Semaphore init, semwait, semsignal

① may be initialized to a nonnegative natural count (corresponding to "how many at once")

② semWait - decrements count, if count becomes negative then process 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);
}

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

can signal & wait on diff threads

```
 1/*
 2 * Semaphore.h
 3 *
 4 *  Created on: Nov 8, 2017
 5 *      Author: keith
 6 */
 7
 8#ifndef SEMAPHORE_H_
 9#define SEMAPHORE_H_
10#include <mutex>
11#include <condition_variable>
12
13class Semaphore {
14public:
15    Semaphore(int cnt=1);
16    virtual ~Semaphore();
17
18    void wait();
19    void signal();
20
21private:
22    volatile int count;
23    std::mutex m;
24    std::condition_variable cv;
25};
26
27#endif /* SEMAPHORE_H_ */
28
```

Semaphore.cpp

```
1 /*
2  * Semaphore.cpp
3  *
4  *  Created on: Nov 8, 2017
5  *      Author: keith
6  */
7 #include <iostream>
8 #include "Semaphore.h"
9 using namespace std;
10
11 Semaphore::Semaphore(int cnt) :
12         count(cnt) {
13 }
14 Semaphore::~Semaphore() {
15 }
16
17 void Semaphore::wait() {
18     unique_lock<mutex> mlk(m);
19
20     while(count == 0)
21         cv.wait(mlk);
22     --count;
23 }
24 void Semaphore::signal() {
25     unique_lock<mutex> mlk(m);
26
27     ++count;
28     cv.notify_one();
29 }
30
31
```

demo

```
Semaphore s(2); // allow 3 @ a time

func(int i){
    s.wait();
    cout << "leaving" << i << endl;
}

thread t1(func,1);
thread t2(func,2);
thread t3(func,3);
thread t4(func,9);
    :
```