Semaphores

- Semaphores are used to synchronize the actions of processes
  - semaphore: an integer with two primitive functions P(sem) and V(sem)
- Both P and V are *atomic*
- P(S) blocks the calling process if S <= 0, once unblocked it decrements S.
- V(S) increments S. If S > 0 then one process blocked on P(S) becomes unblocked.
- e.g if S = 0, the call P(S) will block the process until S becomes positive (via another process calling V(S))

Semaphores

```
P(S):
        While (S <= 0);
        S--;

V(S):   S++;
```

Semaphores

- The value of a semaphore is the "memory" of the difference between the number of V's and P's called
- e.g. 1 - critical sections:
  - (Initially, S = 1)

```
While(1)
{
P(S);
Critical section
V(S);
Non critical section

}
```

Semaphores

- e.g. 2 - process synchronization (want S1 to complete before S2 begins)
  - (Initially, S = 0)

```
P1: S1 (bunch of code)
      V(S);

P2:
     P(S);
     S2
```

Producer / Consumer

```
SEM full = 0;           (# of items available for consumption)
SEM empty = N;          (# of empty buffer entries)
SEM mutex = 1;
Producer:   while (1) {
                -produce new item
                P(empty); (make sure we have at least 1 empty buffer)
                P(mutex); (to guard against concurrent buffer manipulation)
                -add the new item to the buffer
                V(mutex); (signal OK for buffer access by consumer)
                V(full); (records addition of item to buffer)
            }
Consumer: while(1) {
                P(full); (wait until at least 1 buffer is filled)
                P(mutex); (wait for concurrency OK from producer)
                -remove item from the buffer
                V(mutex); (signal concurrency OK to producer)
                V(empty); (record the addition of 1 empty buffer)
                -consume item
            }
```

Producer / Consumer

- What would happen if, on the producer, we reversed the lines "`produce new item`" and "`P(empty)`"?

  This would prevent the producer from producing an item just because no buffer is available (resulting in less concurrency)

- What would happen (again in the producer) if we reversed the lines "`P(empty)`" and "`P(mutex)`"?

  The producer may be allowed into its critical section, but could be blocked on P(empty) if there are no buffers available. (this results in deadlock because we have no buffers available and the mutex is unlocked)