CS241 Lecture 15 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.

|  |
| --- |
| 1. pthread\_mutex\_t m = PTHREAD=MUTEX\_INITIALIZER; 2. pthread\_cond\_t cv = PTHREAD\_COND\_INITIALIZER; 3. int cake = 0; 4. void decrement() { // Waits if nonzero 5. while(cake == 0) { 6. // sleep 7. } 8. cake --; 9. } 10. void increment() { 11. cake ++; 12. } |

3. How does pthread\_cond\_wait *really* work?

4. Challenge. A fixed size stack:

|  |
| --- |
| 1. pthread\_mutex\_t m = PTHREAD=MUTEX\_INITIALIZER; 2. pthread\_cond\_t cv = PTHREAD\_COND\_INITIALIZER; 3. double array[10]; 4. int n = 0; 5. // blocks while full (n ==10) 6. void push(double v) { 7. } 8. // blocks while empty (n == 0) 9. double pop() { 10. } 11. void\* generator(void\*){ 12. for(int i =0; i < 10000; i++) 13. push( i); 14. return; 15. } 16. void \* consumer(void\*result) { 17. double sum = 0, i=0; 18. while( (i=pop() != -1) sum += i; 19. printf("%.0f", sum); 20. } |

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