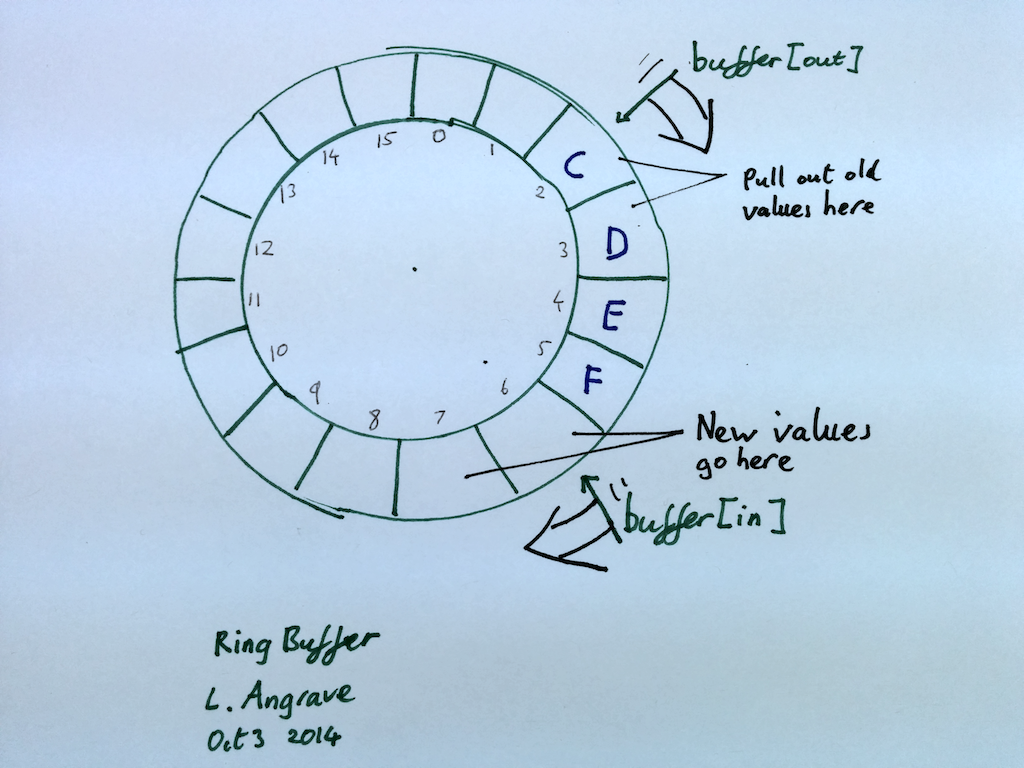
CS241 L15 – RingBuffer. Producer-Consumer. Reader-Writer Problem II

1 What is a ring buffer? How does it work?

Implement add 'enqueue; and remove 'dequeue' methods of Ring Buffer of fixed size 16 for a single-threaded program (assume it never under-/over- flows – we will handle that in the next part, i.e. assume the caller will not remove item from an empty queue, or add an item to a full queue)

|  |  |  |
| --- | --- | --- |
| Globals/init: | enqueue (void\*value){ | void\* dequeue() { |

2 What's wrong with this multi-threaded version. When will it fail?

|  |  |  |
| --- | --- | --- |
| Globals/init:  p\_m\_t lock  sem\_t s1,s2  sem\_init(&s1,0,16)  sem\_init(&s2,0,0)  // + above code from #1 | enqueue(void\*value){  p\_m\_lock(&lock)  sem\_wait( &s1 )  // enqueue code above  sem\_post(&s1)  p\_m\_unlock(&lock) | void\* dequeue(){  p\_m\_lock(&lock)  sem\_wait(&s2)  void \* result = //above  sem\_post(&s2)  p\_m\_unlock(&lock)  return result  } |

3 What's wrong with this multi -threaded version. When will it fail?

|  |  |  |
| --- | --- | --- |
| Globals/init:  p\_m\_t lock  sem\_t s1,s2  sem\_init(&s1,0,16)  sem\_init(&s2,0,0)  // + above code from #1 | enqueue(void\*value){  sem\_wait( &s2 )  p\_m\_lock(&lock)  // enqueue code above  sem\_post(&s1)  p\_m\_unlock(&lock) | void\* dequeue(){  sem\_wait(&s1)  p\_m\_lock(&lock)  void \* result = //above  sem\_post(&s2)  p\_m\_unlock(&lock)  return resul;  } |

4 Write the correct version

|  |  |  |
| --- | --- | --- |
| Globals/init:  p\_m\_t lock  sem\_t s1,s2  sem\_init(&s1,0,\_\_\_\_)  sem\_init(&s2,0,\_\_\_\_)  // + above code from #1 | enqueue(void\*value){ | void\* dequeue(){ |

Q5 Review: What is the Reader-Writer Problem?

Q6 What is wrong with the following `solution’ to the R.W. Problem?

|  |  |  |
| --- | --- | --- |
| Version #4 Problems: | read(){  lock(&m)  while (writing)  cond\_wait(&turn, &m)  reading++  /\* Read here! \*/  reading--  cond\_signal(&turn)  unlock(&m) | write(){  lock(&m)  while (reading || writing)  cond\_wait(&turn, &m)  writing++  /\* Write here! \*/  writing--;  cond\_signal(&turn)  unlock(&m) |

**Version #5**

**int** writers; // # writer threads that want to write (some|all may be blocked)  
**int** writing; // # threads that are actually writing (can only be zero or one)  
**int** reading; // Number of threads that are actually reading   
// if writing !=0 then reading must be zero (and vice versa)  
  
**reader**() {  
 **mutex\_lock**(&m)  
 **while** (writers)  
 **cond\_wait**(&turn, &m)  
 // No need to wait while(writing here)   
 // because we can only exit the above loop  
 // when writing is zero  
 reading++  
 **unlock**(&m)  
  
 // < perform reading here >  
  
 **lock**(&m)  
 reading--  
 **cond\_broadcast**(&turn)  
 **unlock**(&m)  
}  
  
**writer**(){  
 **lock**(&m)   
 writers++   
 **while** (reading || writing)   
 **cond\_wait**(&turn, &m)   
 writing++   
 **unlock**(&m)

// < perform writing here >

**lock**(&m)   
 writing--   
 writers--   
 **cond\_broadcast**(&turn)   
 **unlock**(&m)   
}