~~25~~

Semaphore init, semwait, semsignal

① may be initialized to a nonnegative int val **count**
(corresponding to "how many at once"

② semWait - decrements count,
if count becomes negative then process ~~executing~~ is blocked
otherwise it proceeds [blocking is not busy wait, give up TS]

③ semSignal - increments count
if count is <= 0 then a blocked process is unblocked

BTW no out of the box semaphore in C++ 11

Bank ex.

```
semaphore s(1);
void withdraw (int amount) {
    semwait(s);
    if (balance > amount) {
        cout << "approved";
        balance -= amount;
    }
    semsignal(s);
```

good? what if up → semsig → gi
& down semwait → gi--

```
Thread T1(withdraw, 10)
Thread T2(withdraw, 10)
Thread T3(withdraw, 10)
```

can signal & wait on diff threads

```
semaphore s = 0
s.count = 0;
int gi = 0;

up() {
    for(int i = 0; i < 10; i++)
        gi++;
    }   semsignal(s);

down() {
    for(int i = 0; i < 10; i++)
    semwait(s);
    gi--;
}
```

Semaphore

```
struct semaphore {
        volatile int count;        // # processes allowed in at a time
        queueType queue;           // remove FIFO no starvation strong semaphore
                                   // diff order, starvation possible
};

void semwait (semaphore s)
{
        s.count--;
        if (s.count < 0) {
            // place this process in s.queue
            // block it          std::thread::yield
        }
}

void semSignal ( semaphore s) {
        s.count++;
        if (s.count <= 0) {
            // remove a process P from s.queue
            // place process p on ready list
        }
}
```

C++11 has no semaphore, lets build one with mutex + condition var.

Semaphore
sem_wait(); sem_signal(); sem_init();

```
struct semaphore {
volatile int count;
    mutex m;
    condition_variable cv; }

sem_wait(semaphore& s){
    unique_lock<mutex> m(s.m);
    s.count--;
    if (s.count < 0)
        s.cv.wait(m);
}}
sem_signal(semaphore& s){
    unique_lock<mutex> m(s.m);
    s.count++;
    if (s.count <= 0)
        s.cv.notify_one();
}
```

spurious wakeup handler