# CS 3400: Sharks and divers

## Introduction

Two small stretches of coral reef are right next to each other, and both have a lot of tasty fish. Sharks love feeding in these reefs, and divers also love fishing in them. The two groups have agreed to a deal where divers can fish in the reefs so long as the sharks never see the divers doing it. If a shark ever sees a diver in one of the reefs, it must eat the diver or be shamed by all of its shark friends.

There are two reefs, six sharks, and two divers.

In this assignment, you will write a program with nine threads: one for each shark, one for each diver, and the main thread that sets everything up.

Your job is to synchronize the sharks and the divers. No diver should ever get eaten. You can assume that if a shark is feeding in either reef, a diver attempting to fish in the other reef will be spotted and eaten. When sharks are not feeding, they will not see divers fishing. You may not starve either the sharks or the divers. Only one shark or diver may fish in a given reef at any one time.

## Implementation

You may use either semaphores, or locks and condition variables to synchronize your program. Both are defined in the POSIX threads library.

To use a semaphore:

```
// create a semaphore
sem_t sem;
int s = sem_init(&sem, 0, initial_count);
assert(s == 0);

// P/down/wait/acquire a semaphore
s = sem_wait(&sem);
assert(s == 0);

// V/up/post/release a semaphore
s = sem_post(&sem);
assert(s == 0);
```

To use locks and condition variables:

```
// create a lock
pthread_mutex_t mutex;
int s = pthread_mutex_init(&mutex, NULL);
assert(s == 0);

// acquire a lock
s = pthread_mutex_lock(&mutex);
assert(s == 0);

// release a lock
s = pthread_mutex_unlock(&mutex);
assert(s == 0);

// create a condition variable
pthread_cond_t condition;
s = pthread_cond_init(&condition, NULL);
assert(s == 0);

// wait on a condition variable
s = pthread_cond_wait(&condition, &mutex);
assert(s == 0);

// notify on a condition variable
s = pthread_cond_signal(&condition);
assert(s == 0);
```

```
// notify all on a condition variable
s = pthread_cond_broadcast(&condition);
assert(s == 0);
```

You should not use any other methods or options for either system, including timeouts, non-blocking acquires, etc.

## Starter code

I have supplied code in `sharks.c` to get you started:

- sharks.c

Along with a simple `Makefile` to build it:

- Makefile

It spawns a thread for each shark and each diver. It then lets them run for a while (60 seconds by default), and then shuts everything down.

The code has two global arrays; one with a spot for each shark, and one with a spot for each diver. Whenever a shark or diver starts fishing, it should set its slot to `true` and call `report()`. Likewise, when it stops fishing, it should set its slot to `false` and call `report()` again. Note that `report()` is not synchronized, so it may occasionally print bogus data, but most of the time it should give you a short summary of the current state of the reefs.

Your job is to create whatever synchronization variables you need (semaphores, locks, condition variables, integers, etc.) as global variables (you can initialize them when you create them), and then finish the implementations of `shark()` and `diver()`.