Announcements

- Send email with your presentation preference

Topic I: Concurrency

- Review of basic concepts
- Process Management as OS responsibility
  - process vs thread abstraction
- Synchronization Issues:
  - mutual exclusion & race conditions
  - deadlock & starvation
- Implementing processes & threads
- Programming models for communication
  - threads vs events

Definition: Process/Thread

- Process
  - "program in execution"
  - resources: CPU context, memory maps, open files, privileges, …; isolated
- Threads
  - CPU context (state + stack); not isolated
- "thread" is a historically recent term
  - Threads used to be called "processes"
- Q: what primitives does an OS need to provide?

User View

- Unix/C:
  - fork()/wait() vs pthread_create()/pthread_join()
- Java:
  - new Thread()
  - Thread.start()/join()
- See also [Boehm PLDI 2005]

Aside: Hybrid Models

- The "threads share everything" + "processes share nothing" mantra does not always hold
- Hybrids:
  - WEAVES allows groups of threads to define their own namespace, so they only share data they want
  - Java multitasking systems (KaffeOS, MVM): multiple "processes" may share same address space
Why use Concurrency?

- Overlap I/O and computation
  - Hide latency
- Reduce latency
  - If thread system supports preemption
- Exploit multiprocessors
  - CPU concurrency
- Software engineering reasons
  - Separation of concerns

Q: What are non-reasons to use threads?

Example Use: Threads in Servers

Resource Access

- Access to resources must be protected
- Race Condition problem
  - Definition
  - Approaches for detecting them
  - Static vs dynamic

Critical Section Problem

- Many algorithms known
  - purely software-based (Dekker’s, Peterson’s algorithm) vs. hardware-assisted (disable irqs, test-and-set instructions)
- Criteria for good algorithm:
  - mutual exclusion
  - progress
  - bounded waiting

Synchronization Abstractions

- Atomic primitives
  - e.g. Linux kernel “atomic_inc()”
- Dijkstra’s semaphores
  - \( P(s) := \text{atomic} \{ \text{while}(s\leq0) \text{ /* no op */; } s--; \} \)
  - \( V(s) := \text{atomic} \{ s++; \} \)
  - Q: what’s wrong with this implementation?
- Binary semaphores, locks, mutexes
  - Difference between mutex & semaphore
Expressing Critical Sections

Pthreads/C vs Java

pthread_mutex_t m;
…
pthread_mutex_lock(&m);
/* in critical section */
if (*) {
    pthread_mutex_unlock(&m);
    return;
}
pthread_mutex_unlock(&m);

synchronized (object) {
    /* in critical section */
    if (*) {
        return;
    }
}

Pthreads/C vs Java
Note benefits of language support

Expressing Monitors

Monitors (Hoare)

• Data Type:
  – internal, private data
  – public methods
    wrapped by Enter/Exit
  – wait/signal methods
• “Monitor Invariant”

See also Java’s insecure parallelism [Per Brinch Hansen 1999]