CS241 #15
Condition Variables. Implement a Mutex Lock.
The Critical Section Problem

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

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

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;
05
06    void  decrement() { // Will block if zero
07
08      while(cake == 0) {
09
10          sleep(1)
11
12      }
13      cake --;
14
15    }
16
17    void increment() {
18        cake ++;
19    }
```

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

4. Challenge. A fixed size stack:

```
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
09
10
11
12
13
14
15
16 }
17 // blocks while empty (n == 0)
18 double pop() {
19
20
21
22
23
24
25
26 }
27 // Test with 2+ threads that add values...
28 void* generator(void*){
29    for(int i =0; i < 10000; i++)
30        push( i);
31    return;
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 }
```

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.

Suppose a special '*Atomic_Exchange*' instruction 'exch' exists that swaps the values at two addresses as an *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
}
```

Attempt #1. Use a single, boolean "flag"
```
boolean flag
```

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

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

Attempt #2. Give each thread its own a flag.
```
boolean flagA, flagB
```

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

Problems?

Attempt #3. Change the sequence order

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

Problems?

Attempt #4. Try a single turn-based shared variable.
```
turn=1
```

while(turn == 2) { }	while(turn == 1) { }
Critical Section code here	*Critical Section* code here
turn = *2*	turn = 1

Problems?