1)

I think the best implementation would be to have a global counter, equal to system clock time. In a while loop you could keep track of the milliseconds of system time. First you would need to tell the thread to wait on the conditional queue, and then you could signal, or broadcast depending on how many threads you want to release.

Pthread\_cond\_t cond;

Cond = PTHREAD\_COND\_INITIALIZE;

thread.pthread\_cond\_wait(&cond);

Clock\_t howLongIWantToWait = 10000;

Clock\_t t = clock();

howLongIWantToWait = howLongIWantToWait + t;

while(t < howLongiWantToWait)

T = clock();

pthread\_cond\_signal(&cond);

2)

You would use a producer/consumer methodology with semaphores to implement this. One semaphore used for full, one for empty,and one for locking. The producer function would be handleIncomingconnection(). It would look something like:

Semaphore full(0);

Semaphore empty(n);

Semaphore mutex(1);

HandleNextIncomingConnection() CloseExistingConnection()

{ {

//get connection full.P()

empty.P() mutex.P()

mutex.P() //remove connection

//add connection mutex.V()

mutex.V() empty.V()

full.V() }

}

3)

Three processes can be in the critical section. Because when the mistake of calling V() before P() happens, the semaphore value goes to 2. This leaves room for 2 more threads, while the mistake thread is already in the critical section, which sums to 3 threads at one time in the critical section.

4)

Semaphore full(0);

Semaphore empty\_male(3);

Semaphore empty\_female(3);

Semaphore lock(1);

for(;;)

{

if(female)

{

while(empty\_male.value < 3);

empty\_female.P()

lock.P()

//inside bathroom

lock.V()

full.V()

//leaving bathroom

full.P()

lock.P()

//walks out

lock.V()

empty\_female.V()

}

}