CS241 #16 CV Cookies, Counting Semaphores, Ring buffer.

**1. Condition Variables Warm-up Challenge: Eat cookies fast!**

*****Meanwhile in a Parallel Universe ...*

*Two threads viciously eat cookies but are blocked on a c.v. ...*

1. int jar = 0;
2. pthread\_mutex\_t m = PTHREAD\_MUTEX\_INITIALIZER;
3. pthread\_cond\_t cv1 = PTHREAD\_COND\_INITIALIZER;
4. void\* cookie\_eater(void\*arg) {
5. char\* name = (char\*) arg;
6. while(game\_running) {
7. while(jar == 0) {
8. printf("%s nap time\n", name);
9. ?
10. }
11. jar --;
12. printf("%s eats! %d remain\n",name, jar);
13. }
14. printf("%s is exiting...",name);
15. return NULL;
16. }

Complete the add\_cookies to add cookies to the cookie jar  
(Pretend cookie jar has ∞ capacity)

1. void add\_cookies(int add) {
2. assert(add > 0);

2. What must be locked before calling p\_cond\_wait ? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

3. You wake a thread blocked inside a condition variable but it does not return from p\_cond\_wait. Why?

Another thread still \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

The blocked thread will continue when \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

4. How do I use counting semaphores?

sem\_init

sem\_wait

sem\_post

5. What is a fixed ring buffer? Why would I use it?

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

6. Producer Consumer Case Study:   
Use counting semaphores to implement a fixed ring buffer

pthread\_mutex\_t m;

// (Not OSX!)

sem\_t \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

void init() {

sem\_init(\_\_\_\_\_\_\_\_\_\_\_ , 0, \_\_\_\_\_\_);

sem\_init( \_\_\_\_\_\_\_\_\_\_\_, 0, \_\_\_\_\_\_);

pthread\_mutex\_init( &m , NULL);

}

void sync\_enqueue(work\_t \*work) {

}

work\_t\* sync\_dequeue(){

7. Quick quiz

i) How many threads can be executing line 8 or 14 at a time? Why?

ii) What have I made? (Missing code? + Better function names?

|  |
| --- |
| 1. pthread\_mutex\_t m = PTHREAD=MUTEX\_INITIALIZER; 2. pthread\_cond\_t cv1 = PTHREAD\_COND\_INITIALIZER; 3. int mystery = 5; 4. void A?() { // Waits if count would become -ve 5. p\_m\_lock(&m) 6. while(mystery == 0) p\_cond\_wait(&cv1, &m); 7. mystery --; 8. p\_m\_unlock(&m); 9. } 10. void B?() { 11. p\_m\_lock(&m); 12. mystery ++; 13. if( \_\_\_\_\_\_\_\_\_\_\_\_\_ ) p\_cond\_signal(&cv1); 14. p\_m\_unlock(&m); 15. } |

CRITICAL SECTION PROBLEM

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?

Candidate #4. Try a single turn-based shared variable.

turn=1

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

Problems?