NAME:
FIRST NAME:

Question 1

What is a "deadlock" ? What circumstances are at the origin?

This is a deadlock that occurs between multiple threads or processes, when everyone tries to access a resource that the other has while blocking access to a resource that the other covets. It is also called "Deadlock".
For a deadlock to occur, 4 circumstances must be met:
- Access to resources is in mutual exclusion
- Each process or thread expects to be able to lock a second resource without releasing the first he locked.
- It is not possible to requisition a resource in possession of one of the threads or processes
- There is a circular wait. Example: thread1 has resource A and wants B, thread2 has B and wants C, thread3 has C and wants A.

Wanting to solve the previous problem we can cause a "livelock"? What is it ?

If we force each thread / process to release the first locked resource after a certain amount of time
waiting without being able to get the second resource, it may be that each thread will end up locking the
first resource then release it after a while without ever getting both: we also get a
blocking, but this time the processes are not stuck waiting. The threads / processes repeat indefinitely the same operation, so they are alive, but are locked in a
cycle.

Question 2

Comment on the following pieces of code to explain what they do (the // ... indicate parts of the code that have been omitted):

```
thread 1
// ...
pthread_mutex_lock (m); // this mutex protects the conditional variable and 'data'
if (data == 0) // we check that 'data' is at 0 before stopping waiting
pthread_cond_wait (c, m); // wait for a signal
// ...
data--; // the signal is received, the mutex is re-locked, we can modify data
pthread_mutex_unlock (m); // unlock the mutex
// ...
thread 2
// ...
pthread_mutex_lock (mutex); // this mutex protects the conditional variable and 'data'
data ++;
// modify 'data'
// ...
pthread_cond_signal (cond_var); // signal to a thread the change
pthread_mutex_unlock (mutex); // unlock the mutex at the end
```

// ...
Why is it necessary to protect a wait on a conditional variable with a mutex?
Before you hang on hold, the thread must verify that the variable has not already been modified. If so, a signal may have already been sent, but it was lost because he did not expect it at that time. The mutex is locked to ensure that the variable can not be changed while testing its value and before block pending.
pthread_cond_wait() se charge de déverrouiller le mutex avant de placer le thread en attente pour qu'un autrepthread_cond_wait () will unblock the mutex before placing the thread on hold for another thread can modify the variable and send a signal.

Question 3
In the producer / consumer model how much mutex and semaphores are we using and what is it do they serve?

In this model at least one mutex is used to avoid concurrent access to a list of used buffers by the producer to store the data produced and by the consumer to retrieve this data. We can multiply the number of mutexes to protect each buffer separately.
Two semaphores are also used, one to indicate the number of full buffers, and the other to indicate the number of buffers.
number of empty buffers.