CS452/652 Real-Time Programming Course Notes

Daniel M. Berry, Cheriton School of Computer Science
University of Waterloo

Real-Time Considerations

Hardware interrupts are turned off in the kernel.

∴, the kernel will not be able to respond to any stimuli

∴, the kernel must limit the time it spends responding to a syscall, scheduling, and context switching in order to remain responsive to stimuli.

Real-Time Considerations, Cont'd

∴, the amount of time spent in the kernel responding to a syscall, scheduling, and context switching must be constant ($O(1)$) and *small*.

Why *must* hardware interrupts be turned off in the kernel?

Real-Time Considerations, Cont'd

In order to help ensure that the amount of time spent in the kernel responding to a syscall, scheduling, and context switching be constant ($O(1)$) and *small*.

So this is an if-and-only-if situation! ☺

Problem with Task Creation

Task creation

- requires copying task's DS to newly-allocated memory, and ∴
- requires $O(n)$ time, where n is the size of DS.

How can we fix this problem?

Solution to Task Creation Problem

Arrange for a task to copy its *own* DS.

Then the $O(n)$ copying time occurs outside of the kernel.

How?

Implementation of Solution

The provided `crt0.S` for tasks already does it.

The code needs to push:

- the task's DS,
- the kernel's DS,
- the location of data segment in physical memory,
- the location to which to copy the data segment, and
- the size of `bss`,

in addition to the state necessary in order to be able to switch into the task.

Task Management

- Task Descriptor
- Scheduling

Task Descriptor

A task descriptor is a data structure (i.e., the π in the diagrams) in which the kernel maintains information about a task.

struct taskDescriptor{
- process state (Active, Ready, Blocked, Dead),
- priority,
- SS, ESP,
- links, etc.

} descriptorTable[NUMDESCRIPTORS]

Task Descriptor, Cont'd

Create a descriptor for each new process as it is created.

Task identifier (TID):

- unique ID for each task
- primary purpose of a TID is to locate its task

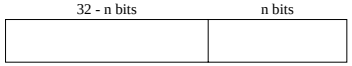
Task Descriptor, Cont'd

Simple implementations:

- A TID is an index into the descriptorTable.
- A TID is the address of its task's descriptor.

Task Descriptor, Cont'd

More sophisticated:



Use upper bits as a generation counter to prevent TID duplication if a task is given the same descriptor of a now-dead task.

Then tid → descriptor
is descriptorTable[tid & mask]
where mask is def'd as a string of *n* 1s.

Scheduling

In scheduling, the idea is that at any time, the most time-critical task should be active.

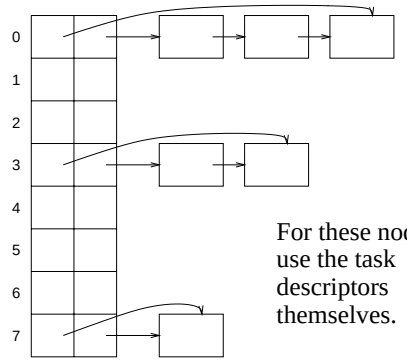
Your kernel must implement scheduling based on fixed priorities.

You must follow this scheme:

- All ready tasks with the highest priority run first.
- All ready task at any one priority are scheduled round robin.

Scheduling, Cont'd

A queue for each priority, 0=high through 8=low:



For these nodes, use the task descriptors themselves.

Scheduling, Cont'd

Reschedule(){
- put the current Active task at the end of the queue for its priority;
- select the task that is at the front of the highest priority queue;

}