

More Types of Synchronization

11/29/16

Today's Agenda

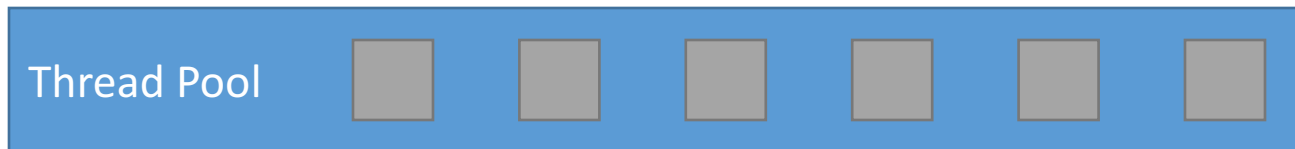
- Classic thread patterns
- Other parallel programming patterns
- More synchronization primitives:
 - RW locks
 - Condition variables
 - Semaphores
- Message passing
- Exercise

Common Thread Patterns

- Thread pool (a.k.a. work queue)
- Producer / Consumer (a.k.a. Bounded buffer)
- Thread per client connection

Thread Pool / Work Queue

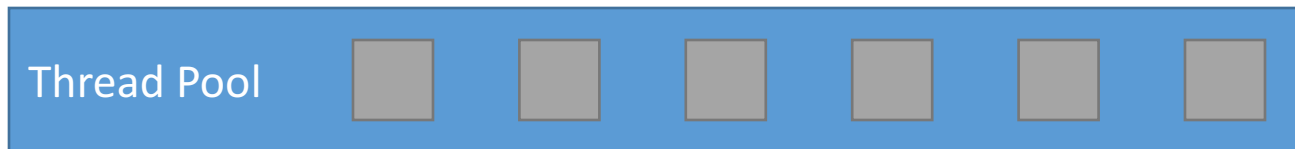
- Common way of structuring threaded apps:



Thread Pool / Work Queue

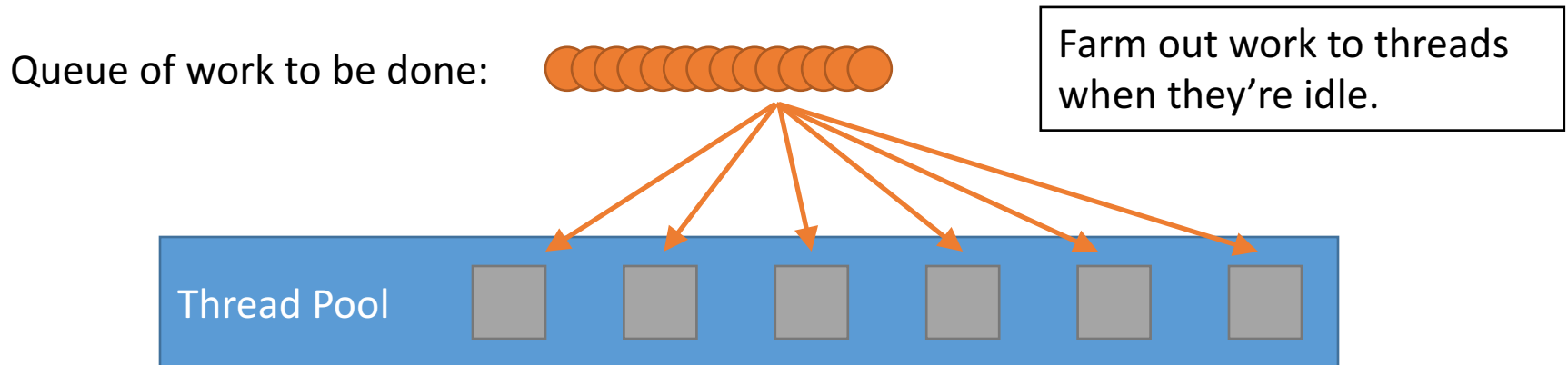
- Common way of structuring threaded apps:

Queue of work to be done: 




Thread Pool / Work Queue

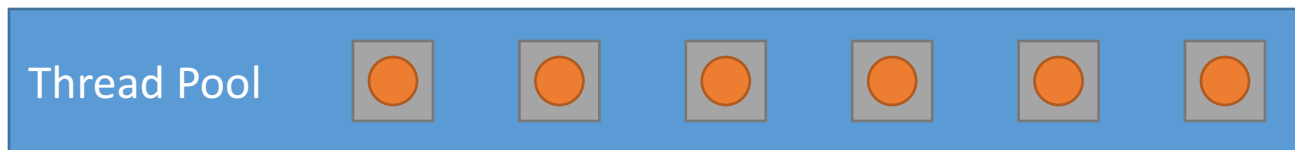
- Common way of structuring threaded apps:



Thread Pool / Work Queue

- Common way of structuring threaded apps:

Queue of work to be done: 

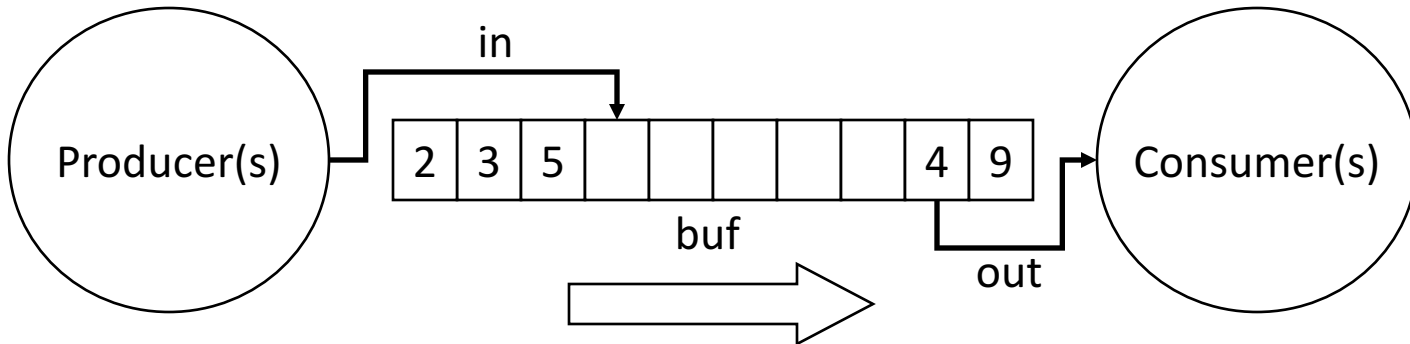


As threads finish work at their own rate, they grab the next item in queue.

Common for “embarrassingly parallel” algorithms.

Works across the network too!

The Producer/Consumer Problem



- Producer produces data, places it in shared buffer
- Consumer consumes data, removes from buffer

All kinds of real-world examples:

print queue: printer is consumer

CPU queue of ready processes/threads to run on CPU

Thread Per Client

- Consider a web server:
 - Client connects
 - Client asks for a page:
 - <http://web.cs.swarthmore.edu/~bryce/cs31/f16>
 - Server looks through file system to find path (I/O)
 - Server sends back html for client browser (I/O)
- Web server does this for MANY clients at once

Thread Per Client

- Server “main” thread:
 - Wait for new connections
 - Upon receiving one, spawn new client thread
 - Continue waiting for new connections, repeat...
- Client threads:
 - Read client request, find files in file system
 - Send files back to client
 - Nice property: Each client is independent
 - Nice property: When a thread does I/O, it gets blocked for a while. OS can schedule another one.

Other Noteworthy Parallel Patterns

- Single instruction, multiple data (SIMD)
 - Apply the same operation independently to many pieces of data.
- Map-Reduce
 - Apply the same operation independently to many pieces of data, then combine the results.

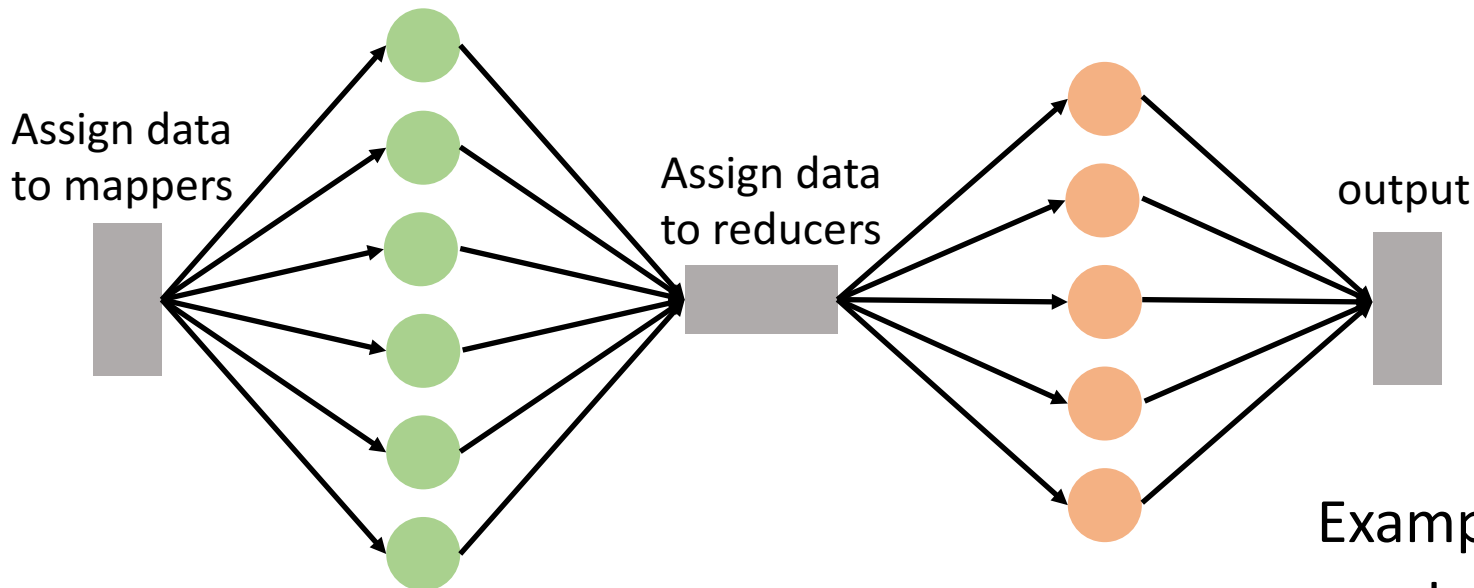
Single instruction, multiple data

- Apply the same operation independently to many pieces of data.
- This is so common in graphics that we have specialized hardware for it (graphics cards).
- Graphics hardware can be used for non-graphics SIMD tasks.
 - Known as GPGPU: general purpose programming on graphics processing units.
 - Example: matrix multiplication for machine learning.



Map-Reduce

- **Map step:** perform some computation on each piece of data.
- **Reduce step:** combine the results of the mappers.



Example: find the most-common words in a book.

Synchronization Mechanisms

- Mutex locks
 - Guarantee *mutually exclusive* access.
- Barriers
 - Wait for other threads to catch up.
- Read/write locks
- Condition variables
- Semaphores

Read/Write locks

- Readers/Writers Problem:
 - An object is shared among several threads.
 - Some threads only read the object, others may write it.
 - We can safely allow multiple readers.
 - But writers need exclusive access.

- `pthread_rwlock_t`:
 - `pthread_rwlock_init`: initialize rwlock
 - `pthread_rwlock_rdlock`: lock for reading
 - `pthread_rwlock_wrlock`: lock for writing

Condition Variables

Wait for a condition to be true.

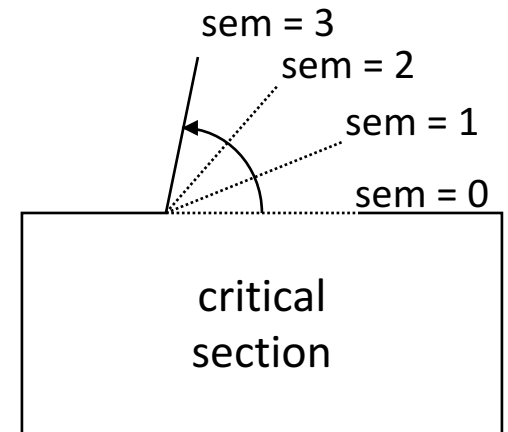
- In the pthreads library:
 - `pthread_cond_init`: Initialize CV
 - `pthread_cond_wait`: Wait on CV
 - `pthread_cond_signal`: Wakeup one waiter
 - `pthread_cond_broadcast`: Wakeup all waiters
- Condition variable is associated with a mutex:
 1. Lock mutex, realize conditions aren't ready yet.
 2. Temporarily give up mutex until CV signaled.
 3. Reacquire mutex and wake up when ready.

Using Condition Variables

```
while (TRUE) {  
    //independent code  
  
    lock(m);  
    while (conditions bad)  
        wait(cond, m);  
  
    //proceed knowing that conditions are now good  
  
    signal (other_cond); // Let other thread know  
    unlock(m);  
}
```

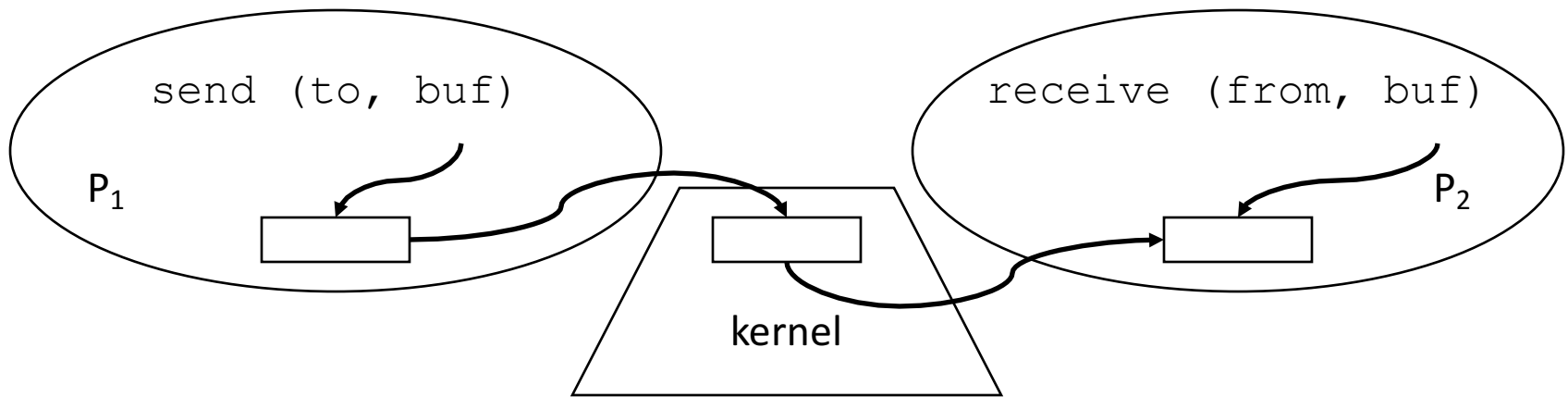
Semaphores: generalized mutexes

- Semaphore: synchronization variable
 - Has integer value
 - List of waiting threads
- Works like a gate
- If $sem > 0$, gate is open
 - Value equals number of threads that can enter
- Else, gate is closed
 - Possibly with waiting threads



A semaphore with initial value 1 is a mutex

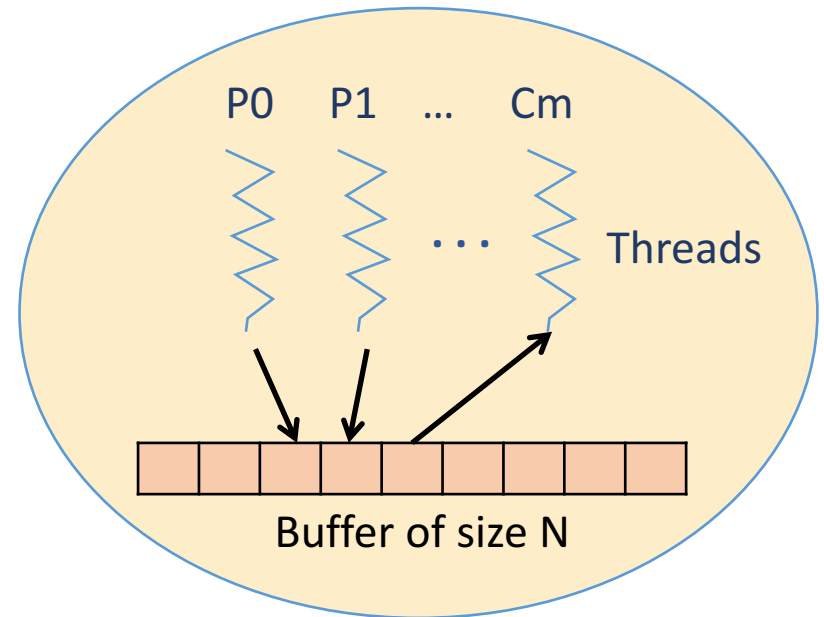
Message Passing



- Operating system mechanism for IPC
 - `send (destination, message_buffer)`
 - `receive (source, message_buffer)`
- Data transfer: in to and out of kernel message buffers
- Synchronization: can't receive until message is sent

Producer-Consumer Problem

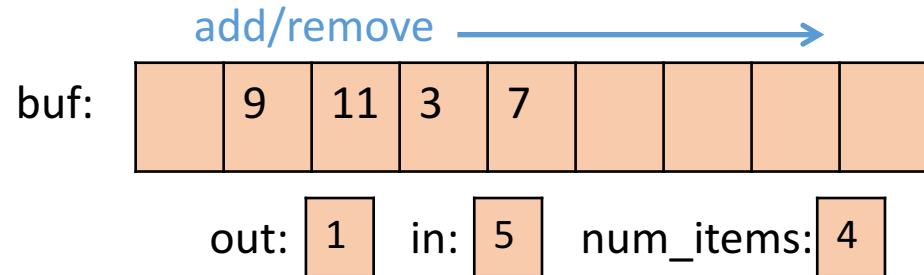
- A shared fix-sized buffer
- Two types of threads:
 1. **Producers:** create an item, add it to buffer.
 2. **Consumers:** remove an item from buffer, and consume it.



Producer/Consumer Synchronization?

Circular Queue Buffer: add to one end (in), remove from other (out)

```
int buf[N];  
int in, out;  
int num_items;
```



Assume Producers & Consumers forever produce & consume

Q: Where is Synchronization Needed in Producer & Consumer?

Producer:

Consumer:

Producer/Consumer Synchronization?

Producer:

- Needs to wait if there is no space to put a new item in the buffer (**Scheduling**)
- Needs to wait to have mutually exclusive access to shared state associated with the buffer (**Atomic**):
 - Size of the buffer (num_items)
 - Next spot to insert into (in)

Consumer:

- Needs to wait if there is nothing in the buffer to consume (**Scheduling**)
- Needs to wait to have mutually exclusive access to shared state associated with the buffer (**Atomic**):
 - Size of the buffer (num_items)
 - Next spot to remove from (out)

Exercise

Come up with a pseudo-code solution to producer and consumer.

- Assume circular buffer add/remove functions provided (don't check overwrite or garbage return value)
- What does Producer need to do to add an item?
- What does Consumer need to do to remove an item?

Questions to Ask:

- Where do you need to add synchronization?
 - What sort of synchronization?
- Do you need any other state information?