Final Notes

Dmitri Loguinov
Texas A&M University

April 25, 2023
Write code for an interlocked multiply that does not use mutexes (lock-free)

- Multiple threads might be calling this function at the same time

**Idea**: grab the target, multiply locally, then try to swap back into shared space

- Main caveat is another thread might have changed the target between our read and write
• **Dining savages 5.0 revisited**
  - A tribe of savages eats communal dinners from a large pot that can hold \( M \) servings of stewed missionary
  - When savage want to eat, they help themselves from the pot, unless it is empty
  - If so, they must wait for the cook to refill the pot with \( M \) new pieces of meat

```java
Semaphore semaEmpty, semaFull;
Thread cook, savage[];

Thread.start(cook); // Initialize the thread

cook.run() {
    while (true) {
        semaEmpty.Wait();
        chunksLeft = MakeFood();
        semaFull.Release(chunksLeft);
    }
}

savage[i].run() {
    AttemptEat() {
        // semaphore counts available chunks
        semaFull.Wait();
        StartEating();
        mutex.Lock();
        if (--chunksLeft == 0)
            semaEmpty.Release();
        mutex.Unlock();
    }
}
```
Semaphore Puzzles

• Dining savages 6.0
  – Doesn’t use shared counters
  – No mutexes either

• Bursty producer-consumer
  – Mutex not needed in either thread

• Better performance?
  – a) install multiple pots; b) run multiple cooks
Chapters 9-10 (scheduling), 14 (networking) were not covered in this class

Some of this material discussed in chapters 2-3
- Ready, blocked, running, suspended process states
- Dispatcher admitting and swapping processes

Main algorithms of chapter 9:
- **First-come, first-served (FCFS):** same as FIFO, no preemption (i.e., each process executes to completion)
- **Round-robin (RR):** assign fixed time slice to each process, preempt after the slice, run the next process in line
- **Weighted RR (WRR):** similar to RR, but assign weights to processes based on their type, then set slice time proportional to weights
Scheduling

- **Algorithms (cont’d)**
  - **Strict priority**: multiple queues for different priority classes, serve class i only when all higher-priority queues are empty
  - **Shortest process next (SPN)**: run process with the shortest estimated duration of execution D, no preemption
  - **Shortest remaining time (SRT)**: preemptive version of SPN
  - **Highest response ratio next (HRRN)**: response ratio is computed as $w / D$, where $w$ is the current wait time

- **Main issue**: difficult to estimate D ahead of time
- **Feedback policy**: gradually penalize long processes
  - Process starts at highest priority, but after fixed intervals of CPU time, its priority drops by one class
  - Eventually, all long processes are in the idle class
In user space, process scheduling isn’t typically feasible or useful since the OS does it better.

However, many other areas involve similar concepts:
- Amazon gets millions of requests per second, in which order to serve them to minimize response time?
- Airport gate assignment to minimize wait time, transfer delay.

Chapter 10 deals with multi-CPU scheduling:
- More complex issue related to RAM/cache locality.
- Chapter also covers real-time scheduling to guarantee hard upper bounds on slice duration.

Even more general is distributed system scheduling:
- Jobs running on multiple hosts in parallel.

more in CSCE 410
Networking

• Networks use sockets to interface with applications
  ─ Kernel APIs to open connections, transfer data
• Programming sockets is fairly easy, the interesting aspect are the underlying protocols
  ─ HTTP, DNS, SMTP, FTP, POP3, P2P: application layer
  ─ TCP/UDP: transport layer
  ─ IP: network layer
  ─ Ethernet, 802.11 wireless: data-link layer
• Homework similar to this class, multi-threaded C++
  ─ STL is allowed, programming should be simpler than here
  ─ CSCE 315 isn’t needed, although listed as a prereq