Basic Idea – consider the operation of an assembly line. Processes don’t have to operate in perfect lock-step, but a certain order must be maintained. For example, must put wheels on before hubcaps. It’s OK for wheel mounter to get ahead, but hub-capper must wait if it gets ahead. This is an example of the Producer and Consumer problem:
- Producer: creates copies of a resource.
- Consumer: uses up (destroys) copies of a resource.
- Synchronization: keeping producer and consumer in step.
- Define constraints (definition of what is “correct”).
  (Note the importance of doing this before coding.)
  - Consumer must wait for Producer to fill buffers. (resource management)
  - Producer must wait for Consumer to empty buffers. (resource management)
  - Only one process at a time must fiddle with buffer pool. (mutual exclusion)
- A separate condition variable is used for each constraint.
- Unix Producer/Consumer Example: `grep foo file.txt | wc -l`

```c
pthread_cond_t empty = PTHREAD_COND_INITIALIZER;
pthread_cond_t full = PTHREAD_COND_INITIALIZER;
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

void produce()
{
    pthread_mutex_lock(&mutex);
    while (is_empty(pool_of_empty))
        pthread_cond_wait(&empty, &mutex);
    get_empty_buffer_from_pool_of_empty();
    pthread_mutex_unlock(&mutex);
    // produce data into buffer
    pthread_mutex_lock(&mutex);
    add_full_buffer_to_pool_of_fulls();
    pthread_cond_signal(&full);
    pthread_mutex_unlock(&mutex);
}

void consumer()
{
    pthread_mutex_lock(&mutex);
    while (is_empty(pool_of_fulls))
        pthread_cond_wait(&full, &mutex);
    get_full_buffer_from_pool_of_fulls();
    pthread_mutex_unlock(&mutex);
    // consume data in buffer
    pthread_mutex_lock(&mutex);
    add_empty_buffer_to_pool_of_empty();
    pthread_cond_signal(&empty);
    pthread_mutex_unlock(&mutex);
}
```

Key questions:
- Why does `produce` do `wait(EMPTY)` but `signal(FULL)`?

- Is order of last two lines important?

- Could we have separate mutexes for each pool?

- How could this be extended to have two consumers?

- Can “if”s replace the two while loops? \[ \text{No, recall Friday!} \]
int CompareAndSwap(int *ptr, int expected, int new)
{
    int actual = *ptr;
    if (actual == expected)
        *ptr = new;
    return actual;
}

Chapter 32
int CompareAndSwap(int *address, int expected, int new)
{
    if (*address == expected)
        *address = new;
    return 1; // success
}
return 0; // failure

while (CAS (lock -> turn, myTurn, myTurn + 1) == 0);