt
thread termination
ready
running
scheduler dispatch
I/O completion or wake signal
I/O or synchronization wait (e.g initiated read from disk, tried to acquire a held lock, slept on condition variable)
waiting

Based on diagram from Silberschatz, Galvin, and Gagne

Side: threading without multicore?

- Is threading useful without multicore?
  - – Yes, because of I/O blocking!
- Canonical web server example:



```
global workQueue;

dispatcher() {
  createThreadPool();
  while(true) {
    task = receiveTask();
    if (task != NULL) {
      workQueue.add(task);
      workQueue.wake();
    }
  }
}

worker() {
  while(true) {
    task = workQueue.get();
    doWorkWithIO(task);
  }
}
```

Questions

?