CS360 Lab B -- The Bonding Lab

- **James S. Plank**
- **CS360**

The layout of this lab is like printer simulation lecture. I have written a driver program and a header file. These are in **bonding-driver.c** and **bonding.h**. You are not allowed to modify these files. These drive a simulation of a system with a bunch of hydrogen and oxygen atoms. Each of these atoms is its own thread. What we want to have happen is to have each atom "bond" with two others to create a molecule. The bonding has to be such that two hydrogen atoms bond with one oxygen atom (and thereby make a water molecule). You'll create the file **bonding.c**, which uses mutexes and condition variables to make this work.

---

bonding.h and simulation parameters

When you compile **bonding-driver.cpp** and **bonding.c** (which you write) into an executable, it runs with the following parameters:

```
usage: bonding seed num_molecules max_outstanding verbosity
```

- **seed** is an integer, which seeds **srand()**.
- **num_molecules** is the total number of water molecules that you'll make in the simulation.
- **max_outstanding** is the maximum number of simultaneous atom threads that can be alive at one time. If **num_molecules** is one, then this should be three. Otherwise, it can have any value from 6 to 1000.
- **verbosity** is a string. I'll describe it more below.

Now, what **bonding-driver** does is the following. It creates two threads -- a **creator** thread and a **joiner** thread. The **creator** thread will create hydrogen and oxygen threads with **pthread_create()**. It makes sure that maximum of **max_outstanding** threads are running at a time. It also makes sure that of the outstanding threads, you can create a maximum number of water molecules. In other words, if there are 10 outstanding threads, you can be assured that at least 6 of them are hydrogen and at least 3 are oxygen. When hydrogen and oxygen threads return, they are cleaned up by the **joiner** thread, and that frees up the **creator** thread to create more threads. The **creator** thread waits until there are at most **max_outstanding/2** outstanding threads, and then it goes ahead and creates more threads.

Hydrogen threads are created by the **creator** thread calling **pthread_create()** on the procedure **hydrogen** (which you are to write). Oxygen threads are created by the **creator** thread calling **pthread_create()** on the procedure **oxygen** (which you are also to write). The argument to each of these is of type **struct bonding arg ***. This contains an id for the thread (id's start with 0 for the first thread, and are numbered consecutively), and a **(void *)**, which where you store your data. Let's take a look at **bonding.h** for a little more detail:

```
/* CS360 Lab B - Bonding.
   Header file bonding.h
   James S. Plank
   April, 2017.

   This file contains the common material and prototypes
   for the Bonding lab.
*/

#include <pthread.h>

/* This is the argument to hydrogen and oxygen threads. */

struct bonding_arg {
  int id;
  void *v;
};

/* You are to write these three procedures. */

void *initialize_v(char *verbosity);
void *hydrogen(void *arg);
void *oxygen(void *arg);

/* When two hydrogen threads and one oxygen thread have bonded
   to create a water molecule, each of them should call Bond()
   with the exact same arguments. This will return a string.
   Each thread should return the string. */

char *Bond(int h1, int h2, int o);

/* I've written these for you.  They are convenient: */

pthread_mutex_t *new_mutex();            /* Allocates, initializes and returns a new mutex. */
pthread_cond_t  *new_cond();             /* Allocates, initializes and returns a new condition variable. */
void print_state(char c, const char *s);  /* Prints the character, the system state, and the string. */
```

You can see the type specification of **struct bonding_arg**, and the three procedures that you are to write with this lab:

- **initialize_v()** is where you allocate and initialize your data for this lab. It is just like the **initialize_v()** calls in the printer simulation and the dining philosophers lecture. This is the **(void *)** that is passed to the hydrogen and oxygen threads. The argument to **initialize_v()** is the **verbosity** string. That lets you put in some print statements that you turn "on" and "off" at runtime.

- **hydrogen()** -- this is the hydrogen thread.

- **oxygen()** -- this is the oxygen thread.

The procedure **Bond()** is implemented in **bonding-driver.cpp**. This is what each thread calls when it makes a bond. Thus, for each bond, three threads will make identical calls to **Bond()**. The arguments of this call are the thread numbers of the first hydrogen thread, the second hydrogen thread, and the oxygen thread. Again, these arguments need to match for all three calls by all three threads in a bond. The **Bond()** call is going to double-check a bunch of things, and if everything is ok, it will return a string to each caller. The string will be the same string (but different copies). At that point, the thread that called **Bond()** should exit, (either via **return** or **pthread_exit()**), returning the string that was returned from **Bond()**. At that point, the joiner thread will join with it and double-check that the return value was ok.

You'll note that there are three helper functions in the header file, that are implemented in **bonding-driver.c**. These are:

- **new_mutex()** -- allocates and initializes a **pthread_mutex_t**.
- **new_cond()** -- allocates and initializes a **pthread_cond_t**.
- **print_state(char c, const char *s)**. This prints the character, then it prints the state of the simulation in terms of threads created, threads joined, hydrogen threads and oxygen threads, and then it prints the string. This is convenient for printing information, and I urge you to use it while you are debugging.

Verbosity

I have written **bonding-driver.c** so that:

- If the **verbosity** string has a 'C', then it will print out information about creating threads.
- If the **verbosity** string has a 'J', then it will print out information about joining threads.
- If the **verbosity** string has a 'B', then it will print out information when **Bond()** is called.

You can put other things into **verbosity** for when you debug. I'd suggest you use **print_state()**. My code does things like the following, to print out, for example, "Creator Thread Sleeping" when there is a 'C' in **verbosity**.

```
  if (strchr(G->verbosity, 'C') != NULL) {
    print_state('C', "Creator Thread Sleeping");
  }
```

Given that backdrop, what are you to do?

You are to implement **initialize_v()**, **hydrogen()** and **oxygen()** so that they work within the above framework. I'm going to give you an example which is in [bonding-example-1.c]() -- the code is commented inline, so give it a read:

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <pthread.h>
#include "bonding.h"

/* This solution only works with one molecule. */

struct global_info {
  pthread_mutex_t *lock;
  int h1;
  int h2;
  int o;
};

/* Initialize_v() sets h1/h2/o to -1 and initializes the lock. */

void *initialize_v(char *verbosity)
{
  struct global_info *g;

  g = (struct global_info *) malloc(sizeof(struct global_info));
  g->lock = new_mutex();
  g->h1 = -1;
  g->h2 = -1;
  g->o = -1;
  return (void *) g;
}

/* The hydrogen thread tries to set h1 or h2 to itself.  If both
   are already taken, then it simply returns NULL.  Otherwise,
   it sleeps to make sure that the molecule is ready, and it
   calls <b>Bond()</b>.
 */

void *hydrogen(void *arg)
{
  struct bonding_arg *a;
  struct global_info *g;
  char *rv;

  a = (struct bonding_arg *) arg;
  g = (struct global_info *) a->v;

  pthread_mutex_lock(g->lock);
  if (g->h1 == -1) {
    g->h1 = a->id;
  } else if (g->h2 == -1) {
    g->h2 = a->id;
  } else {
    pthread_mutex_unlock(g->lock);
    return NULL;
  }

  pthread_mutex_unlock(g->lock);

  sleep(1);   /* You aren't allowed to make this call.  I'm just doing this so that we can be
                 sure that the molecule is ready. You will need to use wait()/signal() do
                 to this correctly. */

  rv = Bond(g->h1, g->h2, g->o);
  return (void *) rv;
}

/* Similarly, the oxygen thread tries to set g->o to itself.  It
   g->o has already been set, then it returns NULL.  Otherwise, it
   sleeps to give the hydrogen threads time to run, and then it
   calls Bond() and returns the result. */

void *oxygen(void *arg)
{
  struct bonding_arg *a;
  struct global_info *g;
  char *rv;

  a = (struct bonding_arg *) arg;
  g = (struct global_info *) a->v;

  pthread_mutex_lock(g->lock);
  if (g->o == -1) {
    g->o = a->id;
  } else {
    pthread_mutex_unlock(g->lock);
    return NULL;
  }

  pthread_mutex_unlock(g->lock);

  sleep(1);   /* See the sleep() call above. */

  rv = Bond(g->h1, g->h2, g->o);
  return (void *) rv;
}
```

Here's some more explanation -- the **(void *)** holds a pointer to a **struct global_info**. This has a mutex and three integers: **h1**, **h2** and **o**. These are initialized to -1. Our **hydrogen** threads try to set **h1** to be their id. If **h1** is already set, then they try to set **h2**. If both **h1** and **h2** are already set, then they simply return NULL.

Similarly, the oxygen threads try to set **o** to be their ids, returning NULL if **o** is already set. The threads that have set the id's sleep for a second. This is bad form, and not allowed in your lab, but I'm doing this so you can see a simple solution. By the time that **sleep()** calls return, **h1**, **h2** and **o** should have been set. Each of the three threads calls **Bond(g->h1, g->h2, g->o)**. In other words the three calls are identical. All three return the return value of the **Bond()** call.

Let's run this on a system with one molecule. We'll set **verbosity** to "CJB" so that you can see the thread creation, thread joining and bonding calls:

```
UNIX> bonding-example-1 1 1 3 CJB
C: TF:    1 - TJ:    0 - HF:    1 - OF:    0 - Created thread with tid 0x700000187000.  Number: 0.  Type: h
C: TF:    2 - TJ:    0 - HF:    2 - OF:    0 - Created thread with tid 0x70000020a000.  Number: 1.  Type: h
C: TF:    3 - TJ:    0 - HF:    2 - OF:    1 - Created thread with tid 0x70000028d000.  Number: 2.  Type: o
C: TF:    3 - TJ:    0 - HF:    2 - OF:    1 - Creator thread is done.  Calling pthread_exit().
A second passes here.
B: TF:    3 - TJ:    0 - HF:    2 - OF:    1 - Thread 0 called Bond(0,1,2).
B: TF:    3 - TJ:    0 - HF:    2 - OF:    1 - Thread 2 called Bond(0,1,2).
B: TF:    3 - TJ:    0 - HF:    2 - OF:    1 - Thread 1 called Bond(0,1,2).
J: TF:    3 - TJ:    0 - HF:    2 - OF:    1 - Joining with thread number 0.  Tid 0x700000187000 (Bond (0,1,2))
J: TF:    3 - TJ:    1 - HF:    2 - OF:    1 - Join complete and verified: thread number 0.  Bond (0,1,2).
J: TF:    3 - TJ:    1 - HF:    2 - OF:    1 - Joining with thread number 1.  Tid 0x70000020a000 (Bond (0,1,2))
J: TF:    3 - TJ:    2 - HF:    2 - OF:    1 - Join complete and verified: thread number 1.  Bond (0,1,2).
J: TF:    3 - TJ:    2 - HF:    2 - OF:    1 - Joining with thread number 2.  Tid 0x70000028d000 (Bond (0,1,2))
J: TF:    3 - TJ:    3 - HF:    2 - OF:    1 - Join complete and verified: thread number 2.  Bond (0,1,2).
J: TF:    3 - TJ:    3 - HF:    2 - OF:    1 - The joiner is done.  Calling pthread_exit().
UNIX>
```

As you can see, the creator thread created threads 0 and 1, which are hydrogen threads, and thread 2, which is an oxygen thread. Thread 0 assigned **g->h1** to be 0, and thread 1 assigned **g->h2** to be 1. Thread 2 assigned **g->o** to be 2. After all the threads wait for a second, they all make their **Bond()** calls. Then they all join. The joiner thread verifies that the return values are all good, and the program exits.

If we run it on a seed of 3 rather than 1, you'll see the thread 0 is the oxygen thread, and everything still works.

```
UNIX> bonding-example-1 3 1 3 CJB
C: TF:    1 - TJ:    0 - HF:    0 - OF:    1 - Created thread with tid 0x700000187000.  Number: 0.  Type: o
C: TF:    2 - TJ:    0 - HF:    1 - OF:    1 - Created thread with tid 0x70000020a000.  Number: 1.  Type: h
C: TF:    3 - TJ:    0 - HF:    2 - OF:    1 - Created thread with tid 0x70000028d000.  Number: 2.  Type: h
C: TF:    3 - TJ:    0 - HF:    2 - OF:    1 - Creator thread is done.  Calling pthread_exit().
A second passes here.
B: TF:    3 - TJ:    0 - HF:    2 - OF:    1 - Thread 1 called Bond(1,2,0).
B: TF:    3 - TJ:    0 - HF:    2 - OF:    1 - Thread 0 called Bond(1,2,0).
B: TF:    3 - TJ:    0 - HF:    2 - OF:    1 - Thread 2 called Bond(1,2,0).
J: TF:    3 - TJ:    0 - HF:    2 - OF:    1 - Joining with thread number 1.  Tid 0x70000020a000 (Bond (1,2,0))
J: TF:    3 - TJ:    1 - HF:    2 - OF:    1 - Join complete and verified: thread number 1.  Bond (1,2,0).
```

```
J: TF:    3 - TJ:    1 - HF:    2 - OF:    1 - Joining with thread number 2.  Tid 0x70000028d000 (Bond (1,2,0))
J: TF:    3 - TJ:    2 - HF:    2 - OF:    1 - Join complete and verified: thread number 2.  Bond (1,2,0).
J: TF:    3 - TJ:    2 - HF:    2 - OF:    1 - Joining with thread number 0.  Tid 0x700000187000 (Bond (1,2,0))
J: TF:    3 - TJ:    3 - HF:    2 - OF:    1 - Join complete and verified: thread number 0.  Bond (1,2,0).
J: TF:    3 - TJ:    3 - HF:    2 - OF:    1 - The joiner is done.  Calling pthread_exit().
UNIX>
```

If we run it on two molecules, the program hangs, because threads 1, 4 and 5 all return NULL, without having called **Bond()**:

```
UNIX> bonding-example-1 3 2 6 CJB
C: TF:    1 - TJ:    0 - HF:    0 - OF:    1 - Created thread with tid 0x700000187000.  Number: 0.  Type: o
C: TF:    2 - TJ:    0 - HF:    0 - OF:    2 - Created thread with tid 0x70000020a000.  Number: 1.  Type: o
C: TF:    3 - TJ:    0 - HF:    1 - OF:    2 - Created thread with tid 0x70000028d000.  Number: 2.  Type: h
C: TF:    4 - TJ:    0 - HF:    2 - OF:    2 - Created thread with tid 0x700000310000.  Number: 3.  Type: h
C: TF:    5 - TJ:    0 - HF:    3 - OF:    2 - Created thread with tid 0x700000393000.  Number: 4.  Type: h
C: TF:    6 - TJ:    0 - HF:    4 - OF:    2 - Created thread with tid 0x700000416000.  Number: 5.  Type: h
C: TF:    6 - TJ:    0 - HF:    4 - OF:    2 - Creator thread is done.  Calling pthread_exit().
A second passes here.
B: TF:    6 - TJ:    0 - HF:    4 - OF:    2 - Thread 2 called Bond(2,3,0).
B: TF:    6 - TJ:    0 - HF:    4 - OF:    2 - Thread 3 called Bond(2,3,0).
B: TF:    6 - TJ:    0 - HF:    4 - OF:    2 - Thread 0 called Bond(2,3,0).
J: TF:    6 - TJ:    0 - HF:    4 - OF:    2 - Joining with thread number 2.  Tid 0x70000028d000 (Bond (2,3,0))
J: TF:    6 - TJ:    1 - HF:    4 - OF:    2 - Join complete and verified: thread number 2.  Bond (2,3,0).
J: TF:    6 - TJ:    1 - HF:    4 - OF:    2 - Joining with thread number 3.  Tid 0x700000310000 (Bond (2,3,0))
J: TF:    6 - TJ:    2 - HF:    4 - OF:    2 - Join complete and verified: thread number 3.  Bond (2,3,0).
J: TF:    6 - TJ:    2 - HF:    4 - OF:    2 - Joining with thread number 0.  Tid 0x700000187000 (Bond (2,3,0))
J: TF:    6 - TJ:    3 - HF:    4 - OF:    2 - Join complete and verified: thread number 0.  Bond (2,3,0).
The program hangs.
```

Take a look at another solution, in **bonding-example-2.c**:

```
/* This solution only works with one molecule,
   and if threads 0 and 1 are hydrogen,
   while thread 2 is oxygen. */

void *initialize_v(char *verbosity)
{
  return NULL;
}

void *hydrogen(void *arg)
{
  return Bond(0, 1, 2);
}

void *oxygen(void *arg)
{
  return Bond(0, 1, 2);
}
```

This one has all threads call **Bond(0, 1, 2)**. When we call it with a seed of 1 on one molecule, it works by complete happenstance, because threads 0 and 1 are hydrogens, and thread 2 is oxygen:

```
UNIX> bonding-example-2 1 1 3 CJB
C: TF:    1 - TJ:    0 - HF:    1 - OF:    0 - Created thread with tid 0x700000187000.  Number: 0.  Type: h
C: TF:    2 - TJ:    0 - HF:    2 - OF:    0 - Created thread with tid 0x70000020a000.  Number: 1.  Type: h
C: TF:    3 - TJ:    0 - HF:    2 - OF:    1 - Created thread with tid 0x70000028d000.  Number: 2.  Type: o
C: TF:    3 - TJ:    0 - HF:    2 - OF:    1 - Creator thread is done.  Calling pthread_exit().
B: TF:    3 - TJ:    0 - HF:    2 - OF:    1 - Thread 0 called Bond(0,1,2).
B: TF:    3 - TJ:    0 - HF:    2 - OF:    1 - Thread 1 called Bond(0,1,2).
B: TF:    3 - TJ:    0 - HF:    2 - OF:    1 - Thread 2 called Bond(0,1,2).
J: TF:    3 - TJ:    0 - HF:    2 - OF:    1 - Joining with thread number 0.  Tid 0x700000187000 (Bond (0,1,2))
J: TF:    3 - TJ:    1 - HF:    2 - OF:    1 - Join complete and verified: thread number 0.  Bond (0,1,2).
J: TF:    3 - TJ:    1 - HF:    2 - OF:    1 - Joining with thread number 1.  Tid 0x70000020a000 (Bond (0,1,2))
J: TF:    3 - TJ:    2 - HF:    2 - OF:    1 - Join complete and verified: thread number 1.  Bond (0,1,2).
J: TF:    3 - TJ:    2 - HF:    2 - OF:    1 - Joining with thread number 2.  Tid 0x70000028d000 (Bond (0,1,2))
J: TF:    3 - TJ:    3 - HF:    2 - OF:    1 - Join complete and verified: thread number 2.  Bond (0,1,2).
J: TF:    3 - TJ:    3 - HF:    2 - OF:    1 - The joiner is done.  Calling pthread_exit().
UNIX>
```

However, if you call it with a seed of 3, it fails, because thread 0 is oxygen, and the **Bond()** call is saying it's hydrogen:

```
UNIX> bonding-example-2 3 1 3 CJB
C: TF:    1 - TJ:    0 - HF:    0 - OF:    1 - Created thread with tid 0x700000187000.  Number: 0.  Type: o
C: TF:    2 - TJ:    0 - HF:    1 - OF:    1 - Created thread with tid 0x70000020a000.  Number: 1.  Type: h
C: TF:    3 - TJ:    0 - HF:    2 - OF:    1 - Created thread with tid 0x70000028d000.  Number: 2.  Type: h
C: TF:    3 - TJ:    0 - HF:    2 - OF:    1 - Creator thread is done.  Calling pthread_exit().
Error -- thread with id 0 called Bond(0,1,2).  It is of the wrong type for this bond.
UNIX>
```

If we call it with a seed of 1 and two molecules, you'll see that threads 0, 1 and 2 are all hydrogen, so the **Bond()** call fails again:

```
UNIX> bonding-example-2 1 2 6 CJB
C: TF:    1 - TJ:    0 - HF:    1 - OF:    0 - Created thread with tid 0x700000187000.  Number: 0.  Type: h
C: TF:    2 - TJ:    0 - HF:    2 - OF:    0 - Created thread with tid 0x70000020a000.  Number: 1.  Type: h
C: TF:    3 - TJ:    0 - HF:    3 - OF:    0 - Created thread with tid 0x70000028d000.  Number: 2.  Type: h
C: TF:    4 - TJ:    0 - HF:    4 - OF:    0 - Created thread with tid 0x700000310000.  Number: 3.  Type: h
C: TF:    5 - TJ:    0 - HF:    4 - OF:    1 - Created thread with tid 0x700000393000.  Number: 4.  Type: o
C: TF:    6 - TJ:    0 - HF:    4 - OF:    2 - Created thread with tid 0x700000416000.  Number: 5.  Type: o
C: TF:    6 - TJ:    0 - HF:    4 - OF:    2 - Creator thread is done.  Calling pthread_exit().
B: TF:    6 - TJ:    0 - HF:    4 - OF:    2 - Thread 0 called Bond(0,1,2).
B: TF:    6 - TJ:    0 - HF:    4 - OF:    2 - Thread 1 called Bond(0,1,2).
Error -- thread with id 2 called Bond(0,1,2).  It is of the wrong type for this bond.
UNIX>
```

If you call it with a seed of 2 and two molecules, then threads 0, 1 and 2 happen to work, but when thread 3 calls **Bond()**, it fails because thread three is not one of the atoms in the **Bond()** call.

```
UNIX> bonding-example-2 2 2 6 CJB
C: TF:    1 - TJ:    0 - HF:    1 - OF:    0 - Created thread with tid 0x700000187000.  Number: 0.  Type: h
C: TF:    2 - TJ:    0 - HF:    2 - OF:    0 - Created thread with tid 0x70000020a000.  Number: 1.  Type: h
C: TF:    3 - TJ:    0 - HF:    2 - OF:    1 - Created thread with tid 0x70000028d000.  Number: 2.  Type: o
C: TF:    4 - TJ:    0 - HF:    3 - OF:    1 - Created thread with tid 0x700000310000.  Number: 3.  Type: h
C: TF:    5 - TJ:    0 - HF:    4 - OF:    1 - Created thread with tid 0x700000393000.  Number: 4.  Type: h
C: TF:    6 - TJ:    0 - HF:    4 - OF:    2 - Created thread with tid 0x700000416000.  Number: 5.  Type: o
C: TF:    6 - TJ:    0 - HF:    4 - OF:    2 - Creator thread is done.  Calling pthread_exit().
B: TF:    6 - TJ:    0 - HF:    4 - OF:    2 - Thread 0 called Bond(0,1,2).
B: TF:    6 - TJ:    0 - HF:    4 - OF:    2 - Thread 1 called Bond(0,1,2).
B: TF:    6 - TJ:    0 - HF:    4 - OF:    2 - Thread 2 called Bond(0,1,2).
Error -- thread with id 3 called Bond(0,1,2).  The id has to be one of the atoms.
UNIX>
```

So, the bottom line here is that the driver tries to double-check you as much as possible. Your goal is to have the program run to completion without any errors.

Restrictions

The TA's will check for these by hand, and you won't get credit if you do any of these, so pay attention to them, and don't do them:

- No **sleep()** calls, or the equivalent (e.g. **usleep()**, **select()** with a timeout).
- No busy waiting -- i.e. no **while()** loops that continuously test a condition in hopes that another thread will change it, without doing the problem **wait()** calls on a condition variable. If you're in doubt -- ask a TA.
- No **pthread_yield()** calls.
- When a thread calls **Bond()** it is *not* allowed to hold any locked mutexes.
- I'd prefer that you free memory and destroy condition variables when your **hydrogen** and **oxygen** threads die. The TA's will deduct 5 points if you don't do this, and you'll lose more points if you run out of memory.

Strategy

There are many ways to do this assignment. Here are two ways that I implemented. The first is simpler than the second, but both are reasonable solutions. You may want to try your own solution. You should not try to solve this with just two condition variables -- that will be a very bug-prone approach, and may not work at all. **Solution #1: A list for waiting hydrogens and a list for waiting oxygens.** Your global data structure will have a mutex and two dllists. One for waiting hydrogens and one for waiting oxygens. Each thread has its own data structure too, which contains a condition variable, the thread's id, and id's of the three threads that will compose its molecule. (One of those three id's will be the thread's id, of course).

When a thread begins, it creates its data structure and locks the global mutex. It then looks at the waiting lists to see if it can create a molecule with itself and two of the waiting threads. If it cannot, then it adds itself to the appropriate list and blocks.

If it can create a molecule, then it sets the three thread id's for all three threads (itself and the two threads that are blocking), removes the two threads from their lists, and calls **pthread_cond_signal()** on them. And then it unlocks the mutex. The other two threads will return from their **pthread_cond_wait()** calls at some point, and then they will unlock the mutex.

At this point in a thread's execution, it has unlocked the mutex, and its three id's have been set, so it calls **Bond()** on the three ids and stores the return value. It then destroys its condition variable, frees up memory, and returns the return value.

I have an executable that works in this way in **bonding_list**. It supports the following verbosities:

- 'H' means that the hydrogen threads will print their states.
- 'O' means that the oxygen threads will print their states.
- 'G' means that periodically, the state of the system (the two lists) is printed.

Here's a run with seed=1 and two molecules. There's a lot of text here, but you can see how the threads hook up, and how the waiting hydrogen thread list grows and shrinks:

```
UNIX> bonding-list 1 2 6 HOGB
H: TF:    2 - TJ:    0 - HF:    3 - OF:    0 - Hydrogen thread 0 started & trying to lock mutex.
G: TF:    4 - TJ:    0 - HF:    4 - OF:    1 - Printing the state of the lists.
G: TF:    4 - TJ:    0 - HF:    4 - OF:    1 -   Waiting H list is empty.
G: TF:    4 - TJ:    0 - HF:    4 - OF:    1 -   Waiting O list is empty.
H: TF:    4 - TJ:    0 - HF:    4 - OF:    1 - Hydrogen thread 0 calling pthread_cond_wait()
G: TF:    4 - TJ:    0 - HF:    4 - OF:    1 - Printing the state of the lists.
G: TF:    4 - TJ:    0 - HF:    4 - OF:    1 -   Waiting H  0: 0.
G: TF:    4 - TJ:    0 - HF:    4 - OF:    1 -   Waiting O list is empty.
H: TF:    4 - TJ:    0 - HF:    4 - OF:    1 - Hydrogen thread 2 started & trying to lock mutex.
H: TF:    4 - TJ:    0 - HF:    4 - OF:    1 - Hydrogen thread 1 started & trying to lock mutex.
H: TF:    5 - TJ:    0 - HF:    4 - OF:    2 - Hydrogen thread 3 started & trying to lock mutex.
O: TF:    6 - TJ:    0 - HF:    4 - OF:    2 - Oxygen thread 4 started & trying to lock mutex.
O: TF:    6 - TJ:    0 - HF:    4 - OF:    2 - Oxygen thread 5 started & trying to lock mutex.
G: TF:    6 - TJ:    0 - HF:    4 - OF:    2 - Printing the state of the lists.
G: TF:    6 - TJ:    0 - HF:    4 - OF:    2 -   Waiting H  0: 0.
G: TF:    6 - TJ:    0 - HF:    4 - OF:    2 -   Waiting O list is empty.
H: TF:    6 - TJ:    0 - HF:    4 - OF:    2 - Hydrogen thread 2 calling pthread_cond_wait()
G: TF:    6 - TJ:    0 - HF:    4 - OF:    2 - Printing the state of the lists.
G: TF:    6 - TJ:    0 - HF:    4 - OF:    2 -   Waiting H  0: 0.
G: TF:    6 - TJ:    0 - HF:    4 - OF:    2 -   Waiting H  1: 2.
G: TF:    6 - TJ:    0 - HF:    4 - OF:    2 -   Waiting O list is empty.
G: TF:    6 - TJ:    0 - HF:    4 - OF:    2 - Printing the state of the lists.
G: TF:    6 - TJ:    0 - HF:    4 - OF:    2 -   Waiting H  0: 0.
G: TF:    6 - TJ:    0 - HF:    4 - OF:    2 -   Waiting H  1: 2.
G: TF:    6 - TJ:    0 - HF:    4 - OF:    2 -   Waiting O list is empty.
H: TF:    6 - TJ:    0 - HF:    4 - OF:    2 - Hydrogen thread 1 calling pthread_cond_wait()
G: TF:    6 - TJ:    0 - HF:    4 - OF:    2 - Printing the state of the lists.
G: TF:    6 - TJ:    0 - HF:    4 - OF:    2 -   Waiting H  0: 0.
G: TF:    6 - TJ:    0 - HF:    4 - OF:    2 -   Waiting H  1: 2.
G: TF:    6 - TJ:    0 - HF:    4 - OF:    2 -   Waiting H  2: 1.
G: TF:    6 - TJ:    0 - HF:    4 - OF:    2 -   Waiting O list is empty.
G: TF:    6 - TJ:    0 - HF:    4 - OF:    2 - Printing the state of the lists.
G: TF:    6 - TJ:    0 - HF:    4 - OF:    2 -   Waiting H  0: 0.
G: TF:    6 - TJ:    0 - HF:    4 - OF:    2 -   Waiting H  1: 2.
G: TF:    6 - TJ:    0 - HF:    4 - OF:    2 -   Waiting H  2: 1.
G: TF:    6 - TJ:    0 - HF:    4 - OF:    2 -   Waiting O list is empty.
H: TF:    6 - TJ:    0 - HF:    4 - OF:    2 - Hydrogen thread 3 calling pthread_cond_wait()
G: TF:    6 - TJ:    0 - HF:    4 - OF:    2 - Printing the state of the lists.
G: TF:    6 - TJ:    0 - HF:    4 - OF:    2 -   Waiting H  0: 0.
G: TF:    6 - TJ:    0 - HF:    4 - OF:    2 -   Waiting H  1: 2.
G: TF:    6 - TJ:    0 - HF:    4 - OF:    2 -   Waiting H  2: 1.
G: TF:    6 - TJ:    0 - HF:    4 - OF:    2 -   Waiting H  3: 3.
G: TF:    6 - TJ:    0 - HF:    4 - OF:    2 -   Waiting O list is empty.
G: TF:    6 - TJ:    0 - HF:    4 - OF:    2 - Printing the state of the lists.
G: TF:    6 - TJ:    0 - HF:    4 - OF:    2 -   Waiting H  0: 0.
G: TF:    6 - TJ:    0 - HF:    4 - OF:    2 -   Waiting H  1: 2.
```

```
G: TF:    6 - TJ:    0 - HF:    4 - OF:    2 -   Waiting H  2: 1.
G: TF:    6 - TJ:    0 - HF:    4 - OF:    2 -   Waiting H  3: 3.
G: TF:    6 - TJ:    0 - HF:    4 - OF:    2 -   Waiting O list is empty.
O: TF:    6 - TJ:    0 - HF:    4 - OF:    2 - Creating bond with h1=0, h2=2 and o=myself=4
O: TF:    6 - TJ:    0 - HF:    4 - OF:    2 - Oxygen thread 4 calling bond(0,2,4)
B: TF:    6 - TJ:    0 - HF:    4 - OF:    2 - Thread 4 called Bond(0,2,4).
G: TF:    6 - TJ:    0 - HF:    4 - OF:    2 - Printing the state of the lists.
G: TF:    6 - TJ:    0 - HF:    4 - OF:    2 -   Waiting H  0: 1.
G: TF:    6 - TJ:    0 - HF:    4 - OF:    2 -   Waiting H  1: 3.
G: TF:    6 - TJ:    0 - HF:    4 - OF:    2 -   Waiting O list is empty.
O: TF:    6 - TJ:    0 - HF:    4 - OF:    2 - Creating bond with h1=1, h2=3 and o=myself=5
O: TF:    6 - TJ:    0 - HF:    4 - OF:    2 - Oxygen thread 5 calling bond(1,3,5)
B: TF:    6 - TJ:    0 - HF:    4 - OF:    2 - Thread 5 called Bond(1,3,5).
H: TF:    6 - TJ:    0 - HF:    4 - OF:    2 - Hydrogen thread 2 calling bond(0,2,4)
H: TF:    6 - TJ:    0 - HF:    4 - OF:    2 - Hydrogen thread 0 calling bond(0,2,4)
H: TF:    6 - TJ:    0 - HF:    4 - OF:    2 - Hydrogen thread 3 calling bond(1,3,5)
H: TF:    6 - TJ:    0 - HF:    4 - OF:    2 - Hydrogen thread 1 calling bond(1,3,5)
B: TF:    6 - TJ:    0 - HF:    4 - OF:    2 - Thread 2 called Bond(0,2,4).
B: TF:    6 - TJ:    0 - HF:    4 - OF:    2 - Thread 0 called Bond(0,2,4).
B: TF:    6 - TJ:    0 - HF:    4 - OF:    2 - Thread 3 called Bond(1,3,5).
B: TF:    6 - TJ:    0 - HF:    4 - OF:    2 - Thread 1 called Bond(1,3,5).
UNIX>
```

This run shows what happens when an oxygen thread is created first:

```
UNIX> bonding-list 3 1 3 HOGB
O: TF:    3 - TJ:    0 - HF:    2 - OF:    1 - Oxygen thread 0 started & trying to lock mutex.
H: TF:    3 - TJ:    0 - HF:    2 - OF:    1 - Hydrogen thread 1 started & trying to lock mutex.
H: TF:    3 - TJ:    0 - HF:    2 - OF:    1 - Hydrogen thread 2 started & trying to lock mutex.
G: TF:    3 - TJ:    0 - HF:    2 - OF:    1 - Printing the state of the lists.
G: TF:    3 - TJ:    0 - HF:    2 - OF:    1 -   Waiting H list is empty.
G: TF:    3 - TJ:    0 - HF:    2 - OF:    1 -   Waiting O list is empty.
O: TF:    3 - TJ:    0 - HF:    2 - OF:    1 - Oxygen thread 0 calling pthread_cond_wait()
G: TF:    3 - TJ:    0 - HF:    2 - OF:    1 - Printing the state of the lists.
G: TF:    3 - TJ:    0 - HF:    2 - OF:    1 -   Waiting H list is empty.
G: TF:    3 - TJ:    0 - HF:    2 - OF:    1 -   Waiting O  0: 0.
G: TF:    3 - TJ:    0 - HF:    2 - OF:    1 - Printing the state of the lists.
G: TF:    3 - TJ:    0 - HF:    2 - OF:    1 -   Waiting H list is empty.
G: TF:    3 - TJ:    0 - HF:    2 - OF:    1 -   Waiting O  0: 0.
H: TF:    3 - TJ:    0 - HF:    2 - OF:    1 - Hydrogen thread 1 calling pthread_cond_wait()
G: TF:    3 - TJ:    0 - HF:    2 - OF:    1 - Printing the state of the lists.
G: TF:    3 - TJ:    0 - HF:    2 - OF:    1 -   Waiting H  0: 1.
G: TF:    3 - TJ:    0 - HF:    2 - OF:    1 -   Waiting O  0: 0.
G: TF:    3 - TJ:    0 - HF:    2 - OF:    1 - Printing the state of the lists.
G: TF:    3 - TJ:    0 - HF:    2 - OF:    1 -   Waiting H  0: 1.
G: TF:    3 - TJ:    0 - HF:    2 - OF:    1 -   Waiting O  0: 0.
H: TF:    3 - TJ:    0 - HF:    2 - OF:    1 - Creating bond with h1=myself=2, h2=1 and o=0
H: TF:    3 - TJ:    0 - HF:    2 - OF:    1 - Hydrogen thread 2 calling bond(2,1,0)
B: TF:    3 - TJ:    0 - HF:    2 - OF:    1 - Thread 2 called Bond(2,1,0).
H: TF:    3 - TJ:    0 - HF:    2 - OF:    1 - Hydrogen thread 1 calling bond(2,1,0)
O: TF:    3 - TJ:    0 - HF:    2 - OF:    1 - Oxygen thread 0 calling bond(2,1,0)
B: TF:    3 - TJ:    0 - HF:    2 - OF:    1 - Thread 1 called Bond(2,1,0).
B: TF:    3 - TJ:    1 - HF:    2 - OF:    1 - Thread 0 called Bond(2,1,0).
UNIX>
```

Solution #2: Lists for incomplete molecules. In my second solution, I did not create data structures for each thread. Instead, I created a data structure for each molecule. This data structure had the following fields:

- The id's of hydrogen atom 1, hydrogen atom 2, and the oxygen atom.
- Condition variables for hydrogen atom 1, hydrogen atom 2, and the oxygen atom.
- A counter to record when to free the data structure.

I also had three Dllists to hold incomplete molecules:

- **Needs_hydrogen** for molecules that are only missing hydrogen atoms.
- **Needs_oxygen** for molecules that are only missing their oxygen atoms.
- **Needs_oh** for molecules that are missing one hydrogen and one oxygen atom.

Now, when a hydrogen thread gets the mutex, it checks **needs_hydrogen** to see if there are any incomplete molecules that need hydrogen. If there is one, then it removes it from the list.

If **needs_hydrogen** is empty, then it checks **needs_oh**, and if there is a molecule there, it removes it from the list.

If both of those lists are empty, then it creates a new molecule.

Now, it adds itself to the molecule, and if the molecule is still incomplete, it adds the molecule to the proper list and blocks. If the molecule is complete, then it signals the two threads that are blocking, and they all call **Bond()**. The last of these frees the molecule and destroys its condition variables (this is why we have the counter).

Obviously, the oxygen threads work similarly, checking **needs_oxyben** and **needs_oh**. If both of these are empty, then it creates a new molecule.

As with the first solution, I have put in code to print the hydrogen and oxygen threads as they go through their various states, and global information to print out the states of the Dllists:

```
UNIX> bonding-molecule 1 2 6 HOGB
H: TF:    2 - TJ:    0 - HF:    2 - OF:    0 - New hydrogen thread 0 has locked the mutex.
G: TF:    5 - TJ:    0 - HF:    4 - OF:    2 - Printing the lists.
G: TF:    5 - TJ:    0 - HF:    4 - OF:    2 -   needs_hydrogen is empty.
G: TF:    6 - TJ:    0 - HF:    4 - OF:    2 -   needs_oxygen is empty.
G: TF:    6 - TJ:    0 - HF:    4 - OF:    2 -   needs_oh is empty.
H: TF:    6 - TJ:    0 - HF:    4 - OF:    2 - Hydrogen 0 creating a new molecule.
H: TF:    6 - TJ:    0 - HF:    4 - OF:    2 - Hydrogen 0.  Appending molecule to needs_oh and blocking.
G: TF:    6 - TJ:    0 - HF:    4 - OF:    2 - Printing the lists.
G: TF:    6 - TJ:    0 - HF:    4 - OF:    2 -   needs_hydrogen is empty.
G: TF:    6 - TJ:    0 - HF:    4 - OF:    2 -   needs_oxygen is empty.
G: TF:    6 - TJ:    0 - HF:    4 - OF:    2 -   needs_oh 0: (0,-1,-1)
H: TF:    6 - TJ:    0 - HF:    4 - OF:    2 - New hydrogen thread 1 has locked the mutex.
G: TF:    6 - TJ:    0 - HF:    4 - OF:    2 - Printing the lists.
G: TF:    6 - TJ:    0 - HF:    4 - OF:    2 -   needs_hydrogen is empty.
G: TF:    6 - TJ:    0 - HF:    4 - OF:    2 -   needs_oxygen is empty.
G: TF:    6 - TJ:    0 - HF:    4 - OF:    2 -   needs_oh 0: (0,-1,-1)
H: TF:    6 - TJ:    0 - HF:    4 - OF:    2 - Hydrogen 1 grabbing molecule from 'needs_oh'. H1=0
H: TF:    6 - TJ:    0 - HF:    4 - OF:    2 - Hydrogen 1.  Appending molecule to needs_oxygen and blocking.
G: TF:    6 - TJ:    0 - HF:    4 - OF:    2 - Printing the lists.
G: TF:    6 - TJ:    0 - HF:    4 - OF:    2 -   needs_hydrogen is empty.
G: TF:    6 - TJ:    0 - HF:    4 - OF:    2 -   needs_oxygen 0: (0,1,-1)
G: TF:    6 - TJ:    0 - HF:    4 - OF:    2 -   needs_oh is empty.
H: TF:    6 - TJ:    0 - HF:    4 - OF:    2 - New hydrogen thread 2 has locked the mutex.
G: TF:    6 - TJ:    0 - HF:    4 - OF:    2 - Printing the lists.
G: TF:    6 - TJ:    0 - HF:    4 - OF:    2 -   needs_hydrogen is empty.
G: TF:    6 - TJ:    0 - HF:    4 - OF:    2 -   needs_oxygen 0: (0,1,-1)
G: TF:    6 - TJ:    0 - HF:    4 - OF:    2 -   needs_oh is empty.
H: TF:    6 - TJ:    0 - HF:    4 - OF:    2 - Hydrogen 2 creating a new molecule.
H: TF:    6 - TJ:    0 - HF:    4 - OF:    2 - Hydrogen 2.  Appending molecule to needs_oh and blocking.
G: TF:    6 - TJ:    0 - HF:    4 - OF:    2 - Printing the lists.
G: TF:    6 - TJ:    0 - HF:    4 - OF:    2 -   needs_hydrogen is empty.
G: TF:    6 - TJ:    0 - HF:    4 - OF:    2 -   needs_oxygen 0: (0,1,-1)
G: TF:    6 - TJ:    0 - HF:    4 - OF:    2 -   needs_oh 0: (2,-1,-1)
H: TF:    6 - TJ:    0 - HF:    4 - OF:    2 - New hydrogen thread 3 has locked the mutex.
G: TF:    6 - TJ:    0 - HF:    4 - OF:    2 - Printing the lists.
G: TF:    6 - TJ:    0 - HF:    4 - OF:    2 -   needs_hydrogen is empty.
G: TF:    6 - TJ:    0 - HF:    4 - OF:    2 -   needs_oxygen 0: (0,1,-1)
G: TF:    6 - TJ:    0 - HF:    4 - OF:    2 -   needs_oh 0: (2,-1,-1)
H: TF:    6 - TJ:    0 - HF:    4 - OF:    2 - Hydrogen 3 grabbing molecule from 'needs_oh'. H1=2
H: TF:    6 - TJ:    0 - HF:    4 - OF:    2 - Hydrogen 3.  Appending molecule to needs_oxygen and blocking.
G: TF:    6 - TJ:    0 - HF:    4 - OF:    2 - Printing the lists.
G: TF:    6 - TJ:    0 - HF:    4 - OF:    2 -   needs_hydrogen is empty.
G: TF:    6 - TJ:    0 - HF:    4 - OF:    2 -   needs_oxygen 0: (0,1,-1)
G: TF:    6 - TJ:    0 - HF:    4 - OF:    2 -   needs_oxygen 1: (2,3,-1)
G: TF:    6 - TJ:    0 - HF:    4 - OF:    2 -   needs_oh is empty.
O: TF:    6 - TJ:    0 - HF:    4 - OF:    2 - New oxygen thread 4 has locked the mutex.
G: TF:    6 - TJ:    0 - HF:    4 - OF:    2 - Printing the lists.
G: TF:    6 - TJ:    0 - HF:    4 - OF:    2 -   needs_hydrogen is empty.
G: TF:    6 - TJ:    0 - HF:    4 - OF:    2 -   needs_oxygen 0: (0,1,-1)
G: TF:    6 - TJ:    0 - HF:    4 - OF:    2 -   needs_oxygen 1: (2,3,-1)
G: TF:    6 - TJ:    0 - HF:    4 - OF:    2 -   needs_oh is empty.
O: TF:    6 - TJ:    0 - HF:    4 - OF:    2 - Oxygen 4 grabbing molecule from 'needs_oxygen'. H1=0.  H2=1.
O: TF:    6 - TJ:    0 - HF:    4 - OF:    2 - Oxygen 4.  Molecule complete - calling pthread_cond_signal() on
O: TF:    6 - TJ:    0 - HF:    4 - OF:    2 - New oxygen thread 5 has locked the mutex.
B: TF:    6 - TJ:    0 - HF:    4 - OF:    2 - Thread 4 called Bond(0,1,4).
G: TF:    6 - TJ:    0 - HF:    4 - OF:    2 - Printing the lists.
G: TF:    6 - TJ:    0 - HF:    4 - OF:    2 -   needs_hydrogen is empty.
```

```
G: TF:    6 - TJ:    0 - HF:    4 - OF:    2 -   needs_oxygen 0: (2,3,-1)
G: TF:    6 - TJ:    0 - HF:    4 - OF:    2 -   needs_oh is empty.
O: TF:    6 - TJ:    0 - HF:    4 - OF:    2 - Oxygen 5 grabbing molecule from 'needs_oxygen'. H1=2.  H2=3.
O: TF:    6 - TJ:    0 - HF:    4 - OF:    2 - Oxygen 5.  Molecule complete - calling pthread_cond_signal() on
B: TF:    6 - TJ:    0 - HF:    4 - OF:    2 - Thread 5 called Bond(2,3,5).
H: TF:    6 - TJ:    0 - HF:    4 - OF:    2 - Hydrogen 0 is unblocked.
H: TF:    6 - TJ:    0 - HF:    4 - OF:    2 - Hydrogen 1 is unblocked.
H: TF:    6 - TJ:    0 - HF:    4 - OF:    2 - Hydrogen 1 is freeing its molecule and destroying condition vari
B: TF:    6 - TJ:    0 - HF:    4 - OF:    2 - Thread 0 called Bond(0,1,4).
B: TF:    6 - TJ:    0 - HF:    4 - OF:    2 - Thread 1 called Bond(0,1,4).
H: TF:    6 - TJ:    0 - HF:    4 - OF:    2 - Hydrogen 2 is unblocked.
H: TF:    6 - TJ:    1 - HF:    4 - OF:    2 - Hydrogen 3 is unblocked.
H: TF:    6 - TJ:    1 - HF:    4 - OF:    2 - Hydrogen 3 is freeing its molecule and destroying condition vari
B: TF:    6 - TJ:    1 - HF:    4 - OF:    2 - Thread 2 called Bond(2,3,5).
B: TF:    6 - TJ:    1 - HF:    4 - OF:    2 - Thread 3 called Bond(2,3,5).
UNIX>
```

Two notes here, in case you're paying attention to that output:

1. I actually free the molecule before calling **Bond()**, copying the ids of h1, h2 and o to local variables before the **free()** call. That way, I can hold the mutex when I call **free()**. (This is because you're not allowed to hold the mutex when you call **Bond()**.
2. When I need two hydrogen atoms, I append the molecule to **needs_hydrogen**, and when I need one, I prepend it. That way, I prefer molecules that are closer to being completed.

Grading

I have provided a gradescript. The TA's are going to grade this as follows:

- They will compile your **bonding.c** file with my **bonding.h** and **bonding-driver.c** (and of course **libfdr.a**).
- They will run a couple of sample runs with verbosity equal to "CJB" to verify that your program is running nicely within the structure of my code.
- Then they will run the **gradeall**. If your program hangs, that will be a failure. Cases 76-100 are slower (since they fork off > 20,000 threads).