Call	Description
int pthread_mutex_lock(pthread_mutex_t *mutex)	Locks the mutex. Blocks and does not return if mutex is already locked. If mutex locks successfully, this function returns with a 0. This function returns a non-0 value if something goes wrong.
int pthread_mutex_unlock(pthread_mutex_t *mutex)	Unlocks the mutex. Returns 0 if successful.
int pthread_mutex_destroy(pthread_mutex_t *mutex)	Destroys the mutex. Returns 0 if successful.

```
typedef struct
{
    sem_t empty, full;
    pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
    char data[QLEN][ENTRY_SIZE];
    int len[QLEN];
    int front, back;
    int count;
} TBuffer;

Code for enq (deq is similar). We do a wait on empty in case there
are no more empty slots. If there are empty slots this wait decrements
the # of empty slots. Similarly we up on full to indicate one more
full slot.

void enq(TBuffer *buffer, const char *data, int len)
{
    sem_wait(&buffer->empty);
    pthread_mutex_lock(&buffer->mutex);
    if(buffer->count >= QLEN)
    {
        pthread_mutex_unlock(&buffer->mutex);
        return;
    }

    unsigned int bytesToCopy = (len < ENTRY_SIZE ? len :
ENTRY_SIZE);

    memcpy(buffer->data[buffer->back], data, bytesToCopy);
    buffer->len[buffer->back] = bytesToCopy;
    buffer->count++;
    buffer->back = (buffer->back + 1) % QLEN;
    sem_post(&buffer->full);
    pthread_mutex_unlock(&buffer->mutex);
}
```


Function	Parameters	Description
initBarrier	TBarrier *barrier Int numProcessess	Initializes a new barrier. Barrier = Barrier to initialize numProcesses = # of processes this barrier is expecting. When this many processes have called reachBarrier, everyone is unblocked. Otherwise a process calling reachBarrier will remain blocked.
reachBarrier	TBarrier *barrier int procNum)	Lets a process/thread tell the barrier it has arrived. barrier = Barrier you are using. procNum = Your own process number. See testbarrier.cpp how to derive this. If fewer than numProcesses processes have called reachBarrier, anyone calling reachBarrier is blocked. Once numProcesses have called reachBarrier, all processes are simultaneously unblocked.
resetBarrier	TBarrier *barrier	Resets number of processes at the barrier to 0.

```
void reachBarrier(TBarrier *barrier, int procNum)
{
    barrier->numReached++;

    if(barrier->numReached == barrier->numProcesses)
    {
        // S2 below
        sem_post(&barrier->semArray[barrier->numProcesses-1]);
    }
    else
        sem_wait(&barrier->semArray[procNum]); // S1

    if(procNum > 0)
        sem_post(&barrier->semArray[procNum-1]); // S3
}
```

Each time a process calls reachBarrier, numReached is incremented. As long as numReached < numProcesses (the expected number of processes), the calling process is blocked with a sem_wait (Statement S1). With the final process numReached == numProcesses, and the calling process will call sem_post to unlock process N-1(Statement S2_ .

Process N-1 will then call sem_post to unblock process N-2 (Statement S3) , N-2 will unblock N-3 etc until process 0 is unblocked. All processes will then be unblocked.