Big Idea (Week 9): Sharing Hardware

The primary job of **Operating System** (OS) is to allow controlled sharing. The OS allows:

- multiple applications to share the same hardware platform
- applications to share common code
- concurrent applications to share the processor, memory, and peripheral devices

Ideally, it facilitates this sharing in a way that hides the details of the sharing from the applications.

The OS provides this controlled sharing by *virtualizing* the physical resources. That is, this layer of software gives each application or task the view that it owns the machine. This view hides the fact that other tasks may use the hardware at times, including times that occur between when the task starts and when it ends. The OS does this by identifying the state required by a task and saving it to memory when the task is not using the processor actively. That is, since the processor's job is to change the state, we can save the state to memory, suspend operation, allow the processor to perform other work for other tasks, then restore the processor's state and resume operation. As long as the restored state exactly matches the stored state, this can all be done without changing the results of the computation—the task cannot tell that the processor was "borrowed" by another task in the middle of its execution.

An important element of this controlled sharing is *isolation*. The OS guarantees that tasks do not interfere with each other (except as they are intended to interact). This means that tasks should not be able to read or change each other's memory or otherwise observe or modify the state of the hardware processor as seen by the other tasks. This also means that the failure of one application (perhaps because it contains bugs) should not cause a separate, non-dependent task to fail. The machine and OS should continue running along with the non-faulty tasks.

This sharing was originally important because machines were expensive and sharing provided a way for many users and applications to gain access to an expensive machine. As computers have become faster and cheaper, today this is important because a single task seldom consumes the entire machine. The task may only need to meet real-time guarantees (*e.g.* mp3 playback) or be limited to processing when a slow external source provides data (*e.g.* user typing at a keyboard, data downloaded from a network). Sharing allows you to have one small, inexpensive machine that handles a variety of tasks (*e.g.* Messaging, video or mp3 playback, web surfing) since none consume the entire capabilities of the machine. This sharing also allows applications to perform a variety of long latency operations (*e.g.* downloading pictures, saving a backup to disk) while continuing to be responsive to user inputs (*e.g.* scrolling a web page, continuing to edit the document, launching additional applications).