CS341 #15
Condition Variables. Implement a Mutex Lock.
The Critical Section Problem

cv : conditional variable

1. How do I block a thread (= send it to 'sleep')?

p_cond_wait (&cv

thread will stop (not return immediately)

2. How do I wake up threads that are blocked on a condition variable?

wakeup at most one ← p_cond_signal (&cv)

wake up all ← p_cond_broadcast (&cv)

Example: Fix the following methods using a condition variable and mutex lock to ensure the cake integer is never negative.

```
01  pthread_mutex_t m = PTHREAD_MUTEX_INITIALIZER;
02  pthread_cond_t cv = PTHREAD_COND_INITIALIZER;
03
04  int  cake = 0; // should not be negative
05
06  void  decrement() { // Will block if zero
07    p_m_lock(&m)
08      while(cake == 0) {
09        pthread_wait(&cv,&m)
10        sleep(1)
11
12      }
13      cake --;
14    p_m_lock(&m)
15  }
16
17  void increment() {
18    p_m_lock
      cake ++;
19    p_m_unlock
  }
```

if two threads look at this line at the same time

change to

let go the lock

① unlock m
② block
⋮
③ lock m

cake can be -1

p_cond_signal (&cv

3. How does `pthread_cond_wait` *really* work?

unlock m
blocking
} as single operation, otherwise a "wake" may happen in between

actually, spurious wake

Solution: use while !

4. Challenge. A fixed size stack that blocks

```
01 pthread_mutex_t m = PTHREAD_MUTEX_INITIALIZER;
02 pthread_cond_t cv = PTHREAD_COND_INITIALIZER;
03 double array[10];
04 int n = 0;
05
06 // blocks while full (n ==10)
07 void push(double v) {
08   p_m_lock(&m)
09   while (n ==  ) {
10
11     p_cond_wait (&cv, &m)
12   }
13   p_cond_broad (&cv)
14   array[n++] = v
15 }
16 // blocks while empty (n == 0)
17 double pop() {
18
19
20
21
22
23
24       double r = array[--n]
25       return r
26 }
27 // Test with 2+ threads that add values...
28 void* generator(void*){
29    for(int i =0; i < 100000; i++)
30       push(i);
31   Return NULL;
32 }
33 // And one thread that remove values
34 void * consumer(void*result) {
35   double sum  = 0, i=0;
36   while( (i=pop()  != -1) sum += i;
37   printf("%.0f", sum);
38   Return NULL;
39 }
40
```

How can you *implement* a reliable mutex lock?

5. Let's try writing a simple implementation...

```
01   pthread_mutex_init(int * m)  {  *m= 0; }
02
03   pthread_mutex_lock(int* m) {
04      while(*m ==1) {
05         pthread_yield(); /*sleeps for a short time */
06      }
07      *m = 1;
08   }
09   pthread_mutex_unlock(int* m)   { ? _____}
```

Problems?

---

6. CPU support: Use an atomic CPU instruction.

Imagine a special *Atomic_Exchange* instruction '`exch`' that swaps the values at two addresses as an *indivisible*, *uninterruptable* operation

```
01   pthread_mutex_init(int* m)  { *m= 0; }
02
03   pthread_mutex_lock(int* m) {
       for(int q = 1; q ; ) { _____ }
     }
04
05   pthread_mutex_unlock(int* m)  {_____ }
```

---

7. The Critical Section Problem

while(running) {

1. Wait to enter the critical section if another thread is in the CS.
2. Critical Section Code here. Only one thread in here at a time!
3. Leave critical section. Allow another waiting thread to enter.
4. // do other stuff most of the time

}

~~ Welcome to the **Critical Section Problem** game show! ~~

Today's prizes: mutual exclusion and progress

Candidate #1. Use a single, boolean "flag"

`boolean flag`

Thread A	*Thread B*
wait while the flag is up raise the flag! *Critical Section* code here lower the flag! ...	wait while the flag is up raise the flag! *Critical Section* code here lower the flag! ...

// Then each thread does other work but will repeat this again sometime in the future

Problems?

Candidate #2. Give each thread its own a flag.

`boolean flagA, flagB`

wait while B's flag is up raise A flag *Critical Section* code here lower A flag	wait while A's flag is up raise B flag *Critical Section* code here lower B flag

Problems?

Candidate #3. Change the sequence order

raise A flag wait until B flag is down *Critical Section* code here lower A flag	raise B flag wait until A flag is down *Critical Section* code here lower B flag

Problems?