CS241 Lecture 14 Lawrence Angrave
Condition Variables

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

1. How do I wake up threads that are blocked on a condition var?

2. The cake is a lie... Complete the following methods using a condition variable and mutex locks. The cake integer must never be negative.

```
pthread_mutex_t m = PTHREAD=MUTEX_INITIALIZER;
pthread_cond_t cv = PTHREAD_COND_INITIALIZER;

int  cake = 0;

void  decrement() { // Waits if nonzero

  while(cake == 0) {
    // sleep


  }
  cake --;

}

void increment() {
    cake ++;
}

```

3. How does pthread_cond_wait *really* work?

4. Challenge. A fixed size stack:

```
pthread_mutex_t m = PTHREAD=MUTEX_INITIALIZER;
pthread_cond_t cv = PTHREAD_COND_INITIALIZER;
double array[10];
int n = 0;

// blocks while full (n ==10)
void push(double v) {








}
// blocks while empty (n == 0)
double pop() {







}

void* generator(void*){
   for(int i =0; i < 10000; i++)
      push( i);
  return;
}
void * consumer(void*result) {
  double sum  = 0, i=0;
  while( (i=pop()  != -1) sum += i;
  printf("%.0f", sum);
}

```

Some more C functions for you:
`sigprocmask pthread_sigmask pthread_self() atexit sigaction`
The big problem: How to implement the mutex lock
Hardware CPU instruction simplified solution (‘*Atomic_Exchange*’ swaps values at two addresses as an *uninterruptable* operation)

```
typedef p_mutex_t int;
pthread_mutex_init(p_mutex_t* m)          { *m = 1; }
pthread_mutex_lock(p_mutex_t* m)          { int local=0;
                                            do {
                                    ATOMIC_EXCHANGE(m, &local);
                                            } while(!local);
                                          }

pthread_mutex_unlock(p_mutex_t* m)        { *m = 1; }
```

C-Code Candidate # 0 (Review) Protect our critical section with a mutex. But how should it work!?

```
pthread_mutex_lock(p_mutex_t* m)          { while(m->lock) {}; m->lock = 1;}
pthread_mutex_unlock(p_mutex_t* m)        { m->lock = 0; }
```

Problems?

Psuedo code Candidate # 1

wait until your flag is lowered raise my flag *// Do Critical Section stuff* lower my flag	wait until your flag is lowered raise my flag *// Do Critical Section stuff* lower my flag

// Threads do other stuff and then will repeat at sometime in the future

Candidate #2

raise my flag wait until your flag is lowered *// Do Critical Section stuff* lower my flag	raise my flag wait until your flag is lowered *// Do Critical Section stuff* lower my flag

// Threads do other stuff and then will repeat at sometime in the future

Problems with 2?

Candidate #3

wait until my turn (turn==id?) *// Do Critical Section stuff* turn = *yourid*	wait until my turn (turn==id?) *// Do Critical Section stuff* turn = *yourid*

// Threads do other stuff and then will repeat at sometime in the future