Chris Park Lab 6

Problem 1 Output & Answer:

Verbose:

```
Philosopher 1 is Thinking...
take: Philosopher 1 is locking mutex
Philosopher 1 is Hungry!
put: Philosopher 1 is signaling wakeup
take: Philosopher 1 is unclocking mutex
take: Philosopher 1 is waiting
Philosopher 1 is Eating.
put: Philosopher 1 is locking mutex
Philosopher 1 is unlocking mutex
Philosopher 1 is Thinking...
take: Philosopher 1 is locking mutex
put: Philosopher 4 is signaling wakeup
Philosopher 3 is unlocking mutex
Philosopher 1 is Hungry!
put: Philosopher 1 is signaling wakeup
take: Philosopher 1 is unclocking mutex
take: Philosopher 1 is waiting
Philosopher 1 is Eating.
put: Philosopher 1 is locking mutex
put: Philosopher 2 is signaling wakeup
Philosopher 1 is unlocking mutex
Philosopher 1 is Thinking...
```

Non Verbose:

```
Philosopher 3 is Thinking...
Philosopher 3 is Hungry!
Philosopher 3 is Eating.
Philosopher 3 is Thinking...
Philosopher 3 is Hungry!
Philosopher 3 is Eating.
Philosopher 0 is Eating.
Philosopher 0 is Thinking...
Philosopher 0 is Hungry!
Philosopher 1 is Eating.
Philosopher 1 is Thinking...
Philosopher 1 is Hungry!
Philosopher 1 is Eating.
Philosopher 1 is Thinking...
Philosopher 1 is Hungry!
```

A) Each philosopher starts out by thinking. It then attempts to pick up the fork, locking down the mutex for the critical section inside of takefork. It's state is then changed to hungry and a call to test is made. From here its state is changed to eating. After returning from the test it unlocks the critical region and locks the mutex for the fork, preventing other threads from using the fork. Its state is changed back to thinking in the call to putfork and it tests its neighbors to give them a chance to eat if necessary. Having then released the lock on the fork and the cycle continues. From the output it would appear that one process can dominate for a lengthy period of time, there are also a couple of instances in the verbose set of output where you can see the threads running concurrently when they do not interfere with each other. These happen when threads wake up, and once to unlock a mutex.

Problem 2 Output & Answer:

```
Writing to database
Reading from database
Reading from database
Writing to database
Reading from database
Reading from database
Writing to database
Reading from database
Reading from database
Writing to database
Reading from database
Reading from database
Writing to database
Reading from database
Reading from database
Writing to database
Reading from database
```

A) There is no starvation going on here, which was the desired result. Based on the output it looks like multiple readers have to read before they allow the writer to change the database. This appears to be due to using pthread_cond_broadcast to wake up all the readers instead of just one at a time with pthread_cond_signal. If I switch to a signal call, the readers and the writer alternate. I'm not sure if N number of readers would result in starvation in the case of using broadcast. The reader checks to see if there is a writer waiting to write to the database, If there is it waits and allows the writer to execute. Once the writer is done it signals for the readers to wake back up and continue reading.