CI583: Data Structures and Operating Systems Scheduling

Shared Servers

The time-sharing approach deals only in *threads*, so it may be entirely unfair in the context of several users.

If, for instance user A is running a single-threaded process and user B is running a process that spawns 9 threads, the previous approach will strongly favour user B.

On a Shared Server (SS), we need to account for time in terms of the user as well as the thread. This is known as proportional-share scheduling – each user gets their fair share of processor time.

Shared Servers

One approach to this is called lottery scheduling (Waldspurger and Weil, 1994).

Each user is given an equal number of "tickets", say 10. These tickets are distributed as evenly as possible among that users' threads.

Shared Servers

So, to continue the previous example, User A has a single thread with 10 tickets and User B has 8 threads with a single ticket and one thread with two tickets.

The winning ticket is picked at random.

Real-time Systems

Real-time systems are mainly used in specialist applications.

They give strict guarantees about the order in which threads will execute (so lottery scheduling would be totally unacceptable!).

In a soft RTS such as a DVD player, a response that is slower then expected may mean that playback hangs, while the users watches a spinning wheel in irritation.

Real-time Systems

In a hard RTS such as some software used in a nuclear reactor, a response that is slower than expected may mean shutting down the reactor and evacuating thousands of homes.

Scheduling for hard RTS is a complex issue and beyond the scope of this lecture. See (Doeppner, 2011) for more detail.

Soft RTS

We can extend the strategies used by non-real-time systems such as Windows and Linux by giving certain threads very high priorities.

Such threads preempt other threads and can never be preempted themselves.

This will give a fast response (needed for soft RTS) but not necessarily a dependable one (needed for hard RTS).

Soft RTS

In order to make response time as fast as possible we need to think about the following:

1. Interrupt processing: interrupts will preempt any thread, including high priority ones.

   By using *deferred work techniques*, we can carry out most of the work that would normally take place in an interrupt context somewhere else, i.e. in a context that can be preempted by a high priority thread.

Soft RTS

2. Caching and paging: we can arrange for the address spaces of high priority threads to be *pinned* in memory, i.e. kept in the cache until the thread ends.

Soft RTS

3. Resource acquisition: when a high priority thread, T_1, needs to acquire a resource, such as access to a particular buffer, we can move T_1 to the front of the queue but the resource may be held by a low priority thread, T_2.

   To ensure that the resource is freed as soon as possible, we can temporarily promote T_2 to have the same priority as T_1.

Multi-core systems

How should scheduling work when we have several processors?

Assuming that all processors can access all memory (symmetric multiprocessor), a simplistic approach is to use one of the strategies described above and have each processor take jobs from the same queue.

Multi-core systems

This causes two problems:

1. Contention for the queue: processors may need to wait for exclusive access to the queue.
2. Caching: if a thread that has been run on processor P_1 is run there again, its address space will probably be in the cache. Thus, there is an advantage to assigning a thread to a particular processor.

Multi-core systems

So, it makes sense to have multiple queues, one per processor, solving both of these problems.

In order to make sure each processor is reasonably busy, we can carry out some simple load balancing.

Networks

From our point of view, the network interface card is pretty similar to the terminal – it just sends and receives data.

Data arrives whether or not there is an outstanding read request.
Data arrives in packets, rather than lines or characters.

Networks

The big difference is in the performance requirements.

A terminal receives input in terms of several chars per second.

Our NIC will be required to process millions or billions of bits per second.

What's more, the device needs to behave in accordance with a particular communications protocol, or set of formats and rules for sending and receiving data.

Networks

Consider TCP, the transmission control protocol.

Data being sent by TCP is organised into self-describing packets.

The receiver sends acknowledgement when it receives a packet successfully.

If the sender doesn't receive this acknowledgement by a given time limit, it tries sending the packet again.

Networks

So, our driver needs to be able to do the following:

1. Pass packets of data from one module to the next (unlike the terminal module, we need to pass by reference in order to keep up, rather than copying data from one buffer to another).
2. Append headers which describe an outgoing packet and remove these headers from incoming packets.

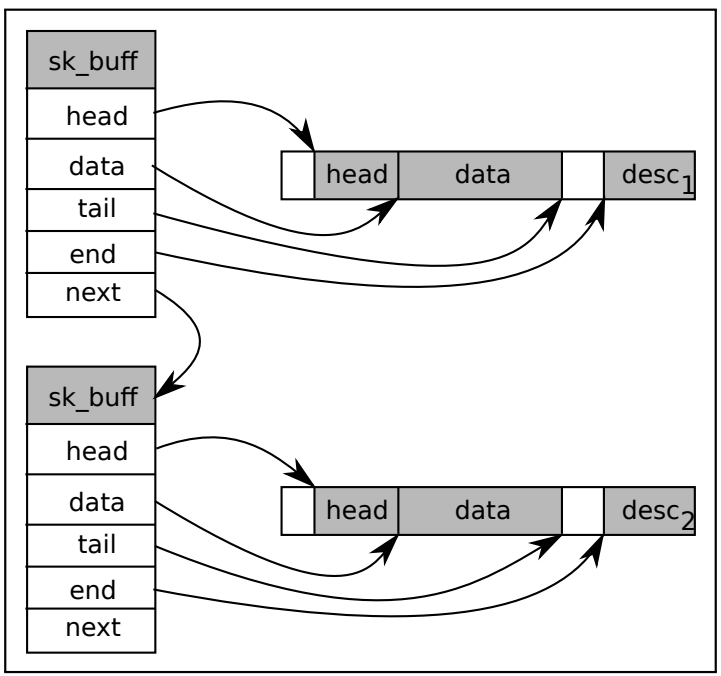
Networks

3. Ensure that outgoing packets are held onto until acknowledgement is received, so that they can be retransmitted if necessary.
4. Request and respond to time-out notifications.

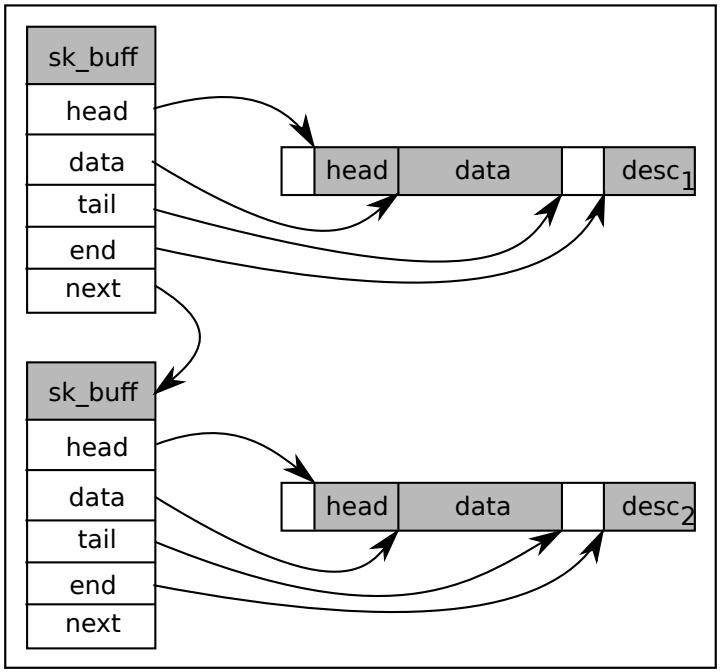
Networks

This is handled in the following way by Linux:

Each packet is represented by a data structure called an `sk_buff` (*socket buffer*), which holds a collection of pointers to the *head*, *data*, *tail* and so on for that packet. It also has a pointer to the next segment in this transmission, which may be empty.



Networks



`tail` refers to the *current* end of the data, so that more can be added as it arrives at the module. `end` points to the maximum allowable address. `desc` contains a reference count. When copies are made of packets (so that they can be retransmitted or passed to another module) the reference count is increased. When one of these copies is no longer needed the reference count is decremented.

Next time

File systems!

- Early file systems,
- UNIX S5FS,
- DOS FAT,
- Crash resiliency and journalled filesystems,
- multiple disks (RAID etc), and
- flash memory.