Suppose we have a real-time system supporting three devices: a keyboard whose interrupt handler has a service time of 800 us, a disk with a service time of 500 us, and a printer with a service time of 400 us.

What is the worst-case latency seen by each device?

For now we'll assume that requests are infrequent, i.e., that each request only happens once in each scenario.

Requests can arrive at any time and in any order.

If we serve the requests in first-come-first-served order, each device might be delayed by the service of all other devices.

So the start of the keyboard handler might be delayed by the execution of the disk and printer handlers, a worst-case latency of 900 us.

The start of the disk handler might be delayed by the keyboard and printer handlers, a worst-case latency of 1200 us.

And the printer handler might be delayed by the keyboard and disk handlers, a worst-case latency of 1300 us.

In this scenario we see that long-running handlers have a huge impact on the worst-case latency seen by the other devices.

What are the possibilities for reducing the worst-case latencies?

Is there a better scheduling algorithm than first-come-first-served?

One strategy is to assign priorities to the pending requests and to serve the requests in priority order.

If the handlers are uninterruptible, the priorities will be used to select the *next* task to be run at the completion of the current task.

Note that under this strategy, the current task always runs to completion even if a higher-priority request arrives while it's executing.

This is called a "nonpreemptive" or "weak" priority system.

Using a weak priority system, the worst-case latency seen by each device is the worst-case service time of all the other devices (since that handler may have just started running when the new request arrives), plus the service time of all higher-priority devices (since they'll be run first).

In our example, suppose we assigned the highest priority to the disk, the next priority to the printer, and the lowest priority to the keyboard.

The worst-case latency of the keyboard is unchanged since it has the lowest priority and hence can be delayed by the higher-priority disk and printer handlers.

The disk handler has the highest priority and so will always be selected for execution after the current handler completes.

So its worst-case latency is the worst-case service time for the currently-running handler, which in this case is the keyboard.

So the worst-case latency for the disk is 800 us.

This is a considerable improvement over the first-come-first-served scenario.

Finally the worst-case scenario for the printer is 1300 us since it may have to wait for the keyboard handler to finish (which can take up to 800 us) and then for a higher-priority disk request to be serviced (which takes 500 us).

How should priorities be assigned given hard real-time constraints?

We'll assume each device has a service deadline D after the arrival of its service request.

If not otherwise specified, assume D is the time until the *next* request for the same device.

This is a reasonably conservative assumption that prevents the system from falling further and further behind.

For example, it makes sense that the keyboard handler should finish processing one character before the next arrives.

"Earliest Deadline" is a strategy for assigning priorities that is guaranteed to meet the deadlines if any priority assignment can meet the deadlines.

It's very simple: Sort the requests by their deadlines.

Assign the highest priority to the earliest deadline, second priority to the next deadline, and so on.

A weak priority system will choose the pending request with the highest priority, i.e., the request that has the earliest deadline.

Earliest Deadline has an intuitive appeal.

Imagine standing in a long security line at the airport.

It would make sense to prioritize the processing of passengers who have the earliest flights assuming that there's enough time to process everyone before their flight leaves.

Processing 10 people whose flights leave in 30 minutes before someone whose flight leaves in 5 min will cause that last person to miss their flight.

But if that person is processed first, the other passengers may be slightly delayed but everyone will make their flight.

This is the sort of scheduling problem that Earliest Deadline and a weak priority system can solve.

It's outside the scope of our discussion, but it's interesting to think about what should happen if some flights are going to be missed.

If the system is overloaded, prioritizing by earliest deadline may mean that everyone will miss their flights!

In this scenario it might be better to assign priorities to the minimize the total number of missed flights.

This gets complicated in a hurry since the assignment of priorities now depends on exactly what requests are pending and how long it will take them to be serviced.

An intriguing problem to think about!