Assignment 8

(1) **(1 pt)** Use pthread to create 5 threads and print "Starting thread i" for each thread.

Answer: We initialize an array of `pthread_t`s with a for loop, and print the required message:

```
pthread_t pt[5];
for(int i=0; i<5; i++) {
    printf("Starting thread %d\n", i);
    pthread_create(&pt[i], NULL, thr_fn, (void*)1);
}
```

(2) **(2 pts)** Implement your barrier by using synchronization primitives to block the threads and await the completion of the creation process for all threads.

Answer: We make an improvised barrier wait function with our own barrier structure here.

The structure contains a mutex, a thread condition, and integer variables to keep track of which thread should execute.

```
typedef struct barrier_type {
    pthread_mutex_t mutex;
    pthread_cond_t cond;
    int s_count;
    int count;
    int cycle;
} barrier_t;
```

The implementation for barrier wait first starts with locking the mutex, and keeping track of the cycle the current thread is on.

```
pthread_mutex_lock(&barrier->mutex);
int cycle = barrier->cycle; // Determines whether we are waiting
for all threads to initialize
```

Then, we decrement the counter to indicate the process has started. The first important condition is that we wait till all the other threads are ready. So we release the lock to make sure all the threads have begun.

```
while(cycle == barrier->cycle) { // release mutex as long as
counter isn't 0
    int t = pthread_cond_wait(&barrier->cond, &barrier->mutex);
```

```
    if (t != 0) break;
}
```

`pthread_cond_wait` is an atomic wait, so it will make sure that all other operations happen between waits. This lets use monitor the cycle variable safely. On the last thread's startup, we need to release the barrier and let all the threads run. `pthread_cond_broadcast` is used for safely notifying all threads to stop waiting and continue execution.

```
if(barrier->count == 0) { // release all threads if all have
started.
    barrier->cycle = !barrier->cycle;
    barrier->count = barrier->s_count;
    pthread_cond_broadcast(&barrier->cond);
}
```

Once all threads are unlocked, each one of them will reacquire the mutex they released from `pthread_cond_wait`. So we will unlock the mutex to allow any remaining threads to run.

```
pthread_mutex_unlock(&barrier->mutex);
```

(3) **(1 pt)** Print *"Thread # running"* after calling the barrier.

Answer: Inside the thread function we add a print statement with the return value from `pthread_self()`. It is cast to an unsigned long as per Example 11.2 from the textbook.

```
void* thr_fn(void* a) {
    barrier_wait(&b);
    printf("Thread %lu running\n", (unsigned
long)pthread_self());
    return((void*)1);
}
```

In order to make sure all threads perform printing correctly, we then make sure we block `main()` until each thread is finished executing.

```
for(int i=0;i<5; i++) {
    pthread_join(pt[i],NULL);
}
```