|  |  |
| --- | --- |
| CS 241 | #18 Barriers. Deadlock. The Reader-Writer Problem |

Challenge 1: "Make a barrier using only one mutex lock() and unlock() call!"

"Impossible! Line 2 is a Critical Section, if a thread has locked the mutex..."

But here is an awful solution. (Why is this a 'poor' solution?)

1. void barrier() {
2. count ++
3. while( count != N) ?

05 }

2. When is disabling interrupts a solution to the Critical Section Problem?

|  |
| --- |
| pthread\_mutex\_lock => { disable interrupts on the CPU }  pthread\_mutex\_unlock => {enable interrupts on the CPU } |

Are there other limitations to this approach?

3. Challenge II: Create a barrier using each of the following lines once. All 5 threads must call barrier before they all continue.

|  |
| --- |
| int remain =5; *earlier...* sem\_init(&s,0,\_\_\_?)  void barrier() { *... Rearrange the following!*  sem\_wait(&s);  sem\_post(&s);  remain --;  pthread\_mutex\_lock(&m);  pthread\_mutex\_unlock(&m);  if(remain)  } |

4. Is there a Race condition?

|  |  |
| --- | --- |
| pleaseStop = 1  p\_cond\_broadcast(&cv) | while(!pleaseStop)  p\_cond\_wait(&cv,&m) |

5. Challenge III. What is the largest value printed by the following?

|  |
| --- |
| int **fireworks**=0;  pthread\_cond\_t cv = P\_COND\_INITIALIZER;  pthread\_mutex\_t m = P\_MUTEX\_INITIALIZER;  pthread\_t tids[5];  int main(argc,argv) {  for(int i=0;i<5;i++) pthread\_create( tids+i , NULL, firework, NULL);  **fireworks** = 1;  p\_cond\_signal(&cv);  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_; // wait for all threads to finish  return 0;  }  void\* firework(void\*param) {  p\_mutex\_lock(&m);  while(**fireworks** ==0) {p\_cond\_wait(&cv, &m); }  p\_cond\_broadcast(&cv);  fireworks ++;  printf("Oooh ahh %d\n", fireworks);  fireworks --;  p\_mutex\_unlock(&m);  return NULL;  } |

**6. Deadlock: " \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_"**

Use two mutex locks and two threads to create an example of deadlock

|  |  |
| --- | --- |
| Thread1: | Thread 2: |

Use three counting semaphores and three threads to deadlock 3 threads

|  |  |  |
| --- | --- | --- |
| thread #1: | thread #2: | thread #3: |

Must deadlock involve threads? What about single-threaded processes? **7.** **The Reader Writer problem**

A common problem in many different system applications

|  |  |
| --- | --- |
| read\_database(table, query) {...} | update\_row(table, id, value) {...} |

|  |  |
| --- | --- |
| cache\_lookup(id) {...} | cache\_modify(id, value) {...} |

**8.** ReaderWriter locks are useful primitives & included in the pthread library!

|  |  |
| --- | --- |
| 1. pthread\_rwlock\_t lock; 2. p\_rwlock\_init 3. p\_rwlock\_wrlock 4. p\_rwlock\_rdlock 5. p\_rwlock\_unlock | 1. cache\_lookup(id) { 2. p...rdlock(...) 3. read from resource 4. p...unlock(...) 5. return result 6. } |

CS241: synch. skills and the ability to *build* these! Along the way, also learn to reason about, develop and fix multi-threaded code

9. ~~ Welcome to the *Reader Writer* Game Show! ~~

*Contestant #1*

|  |  |
| --- | --- |
| p\_mutex\_t \*readlock,\*writelock  readlock=malloc(sizeof p\_mutex\_t)  writelock=malloc(sizeof p\_mutex\_t)  p\_m\_init(readlock,NULL)  P\_m\_init(writelock,NULL)  read() {  lock(readlock)  // do read  unlock(readlock)  } | write() {  lock(writelock)  lock(readlock)  // do writing  unlock(readlock)  unlock(writelock)  } |

Is #1 a Solution? Problems?*Contestant #2*

bool reading=0, writing=0

|  |  |
| --- | --- |
| read() {  while(writing) {}  reading = true  // do reading here  reading = false  } | write() {  while(reading||writing) {}  writing = true  // do writing here  writing = false  } |

Is #2 a Solution? Problems?

*Contestant #3*

|  |  |
| --- | --- |
| read(){  lock(&m)  while (writing)  cond\_wait(cv,m)    reading++  /\* Read here! \*/  reading--  cond\_signal(cv)  unlock(&m) | write(){  lock(&m)  while (reading||writing)  cond\_wait(cv,m)  writing++  /\* Write here! \*/  writing--;  cond\_signal(cv)  unlock(&m) |

Is #3 a Solution? Problems?