Question 1:

helgrind reported the following problems with main-race.c.

helgrind output:

```
david@yogata:~/Desktop/422/exam2/part1$ valgrind --tool=helgrind ./main-race
==5304== Helgrind, a thread error detector
==5304== Copyright (C) 2007-2015, and GNU GPL'd, by OpenWorks LLP et al.
==5304== Using Valgrind-3.11.0 and LibVEX; rerun with -h for copyright info
==5304== Command: ./main-race
==5304==
==5304== ---Thread-Announcement------------------------------------------
==5304==
==5304== Thread #1 is the program's root thread
==5304==
==5304== ---Thread-Announcement------------------------------------------
==5304==
==5304== Thread #2 was created
==5304==    at 0x5163B1E: clone (clone.S:74)
==5304==    by 0x4E46189: create_thread (createthread.c:102)
==5304==    by 0x4E47EC3: pthread_create@@GLIBC_2.2.5 (pthread_create.c:679)
==5304==    by 0x4C34BB7: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-linux.so)
==5304==    by 0x400BAE: Pthread_create (mythreads.h:51)
==5304==    by 0x400C72: main (main-race.c:14)
==5304==
==5304== ----------------------------------------------------------------
==5304==
==5304== Possible data race during read of size 4 at 0x60208C by thread #1
==5304== Locks held: none
==5304==    at 0x400C73: main (main-race.c:15)
==5304==
==5304== This conflicts with a previous write of size 4 by thread #2
==5304== Locks held: none
==5304==    at 0x400C2D: worker (main-race.c:8)
==5304==    by 0x4C34DB6: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-linux.so)
==5304==    by 0x4E476F9: start_thread (pthread_create.c:333)
==5304==  Address 0x60208c is 0 bytes inside data symbol "balance"
==5304==
==5304== ----------------------------------------------------------------
==5304==
==5304== Possible data race during write of size 4 at 0x60208C by thread #1
==5304== Locks held: none
==5304==    at 0x400C7C: main (main-race.c:15)
==5304==
==5304== This conflicts with a previous write of size 4 by thread #2
==5304== Locks held: none
==5304==    at 0x400C2D: worker (main-race.c:8)
==5304==    by 0x4C34DB6: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-linux.so)
==5304==    by 0x4E476F9: start_thread (pthread_create.c:333)
==5304==  Address 0x60208c is 0 bytes inside data symbol "balance"
==5304==
==5304==
==5304== For counts of detected and suppressed errors, rerun with: -v
==5304== Use --history-level=approx or =none to gain increased speed, at
==5304== the cost of reduced accuracy of conflicting-access information
==5304== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 0 from 0)
```

In summary, it reports that lines *8* and *15* conflict because of concurrent writes to the same memory address. Helgrind gives other useful information, such as the name of the variable "balance" and its smemory address.

Question2

When removing line 15, helgrind reported this.

helgrind output:

```
==5380== Helgrind, a thread error detector
==5380== Copyright (C) 2007-2015, and GNU GPL'd, by OpenWorks LLP et al.
==5380== Using Valgrind-3.11.0 and LibVEX; rerun with -h for copyright info
==5380== Command: ./main-race
==5380==
==5380==
==5380== For counts of detected and suppressed errors, rerun with: -v
==5380== Use --history-level=approx or =none to gain increased speed, at
==5380== the cost of reduced accuracy of conflicting-access information
==5380== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
david@yogata:~/Desktop/422/exam2/part1$
```

There are no longer any race conditions.

Code is reproduced below.

main-race.c:

```
#include <stdio.h>

#include "mythreads.h"

int balance = 0;

void* worker(void* arg) {
    balance++; // unprotected access
    return NULL;
}

int main(int argc, char *argv[]) {
    pthread_t p;
    Pthread_create(&p, NULL, worker, NULL);
    Pthread_join(p, NULL);
    return 0;
}
```

Case1: Adding one lock.

I added a lock around line 15, and reran valgrind. I recived the following output.

helgrind output:

```
==5484== Helgrind, a thread error detector
==5484== Copyright (C) 2007-2015, and GNU GPL'd, by OpenWorks LLP et al.
==5484== Using Valgrind-3.11.0 and LibVEX; rerun with -h for copyright info
==5484== Command: ./main-race
```

```
==5484==
==5484== ---Thread-Announcement------------------------------------------
==5484==
==5484== Thread #1 is the program's root thread
==5484==
==5484== ---Thread-Announcement------------------------------------------
==5484==
==5484== Thread #2 was created
==5484==    at 0x5163B1E: clone (clone.S:74)
==5484==    by 0x4E46189: create_thread (createthread.c:102)
==5484==    by 0x4E47EC3: pthread_create@@GLIBC_2.2.5 (pthread_create.c:679)
==5484==    by 0x4C34BB7: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-linux.so)
==5484==    by 0x400BAE: Pthread_create (mythreads.h:51)
==5484==    by 0x400C72: main (main-race.c:15)
==5484==
==5484== ----------------------------------------------------------------
==5484==
==5484==  Lock at 0x6020E0 was first observed
==5484==    at 0x4C321BC: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-linux.so)
==5484==    by 0x400C7C: main (main-race.c:16)
==5484==  Address 0x6020e0 is 0 bytes inside data symbol "lock"
==5484==
==5484== Possible data race during read of size 4 at 0x6020C0 by thread #1
==5484== Locks held: 1, at address 0x6020E0
==5484==    at 0x400C7D: main (main-race.c:17)
==5484==
==5484== This conflicts with a previous write of size 4 by thread #2
==5484== Locks held: none
==5484==    at 0x400C2D: worker (main-race.c:9)
==5484==    by 0x4C34DB6: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-linux.so)
==5484==    by 0x4E476F9: start_thread (pthread_create.c:333)
==5484==  Address 0x6020c0 is 0 bytes inside data symbol "balance"
==5484==
==5484== ----------------------------------------------------------------
==5484==
==5484==  Lock at 0x6020E0 was first observed
==5484==    at 0x4C321BC: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-linux.so)
==5484==    by 0x400C7C: main (main-race.c:16)
==5484==  Address 0x6020e0 is 0 bytes inside data symbol "lock"
==5484==
==5484== Possible data race during write of size 4 at 0x6020C0 by thread #1
==5484== Locks held: 1, at address 0x6020E0
==5484==    at 0x400C86: main (main-race.c:17)
==5484==
==5484== This conflicts with a previous write of size 4 by thread #2
==5484== Locks held: none
==5484==    at 0x400C2D: worker (main-race.c:9)
==5484==    by 0x4C34DB6: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-linux.so)
==5484==    by 0x4E476F9: start_thread (pthread_create.c:333)
==5484==  Address 0x6020c0 is 0 bytes inside data symbol "balance"
==5484==
==5484==
==5484== For counts of detected and suppressed errors, rerun with: -v
==5484== Use --history-level=approx or =none to gain increased speed, at
==5484== the cost of reduced accuracy of conflicting-access information
==5484== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 0 from 0)
```

It reports that there are data races around both times balance is incremented. Helgrind does mention that main holds one lock; however, worker holds none. Worker's independence of locks makes main's lock useless.

Code below.

main-race.c:

```
#include <stdio.h>

#include "mythreads.h"

int balance = 0;
pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;

void* worker(void* arg) {
    balance++; // unprotected access
    return NULL;
}

int main(int argc, char *argv[]) {
    pthread_t p;
    Pthread_create(&p, NULL, worker, NULL);
    pthread_mutex_lock(&lock);
    balance++; // unprotected access
    pthread_mutex_unlock(&lock);
    Pthread_join(p, NULL);
    return 0;
}
```

Case 2: Two locks.

I added locking around both problematic sections of code, and reran helgrind.

helgrind output:

```
==5565== Helgrind, a thread error detector
==5565== Copyright (C) 2007-2015, and GNU GPL'd, by OpenWorks LLP et al.
==5565== Using Valgrind-3.11.0 and LibVEX; rerun with -h for copyright info
==5565== Command: ./main-race
==5565==
==5565==
==5565== For counts of detected and suppressed errors, rerun with: -v
==5565== Use --history-level=approx or =none to gain increased speed, at
==5565== the cost of reduced accuracy of conflicting-access information
==5565== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 7 from 7)
```

main-race.c:

```
#include <stdio.h>

#include "mythreads.h"

int balance = 0;
pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;

void* worker(void* arg) {
  pthread_mutex_lock(&lock);
  balance++; // unprotected access
  pthread_mutex_unlock(&lock);
  return NULL;
}

int main(int argc, char *argv[]) {
    pthread_t p;
    Pthread_create(&p, NULL, worker, NULL);
    pthread_mutex_lock(&lock);
    balance++; // unprotected access
    pthread_mutex_unlock(&lock);
    Pthread_join(p, NULL);
    return 0;
}
```

In conclusion, adding locks around the problematic lines of code prevented the data race.

Question 3

The deadlock in **main-deadlock.c**_ is due to neither thread being able to acquire both locks (m1 & m2). Imagine the case where p1 acquires m1 and p2 acquires m2. p1 will not release m1 until it acquires m2, and p2 will not release m2 until it acquires m1. Execution will stall.

Question 4

Helgrind Output:

```
==5642== Helgrind, a thread error detector
==5642== Copyright (C) 2007-2015, and GNU GPL'd, by OpenWorks LLP et al.
==5642== Using Valgrind-3.11.0 and LibVEX; rerun with -h for copyright info
==5642== Command: ./main-deadlock
==5642==
==5642== ---Thread-Announcement------------------------------------------
==5642==
==5642== Thread #3 was created
==5642==    at 0x5163B1E: clone (clone.S:74)
==5642==    by 0x4E46189: create_thread (createthread.c:102)
==5642==    by 0x4E47EC3: pthread_create@@GLIBC_2.2.5 (pthread_create.c:679)
==5642==    by 0x4C34BB7: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-linux.so)
==5642==    by 0x400BAE: Pthread_create (mythreads.h:51)
==5642==    by 0x400CC9: main (main-deadlock.c:24)
==5642==
==5642== ----------------------------------------------------------------
==5642==
==5642== Thread #3: lock order "0x6020C0 before 0x602100" violated
==5642==
==5642== Observed (incorrect) order is: acquisition of lock at 0x602100
==5642==    at 0x4C321BC: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-linux.so)
==5642==    by 0x400A4F: Pthread_mutex_lock (mythreads.h:23)
==5642==    by 0x400C50: worker (main-deadlock.c:13)
==5642==    by 0x4C34DB6: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-linux.so)
==5642==    by 0x4E476F9: start_thread (pthread_create.c:333)
==5642==
==5642==  followed by a later acquisition of lock at 0x6020C0
==5642==    at 0x4C321BC: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-linux.so)
==5642==    by 0x400A4F: Pthread_mutex_lock (mythreads.h:23)
==5642==    by 0x400C5A: worker (main-deadlock.c:14)
```

```
==5642==    by 0x4C34DB6: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-linux.so)
==5642==    by 0x4E476F9: start_thread (pthread_create.c:333)
==5642==
==5642== Required order was established by acquisition of lock at 0x6020C0
==5642==    at 0x4C321BC: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-linux.so)
==5642==    by 0x400A4F: Pthread_mutex_lock (mythreads.h:23)
==5642==    by 0x400C3A: worker (main-deadlock.c:10)
==5642==    by 0x4C34DB6: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-linux.so)
==5642==    by 0x4E476F9: start_thread (pthread_create.c:333)
==5642==
==5642==  followed by a later acquisition of lock at 0x602100
==5642==    at 0x4C321BC: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-linux.so)
==5642==    by 0x400A4F: Pthread_mutex_lock (mythreads.h:23)
==5642==    by 0x400C44: worker (main-deadlock.c:11)
==5642==    by 0x4C34DB6: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-linux.so)
==5642==    by 0x4E476F9: start_thread (pthread_create.c:333)
==5642==
==5642==  Lock at 0x6020C0 was first observed
==5642==    at 0x4C321BC: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-linux.so)
==5642==    by 0x400A4F: Pthread_mutex_lock (mythreads.h:23)
==5642==    by 0x400C3A: worker (main-deadlock.c:10)
==5642==    by 0x4C34DB6: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-linux.so)
==5642==    by 0x4E476F9: start_thread (pthread_create.c:333)
==5642==  Address 0x6020c0 is 0 bytes inside data symbol "m1"
==5642==
==5642==  Lock at 0x602100 was first observed
==5642==    at 0x4C321BC: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-linux.so)
==5642==    by 0x400A4F: Pthread_mutex_lock (mythreads.h:23)
==5642==    by 0x400C44: worker (main-deadlock.c:11)
==5642==    by 0x4C34DB6: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-linux.so)
==5642==    by 0x4E476F9: start_thread (pthread_create.c:333)
==5642==  Address 0x602100 is 0 bytes inside data symbol "m2"
==5642==
==5642==
==5642==
==5642== For counts of detected and suppressed errors, rerun with: -v
==5642== Use --history-level=approx or =none to gain increased speed, at
==5642== the cost of reduced accuracy of conflicting-access information
==5642== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 9 from 9)
```

Helgrind does not use the term "deadlock", but it does report that **"lock order has been violated"**. This is essentially deadlock (e.g. one thread acquiring one lock before releasing a lock another needs).

Question 5

Helgrind Output:

```
==5766== Helgrind, a thread error detector
==5766== Copyright (C) 2007-2015, and GNU GPL'd, by OpenWorks LLP et al.
==5766== Using Valgrind-3.11.0 and LibVEX; rerun with -h for copyright info
==5766== Command: ./main-deadlock-global
==5766==
==5766== ---Thread-Announcement------------------------------------------
==5766==
==5766== Thread #3 was created
==5766==    at 0x5163B1E: clone (clone.S:74)
==5766==    by 0x4E46189: create_thread (createthread.c:102)
```

```
==5766==    by 0x4E47EC3: pthread_create@@GLIBC_2.2.5 (pthread_create.c:679)
==5766==    by 0x4C34BB7: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-linux.so)
==5766==    by 0x400BAE: Pthread_create (mythreads.h:51)
==5766==    by 0x400CDD: main (main-deadlock-global.c:27)
==5766==
==5766== ----------------------------------------------------------------
==5766==
==5766== Thread #3: lock order "0x602100 before 0x602140" violated
==5766==
==5766== Observed (incorrect) order is: acquisition of lock at 0x602140
==5766==    at 0x4C321BC: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-linux.so)
==5766==    by 0x400A4F: Pthread_mutex_lock (mythreads.h:23)
==5766==    by 0x400C5A: worker (main-deadlock-global.c:15)
==5766==    by 0x4C34DB6: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-linux.so)
==5766==    by 0x4E476F9: start_thread (pthread_create.c:333)
==5766==
==5766==  followed by a later acquisition of lock at 0x602100
==5766==    at 0x4C321BC: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-linux.so)
==5766==    by 0x400A4F: Pthread_mutex_lock (mythreads.h:23)
==5766==    by 0x400C64: worker (main-deadlock-global.c:16)
==5766==    by 0x4C34DB6: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-linux.so)
==5766==    by 0x4E476F9: start_thread (pthread_create.c:333)
==5766==
==5766== Required order was established by acquisition of lock at 0x602100
==5766==    at 0x4C321BC: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-linux.so)
==5766==    by 0x400A4F: Pthread_mutex_lock (mythreads.h:23)
==5766==    by 0x400C44: worker (main-deadlock-global.c:12)
==5766==    by 0x4C34DB6: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-linux.so)
==5766==    by 0x4E476F9: start_thread (pthread_create.c:333)
==5766==
==5766==  followed by a later acquisition of lock at 0x602140
==5766==    at 0x4C321BC: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-linux.so)
==5766==    by 0x400A4F: Pthread_mutex_lock (mythreads.h:23)
==5766==    by 0x400C4E: worker (main-deadlock-global.c:13)
==5766==    by 0x4C34DB6: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-linux.so)
==5766==    by 0x4E476F9: start_thread (pthread_create.c:333)
==5766==
==5766==  Lock at 0x602100 was first observed
==5766==    at 0x4C321BC: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-linux.so)
==5766==    by 0x400A4F: Pthread_mutex_lock (mythreads.h:23)
==5766==    by 0x400C44: worker (main-deadlock-global.c:12)
==5766==    by 0x4C34DB6: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-linux.so)
==5766==    by 0x4E476F9: start_thread (pthread_create.c:333)
==5766==  Address 0x602100 is 0 bytes inside data symbol "m1"
==5766==
==5766==  Lock at 0x602140 was first observed
==5766==    at 0x4C321BC: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-linux.so)
==5766==    by 0x400A4F: Pthread_mutex_lock (mythreads.h:23)
==5766==    by 0x400C4E: worker (main-deadlock-global.c:13)
==5766==    by 0x4C34DB6: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-linux.so)
==5766==    by 0x4E476F9: start_thread (pthread_create.c:333)
==5766==  Address 0x602140 is 0 bytes inside data symbol "m2"
==5766==
==5766==
==5766==
==5766== For counts of detected and suppressed errors, rerun with: -v
==5766== Use --history-level=approx or =none to gain increased speed, at
==5766== the cost of reduced accuracy of conflicting-access information
==5766== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 9 from 9)
```

Helgrind reports the same **"lock order violated"** error. It should not report this error as there is no deadlock in the program. Originally, there could be a deadlock if p1 attempts to acquire m2 before p2 relinquishes it. This can no longer happen because p1 would have to acquire g before preforming any other operations. It is only possible to obtain g if m1 and m2 are both unlocked. Helgrind does not realize this, and reports the same error.

Question 6

This code is ineficient because it preforms and I/O operation before setting done to 1. As such, the main thread spins for a long time waiting for the I/O operation to first complete and then for done to be set to 1. What is worse is that cpu cycles are spent in the main thread jumping to the same address do to the while statement.

Question 7

helgrind output:

```
==5877== Helgrind, a thread error detector
==5877== Copyright (C) 2007-2015, and GNU GPL'd, by OpenWorks LLP et al.
==5877== Using Valgrind-3.11.0 and LibVEX; rerun with -h for copyright info
==5877== Command: ./main-signal
==5877==
this should print first
==5877== ---Thread-Announcement------------------------------------------
==5877==
==5877== Thread #1 is the program's root thread
==5877==
==5877== ---Thread-Announcement------------------------------------------
==5877==
==5877== Thread #2 was created
==5877==    at 0x5163B1E: clone (clone.S:74)
==5877==    by 0x4E46189: create_thread (createthread.c:102)
==5877==    by 0x4E47EC3: pthread_create@@GLIBC_2.2.5 (pthread_create.c:679)
==5877==    by 0x4C34BB7: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-linux.so)
==5877==    by 0x400BFE: Pthread_create (mythreads.h:51)
==5877==    by 0x400CCB: main (main-signal.c:15)
==5877==
==5877== ----------------------------------------------------------------
==5877==
==5877== Possible data race during read of size 4 at 0x602094 by thread #1
==5877== Locks held: none
==5877==    at 0x400CCD: main (main-signal.c:16)
==5877==
==5877== This conflicts with a previous write of size 4 by thread #2
==5877== Locks held: none
==5877==    at 0x400C82: worker (main-signal.c:9)
==5877==    by 0x4C34DB6: ??? (in /usr/lib/valgrind/vgpreload_helgrind-amd64-linux.so)
==5877==    by 0x4E476F9: start_thread (pthread_create.c:333)
==5877==  Address 0x602094 is 0 bytes inside data symbol "done"
==5877==
this should print last
==5877==
==5877== For counts of detected and suppressed errors, rerun with: -v
==5877== Use --history-level=approx or =none to gain increased speed, at
==5877== the cost of reduced accuracy of conflicting-access information
==5877== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 67 from 44)
```

Helgrind reports that there is a race condition on the "done" variable as one thread is loading a value that may be changed by the worker thread. The race condition is that the load instruction of done and the store instruction can happen concurrently. However, this only results in main spinning 1 more time than it otherwise would have. This method is likely faster than dealing with the overhead of using locks, but atomic instructions would be more useful. The program is **"correct"** if the specification is that the line "this should print first" comes before "this should print last", which it always will.

Question 8

"main-signal-cv" is prefered over "main-signal" because it has better performance (the main thread does not spin). Both pieces of code are correct.

Question 9

helgrind output:

```
==6343== Helgrind, a thread error detector
==6343== Copyright (C) 2007-2015, and GNU GPL'd, by OpenWorks LLP et al.
==6343== Using Valgrind-3.11.0 and LibVEX; rerun with -h for copyright info
==6343== Command: ./main-signal-cv
==6343==
this should print first
this should print last
==6343==
==6343== For counts of detected and suppressed errors, rerun with: -v
==6343== Use --history-level=approx or =none to gain increased speed, at
==6343== the cost of reduced accuracy of conflicting-access information
==6343== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 7 from 7)
```

Helgrind reported no errors.