## IPC Primitives

### Semaphore

*Data structures:* Non-negative integer variable $s$; queue of any processes blocked on that semaphore integer.

*Primitive operations:*

*down* ($s$) `if` $s = 0$ `then block`
`else decrement` $s$

*up* ($s$) `if anyone is blocked then unblock one of them`
`else increment` $s$

### Monitor

A *monitor* is a special programming-language structure (comparable to a class).

*Data structures: condition variable*, which has an implicit queue of any processes blocked on that condition variable.

*Primitive operations:*

*wait* ($c$) `get added to the queue` $c$
`block`

*signal* ($c$) `if` $c$ `is not empty then`
`remove one process from queue`
`unblock that process`

*Convention:* For technical reasons, we will always call *signal*() **last** in monitor routines.

### Message passing

*Data structures:* messages (sequences of bytes); implicit or explicit queues of messages between two processes, one of which may block if the queue is empty.

*Primitive operations:*

*send* (*dst*, $m$) `send message` $m$ `to` *dst*

*receive* (*dst*, $m$) `receive a message from` *dst* `and store it in` $m$

---

### Barrier

For parallel computations with any number of processes: when each process reaches the barrier, it is blocked until all processes have reached the barrier.

*Barrier* () `block until all processes have called` *Barrier()*