Monitors

Producer- Consumer problem

- Monitors: A high-level data abstraction tool that automatically generates atomic operations on a given data structure. A monitor has:
  - Shared data.
  - A set of atomic operations on that data.
  - A set of condition variables.
- Monitors can be imbedded in a programming language: Mesa/Cedar from Xerox PARC.
- Typical implementation: each monitor has one lock. Acquire lock when begin a monitor operation, and Release lock when operation finishes. Optimization: reader/writer locks. Statically identify operations that only read data, then allow these read-only operations to go concurrently. Writers get mutual exclusion with respect to other writers and to readers. Standard synchronization mechanism for accessing shared data.
- Advantages: reduces probability of error (never forget to Acquire or Release the lock), biases programmer to think about the system in a certain way (is not ideologically neutral). Trend is away from encapsulated high-level operations such as monitors toward more general purpose but lower level synchronization operations.

- Bounded buffer using monitors and signals
  - Shared State
    - data[10] - a buffer holding produced data.
    - num - tells how many produced data items there are in the buffer.
  - Atomic Operations
    - Produce(v) called when producer produces data item v.
    - Consume(v) called when consumer is ready to consume a data item. Consumed item put into v.
  - Condition Variables
    - bufferAvail - signalled when a buffer becomes available.
    - dataAvail - signalled when data becomes available.

- monitor {

```
Condition *bufferAvail, *dataAvail;
int num = 0;
int data[10];

Produce(v) {
    while (num == 10) { /* Mesa semantics */
        bufferAvail->Wait();
    }
    put v into data array
    num++;
    dataAvail->Signal(); /* must always do this? */     /* can replace with broadcast? */
}

Consume(v) {
    while (num == 0) { /* Mesa Semantics */
        dataAvail->Wait();
    }
    put next data array value into v
    num--;
    bufferAvail->Signal(); /* must always do this? */      /* can replace with broadcast? */
}
```

}