A solution to the critical section problem must satisfy 3 criteria:

Mutual Exclusion If a process is executing its critical section, no other process can be executing its critical section

Progress If no other process is executing its critical section, and some process wants to enter its critical section, then only those processes not executing code in their remainder section can decide who enters

Bounded Waiting There must be a limit on the number of times that another process is allowed to enter its critical section after a process has made a request to enter its critical section

```
do {
    flag[i] = true;
    turn = j;
    while (flag[j] && turn == j) ;
    // critical section
    flag[i] = false;
    // remainder section
} while (true);
```

- Works for 2 processes `j = 1 - i;`
- Requires additional (shared) data

```
int turn;
boolean flag[2];
```

- What is the utility of `turn, flag`?

`j` refers to the 'other' process `turn` indicates whose turn it is to enter their section `flag[]` specifies if one of `i` or `j` are ready to enter their section

```
do {
    // acquire lock
    // critical section
    // release lock
    // other stuff (remainder)
} while(true);
```

Semaphores

A semaphore is a data structure that permits threads to coordinate among each other and specify which thread(s) wait and which thread(s) execute.

- A data struct that contains only a single non-negative integer as a datum

- The int can be initialized to any non-negative value, but once declared and set, it is modified only by several methods (there are no direct get/set methods)
- Operation 1: increment
- Operation 2: decrement
- Has a constructor that initializes the semaphore

If there *were* set/get functions, it would run into the same issue as the critical section problem.

If decrement would result in the datum being negative, then the thread that issued the decrement will be blocked and will not continue until another thread increments the semaphore.

If increment occurs, an already waiting thread is unblocked.

Which thread is executed after a shared semaphore's value is incremented?

Nuances

- Incrementing may affect other threads
- Decrementing will only affect the thread that called it

signal/wait