Synchronization
Classic Problems

Summer 2011
Cornell University
Today

• Which practical problems can we solve with semaphores?
• Bounded-Buffer Problem
• Readers-Writers Problem
Bounded Buffer

- Arises when **two or more threads communicate** with each other.
- And, some threads “**produce**” data and other threads “**consume**” this data.
- Real example: Production line
Producer-Consumer Problem

- Start by imagining an unbounded (infinite) buffer
  - Producer process writes data to buffer
    - Writes to In and moves rightwards
  - Consumer process reads data from buffer
    - Reads from Out and moves rightwards
    - Should not try to consume if there is no data
Producer-Consumer Problem

- Bounded buffer: size ‘N’
  - Access entry 0… N-1, then “wrap around” to 0 again
- Producer process writes data to buffer
  - Must not write more than ‘N’ items more than consumer “ate”
- Consumer process reads data from buffer
  - Should not try to consume if there is no data
Producer-Consumer Problem

- Solving with semaphores
  - We'll use two kinds of semaphores
  - We'll use *counters* to track how much data is in the buffer
    - One counter counts as we add data and stops the producer if there are $N$ objects in the buffer
    - A second counter counts as we remove data and stops a consumer if there are 0 in the buffer
  - Idea: since general semaphores can count for us, we don’t need a separate counter variable
- Why do we need a second kind of semaphore?
  - We’ll also need a mutex semaphore
Producer-Consumer Problem

Shared Semaphores: mutex, empty, full;
Init: mutex = 1; /* for mutual exclusion*/
    empty = N; /* number empty buf entries */
    full = 0;  /* number full buf entries */

**Producer**
do {
    ...
    // produce an item in nextp
    ...
    wait(empty);
    wait(mutex);
    ...
    // add nextp to buffer
    ...
    signal(mutex);
    signal(full);
} while (true);

**Consumer**
do {
    wait(full);
    wait(mutex);
    ...
    // remove item to nextc
    ...
    signal(mutex);
    signal(empty);
    ...
    // consume item in nextc
    ...
} while (true);
Readers and Writers

• In this problem, threads share data that some threads “read” and other threads “write”.

• Goal: allow multiple concurrent readers but only a single writer at a time, and if a writer is active, readers wait for it to finish
Readers-Writers Problem

- Models access to a database
  - A reader is a thread that needs to look at the database but won’t change it.
  - A writer is a thread that modifies the database
- Example: making an airline reservation
  - When you browse to look at flight schedules the web site is acting as a reader on your behalf
  - When you reserve a seat, the web site has to write into the database to make the reservation
Readers-Writers Problem

- Many threads share an object in memory
  - Some write to it, some only read it
  - Only one writer can be active at a time
  - Any number of readers can be active simultaneously
- One issue we need to settle, to clarify problem statement.
  - Suppose that a writer is active and a mixture of readers and writers now shows up. Who should get in next?
  - Or suppose that a writer is waiting and an endless stream of readers keeps showing up. Is it fair for them to become active?
- We'll favor a kind of back-and-forth form of fairness:
  - Once a reader is waiting, readers will get in next.
  - If a writer is waiting, one writer will get in next.
Readers-Writers Problem

mutex = Semaphore(1)
wrt = Semaphore(1)
readcount = 0;

\textbf{Writer}
do{
    wait(wrt);
    \ldots
    /*writing is performed*/
    \ldots
    signal(wrt);
}while(true)

\textbf{Reader}
do{
    wait(mutex);
    readcount++;
    if (readcount == 1)
        wait(wrt);
    signal(mutex);
    \ldots
    /*reading is performed*/
    \ldots
    wait(mutex);
    readcount--;
    if (readcount == 0)
        signal(wrt);
    signal(mutex);
}while(true)
Readers-Writers Notes

- If there is a writer
  - First reader blocks on `wrl`
  - Other readers block on `mutex`
- Once a reader is active, all readers get to go through
  - Which reader gets in first?
- The last reader to exit signals a writer
  - If no writer, then readers can continue
- If readers and writers waiting on `wrl`, and writer exits
  - Who gets to go in first?
- Why doesn’t a writer need to use `mutex`?
- Is the previous solution fair?
Readers-Writers Notes

- If readers are active, no writer can enter
  - The writers wait doing a P(wrt)
- While writer is active, nobody can enter
  - Any other reader or writer will wait
- But back-and-forth switching is buggy:
  - Any number of readers can enter in a row
- Readers can “starve” writers
- With semaphores, building a solution that has the desired back-and-forth behavior is tricky!
Today

- Which practical problems can we solve with semaphores?
- Bounded-Buffer Problem
- Readers-Writers Problem