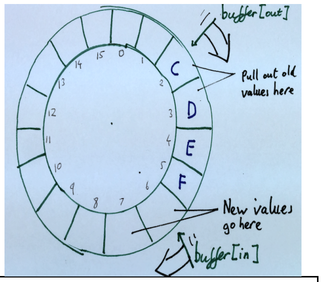
CS241 L15 – RingBuffer. Producer-Consumer. Reader-Writer Problem II

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:	enqueue(void*value){	void* dequeue(){

p_m_t lock
sem_t s1,s2
sem_init(&s1,0,16)
sem_init(&s2,0,0)
// + above code from #1
	p_m_lock(&lock)
	sem_wait(&s1)
	// enqueue code above
	sem_post(&s1)
	p_m_unlock(&lock)
		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:	enqueue(void*value){	void* dequeue(){

p_m_t lock
sem_t s1,s2
sem_init(&s1,0,16)
sem_init(&s2,0,0)
// + above code from #1
	sem_wait(&s2)
	p_m_lock(&lock)
	
	// enqueue code above
	sem_post(&s1)
	p_m_unlock(&lock)
		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:	enqueue(void*value){	void* dequeue(){
p_m_t lock sem_t s1,s2 sem_init(&s1,0,____) sem_init(&s2,0,____) // + above code from #1		

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)
}
```