

# Producers Consumers **using a condition variable**

```
cond_init(not_full)
cond_init(not_empty)
```

```
void producer () {
    while(1) {
        item := produce()
        acquire(mutex)
        while(!empty(buffer))
            cond_wait(not_full, mutex)
        write(buffer, item)
        cond_signal(not_empty)
        release(mutex)
    }
}
```

```
void consumer () {
    while(1) {
        acquire(mutex)
        while(empty(buffer))
            cond_wait(not_empty, mutex)
        item := read(buffer)
        cond_signal(not_full)
        release(mutex)
        consume(item)
    }
}
```

# Another Synchronization Construct

## **Semaphore**

An abstract data type to provide mutual exclusion  
described by *Dijkstra* in the "*THE multiprogramming system*" in 1968

➔ Semaphores are “integers” that support two operations:

- Semaphore::P() decrement, block until semaphore is open  
a.k.a wait(), or sem\_wait(), or sema\_down()
- Semaphore::V() increment, allow another thread to enter  
a.k.a signal(), or sem\_post(), or sema\_up()

✓ Semaphore safety property  
the semaphore value is always greater than or equal to 0