OS3: Assessed Coding Exercise

Sheet 1 (of 2)
NB: Sheet 2 will only be available after completion of the first stage

This is an assessed exercise, which will count for 12% of the marks in this course. This sheet is intended to provide an initial overview and allow you to begin designing your solution and implementing key components. Sheet 2 will be made available once you demonstrate that you have engaged with the exercise by submitting the first part on Moodle.

This exercise is structured as follows: The current document describes the basic problem and indicates what is needed as a solution. The exercise involves the construction of a single piece of a complex software system, and will require the use of the Pthread concurrency facilities in C. As the exercise progresses, further information (e.g. header files and test harnesses) will be made available, however all necessary information to begin the exercise is provided in this sheet.

Note that it is not acceptable to use lab time that should be dedicated to other courses for work on this exercise. Spot checks may be undertaken to ensure you aren't disrupting your other studies in order to further this piece of work.

Also, be careful not to spend too long on the exercise. If you have been steadily improving your C coding, through the OS lab sessions, and are on top of the concurrency material from AP3, it will be manageable in about twelve hours - *provided you follow the advice about how to tackle the exercise, and seek guidance where you are allowed to and need to*. If you do not tackle this exercise in the suggested fashion, it has the potential to become very time consuming - remember that it is only worth 12% of the course.

NOTE THAT YOU SHOULD **NOT** START WRITING ANY CODE UNTIL YOU'VE RECEIVED THE SOURCE CODE ARCHIVE! It will be a waste of your time if you do.

General Overview

A piece of code is required which will function as a (de)multiplexor for a simple network driver. Calls will be made to this software component to pass packets of data onto the network. Packets arriving from the network will be demultiplexed and passed to appropriate waiting callers. The emphasis is on the concurrency, not on networking issues.

The core of your solution will be one or more threads, which move packets of data between queues or buffers (amongst other things). You will be programming in C, and using the PThreads libraries. PThreads allow you to dynamically create threads and provides support for both mutual exclusion locks and for condition variables (which support a simple wait/signal mechanism and allow a timeout on waits).

You should start the design of your solution now, sketching out what you are trying to achieve and the rough code needed to do it. Your sketch will take the form of an interaction diagram.

Detailed Problem

You are to deliver a piece of code which provides the central functionality of a network device driver. The abstract data type (ADT) you are producing will be called NetworkDriver, and you will make calls onto an instance of the NetworkDevice ADT. You will also receive calls back from that device and calls from the application level, as well.

Your code will be passing around pointers to PacketDescriptors; one of the components of the system is a PacketDescriptor constructor; this will be handed a region of memory (actually a void pointer and a length) and it divides that into pieces each the right size to hold a PacketDescriptor. The PacketDescriptors are then passed to another component, the FreePacketDescriptorStore. The FreePacketDescriptorStore is an unbounded container, which will be used by your code and the test harness as a place from which to acquire PacketDescriptors. When they are no longer used, the PacketDescriptors must be returned to the FreePacketDescriptorStore.

You will use an existing piece of code for the FreePacketDescriptorStore, along with other components of the test harness. Similarly you will use the BoundedBuffer from the Pthreads introduction.

The test harness includes fake applications, which will acquire PacketDescriptors from the store, initialise their contents and then pass them to your code for delivery onto the network.

The test harness will also include a fake NetworkDevice to which you can pass PacketDescriptors for dispatch. The complication is that the NetworkDevice can only process one packet at a time, and is quite slow in doing so, therefore you may have to queue up other requests. An application thread should not normally have to wait for its packet to be placed on the network. Once it has handed it to your code, it should be allowed to return and engage in other activity. This means you are likely to need dedicated threads within the code to pass packets to the network device.

Once a packet has been sent onto the network, the code should pass the descriptor back to the store for free packet descriptors.

The NetworkDevice will also pass packets from the network which will entail having a PacketDescriptor ready and waiting and a thread blocked ready to handle the pseudo-interrupt saying a packet has arrived. This requires to set-up a new empty packet descriptor to await the next packet and block as soon as possible waiting for it. Consequently any processing of the recently arrived packet must be minimal, handing it over to another (buffered?) part of the code for dispatch by a different thread.

Packets will need to be passed to the fake application layer when they are requested, and eventually the packet descriptors will be returned to the free packet descriptor store by the fake applications.

Because this code is supposed to live inside the OS, it is required to use bounded sized data structures within the parts of the code that handle the passing of packet descriptors between the applications and the network device. It is possible that application threads will block, either because they are awaiting incoming packets, because the buffers for outgoing packets are full (as those data structures are supposed to be bounded), or because there are no free packet descriptors available.

The thread that handles incoming packets must not block, except to await an incoming packet. So either: the thread that processes the received packets must also not block; or, if it does and the buffer for received packets subsequently becomes full, the packet receiver thread must discard packets rather than blocking on a full buffer.

First Activities and Acquiring Sheet 2

Sheet 2 will contain a rough outline architecture for the software for the system. However, you must work through the description above, identifying the key features of the problem and determining how the pieces interact before sheet 2 is issued. You are allowed to discuss this problem sheet with your classmates, however you should develop your own view of how the pieces fit together, and what each piece does prior to sheet 2 being issued. There will be a short Q&A session during a lecture next week covering this exercise. Do not get bogged down trying to work out how to express your ideas in C at this first stage - make sure you understand what you are trying to achieve (bearing in mind the sorts of facility available to you) before you start working on how to express those actions in C with Pthreads.

Sheet 2 will be issued after you've submitted on Moodle a rough sketch (worth 25% of the marks) showing the various components of this exercise, including: the different threads you've identified as necessary; the buffers etc and whether they are bounded or unbounded; and how data flows through your system.

Preparation of this sketch should not involve a computer, but it should be clear and readable! It should take no more than 10-15 minutes after reading through this first problem sheet and spending an hour or so thinking about the problem to be solved.

NOTE THAT YOU SHOULD **NOT** START WRITING ANY CODE UNTIL YOU'VE RECEIVED THE SOURCE CODE ARCHIVE! It will be a waste of your time if you do.