Problem 1 - Huey, Dewey, and Louie

```
# initialize all the semaphores
# with a being set to one means
# that Huey doesn't need to wait.

semaphore a
semaphore b
semaphore c

init(a, 0)
init(b, 0)
init(c, 0)
```

```
while true:
  Huey(i)
  signal(b)
  wait(a)
  i++
```

```
while true:
  wait(b)
  Dewey(i)
  signal(c)
```

```
while true:
  wait(c)
  Louie(i)
  signal(a)
```

Problem 2 - Producer/Consumer

I implemented the producer consumer program in `p2.c`. I set each thread to wait a random amount of time and was easily able to get the following output:

```
% ./p2
```

```
wait(mutex);
take();
n = -1;
consume();
wait(mutex);
take();
```

```
n = -2;
consume();
produce();
wait(mutex);
append();
n = -1;
wait(mutex);
take();
n = -2;
consume();
```

This shows that the consumer thread can consume before the producer thread can put an item onto the buffer.

The main issue is that the mutex is open to either thread and whichever thread gets there first, wins the race. Thus there is a race condition in the program.

Problem 3 - Roller Coaster

```
semaphore loading
semaphore loaded
semaphore unloading
semaphore unloaded

int n
int c

# function that represents the passengers
def passenger(id):
  while true:
    wait(loading)
    board()
    signal(loaded)
    wait(unloading)
    unboard()
    signal(unloaded)

# function that represents the roller
# coaster car
def roller_coaster():
  while true:
    load()
    for i = 0 to c:
      signal(loading)
    for i = 0 to c:
      wait(loaded)
    run()
    unload()
    for i = 0 to c:
```

```
      signal(unloading)

    for i = 0 to c:
      wait(unloaded)

def main():

  n = 50
  c = 10

  # initialize all the semaphores
  init(loaded, 0)
  init(loading 0)
  init(unloaded, 0)
  init(unloading, 0)

  # spawn all the threads
  thread_create(roller_coaster)
  for i = 0 to n:
    thread_create(passenger, i)
```

Problem 4 - inc.c

NOTE: all of the source files have a target defined in the `Makefile` for easy testability.

a. See included source code (inc.c)

b. Max Threads

The following is the info for my test machine. The modified code is in the `inc-max.c` source file.

CPU info

```
% cat /proc/cpuinfo
```

```
processor : 0
vendor_id : GenuineIntel
cpu family  : 6
model   : 69
model name  : Intel(R) Core(TM) i3-4010U CPU @ 1.70GHz
stepping  : 1
microcode : 0x14
cpu MHz   : 1700.000
cache size  : 3072 KB
physical id : 0
siblings  : 4
core id   : 0
```

```
cpu cores : 2
apicid    : 0
initial apicid  : 0
fpu   : yes
fpu_exception : yes
cpuid level : 13
wp    : yes
```

Operating System

```
% uname -a
```

```
Linux archy 3.18.6-1-ARCH #1 SMP PREEMPT Sat Feb 7 08:44:05 CET 2015
x86_64 GNU/Linux
```

gcc version

```
% gcc --version
```

```
gcc (GCC) 4.9.2 20150204 (prerelease)
Copyright (C) 2014 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There
is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR
PURPOSE.
```

Results

```
% ./inc-max
```

```
Enter number of threads to create (< 500000):
50000
max threads = 32751 before error
```

It seems that my laptop can at max can only spawn 32751 threads before it dies.

c. See included source code `p1.c` and output `p1.out`.

d. See included source code `p3.c` and output `p3.out`.