Virtualization

> Each peach-eater can eat their own *virtual* peach, even though there's only one *physical* peach.

The Base Abstraction: Processes

A *process* is a running program. As users, we often want to run multiple processes at the same time. To make this happen, the operating system must provide the illusion that there are many CPUs, so that each process can run on one.

This is done through *virtualization* – processes are stopped and started (*context switching*), sharing the CPU time at the cost of taking longer to run. Operating systems also have *policies* that determine exactly how decisions like scheduling are made.

Because a process is by definition a running program, we can understand what processes are by taking account of everything that happens during execution.

The *machine state* of a process is everything that the program can read of update while running – all the parts we're interested in when thinking about the process. Key components of the machine state are:

- *Memory*, which includes the instructions and all the data the program touches (its *address space*)
- *Registers*, part of the processor that are read from and written to over the course of program execution (including the *program counter*, *stack pointer*, and *frame pointer*)
- Various IO devices, like persistent storage

The Process API

In any implementation, the OS needs to provide a basic interface for interacting with processes, supporting operations:

- *Create*
- *Destroy*
- *Status*
- *Wait*, which can allow one process to wait for another to terminate
- Miscellaneous control, including support for *suspending* and *resuming* processes

Process Creation

Process creation involves taking bytes on disk that specify instructions and turning them into a running program. First, data is *loaded* into memory,

including all of its static data. On modern operating systems, this is done *lazily*. Next, memory is allocated for the *stack*, possibly initialized with values (eg. C's `argc` and `argv`). *Heap* data might also be allocated, and depending on the OS there might be additional initialization tasks (like opening file descriptors). Finally, execution begins from `main`, and control is transferred to the CPU to begin execution.

Process States

Processes have three high-level states:

- Running – executing instructions
- Ready – ready to run but not currently being executed
- Blocked – not ready to run until some other event takes place (for example, waiting for an IO request to resolve)

Transitions between ready and running (*scheduled* vs *descheduled*) happen at the discretion of the OS based on decisions of the *scheduler*.

Data Structures

A few key data structures are used to keep track of state in typical operating systems. For example, a *process list* tracks process state. This includes the *register context* (register contents) of each stopped process, process states, etc.