Posix threads condition variables exercises

For a C++ program you save your program with the extension .cpp to your program name. For a C++ program compile your program with the following command:

```
g++   –o   yourprogram    yourprogram.cpp    –l pthread
```

for a C program you save your program with the extension .c to your program name and you just use gcc instead like this

```
gcc   –o   yourprogram    headerfile.c  yourprogram.c    –l pthread
```

If you have no errors then you can execute your program with the following command: `./yourprogram`

Exercise: a ringbuffer monitor

It is perfectly fine to implement the ringbuffer in C. Put the class interface in a .h file and the implementation including the member variables in a .c file, where you include the .h fil. Finally, in your main file you include your .h file. You will find the ringbuffer example of how to build an object in C in it'slearning.

First step is to implement mutual exclusion for the dequeue and enqueue functions using a mutex

We have two dangerous situations in the ringbuffer:

Empty buffer: make a condition variable called not_empty. In the dequeue function you must wait at it if the queue is empty. In enqueue you must signal the not-full condition variable, when an element has been inserted.

Likewise create a not_full condition variable for handling the situation that the ringbuffer is full when trying to insert an element.

Exercise: a mailbox monitor

Now make a copy of your ringbuffer and change it into a mailbox by changing the element type to a void pointer: void*. Rebaptize enqueue to put, and dequeue to get. This means that you can put addresses of different kind of messages in your mailbox, but the receiver has to be aware of the type of information the void pointer points at.

Then make two threads sender and receiver. The sender puts the address of an item in the mailbox, and the receiver gets this address from the mailbox, and reads the information by type-casting the received void pointer to the correct pointer type.

Add non-blocking versions of put and get to the mail-box: try_put and try_get. The threads calling these functions should inspect the return values of these functions in order to check whether the function was successful or not. The member variable actual_size will tell you what to return. For instance if you call try_get then a NULL (in C++: nullptr) will be returned if actual_size is 0 (remember to lock the monitor mutex before reading actual_size and remember to unlock the monitor mutex before returning NULL). Try_put returns true or false to indicate whether the put was successful or not. Do not forget to signal the relevant condition variables if the function calls are successful.