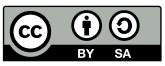
Operating systems



2019-10-21 09:44
e8fd88f

Evolution

Batch systems

- One task at a time
- Human operator would determine when to run each task
- Goal was to keep the machine busy
- (Machine much more expensive than the human labor to program and operate it.)

Time-sharing systems

- Automate the job of the human operator
- Software can schedule tasks for the system
- Called an “operating system”
- 1 computer shared by multiple (2-20) users
- Means that OS needs multitasking, security / protection
- Pretty sophisticated operating systems: Multics, CMS, **UNIX.**

Personal computers

- Smaller, cheaper – everyone could have their own.
- **But** had to reduce memory and other capabilities to make them cheap enough.
- Not initially capable of running time-sharing operating system
- But also didn’t need multitasking, security
- **MS-DOS** = (Microsoft) Disk Operating System
- Apple Mac = 1984

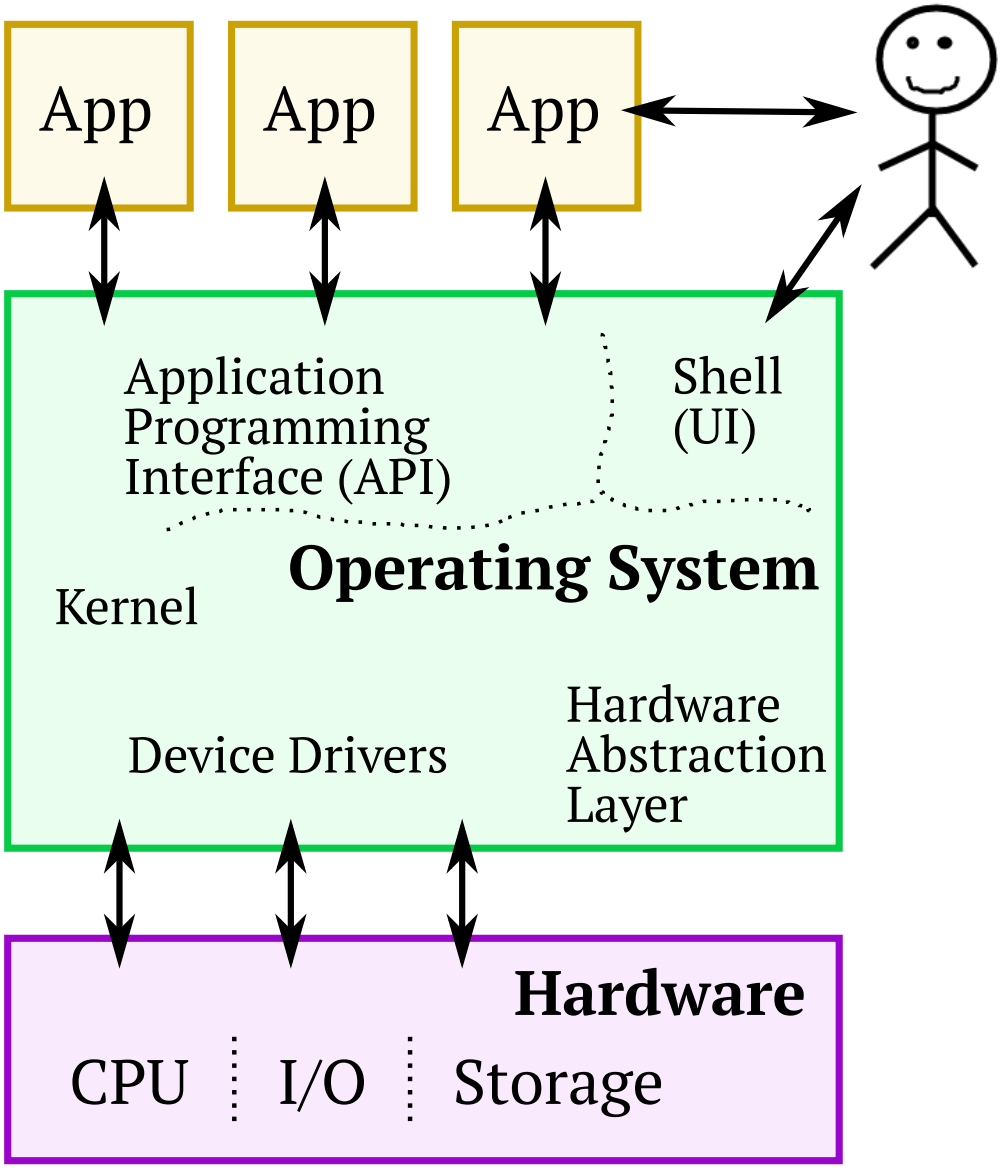
Networking

- To connect to networks, need more powerful machines, need multitasking, need security
- Developers reached back to time-sharing OS technology to integrate into new PC operating systems.

Structure and responsibilities

- OS manages and controls hardware devices
- OS provides a user interface (UI or Shell)
- OS provides an application programming interface (API)

- OS enforces responsible sharing of the hardware among various applications



CPU Scheduling

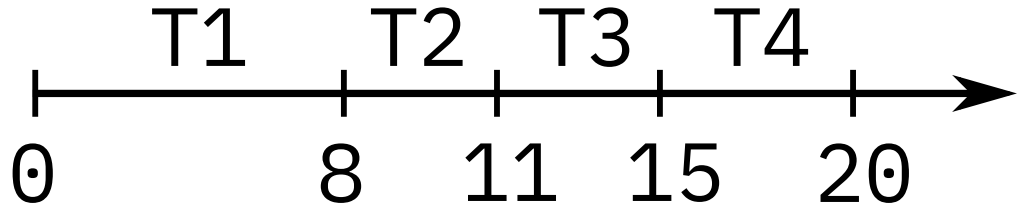
One responsibility of the OS is to decide **which task should run next** on the CPU. (Tasks are also known as applications, jobs, or processes.) In the following, we will assume a uniprocessor (just one task at a time) and that we run in *batch* mode — once started, a task runs until completion. Suppose we need to run these four tasks:

Task	Duration (sec)
T1	8
T2	3
T3	4
T4	5

There are a variety of scheduling *strategies* we can use for making the decision. We will consider two of them.

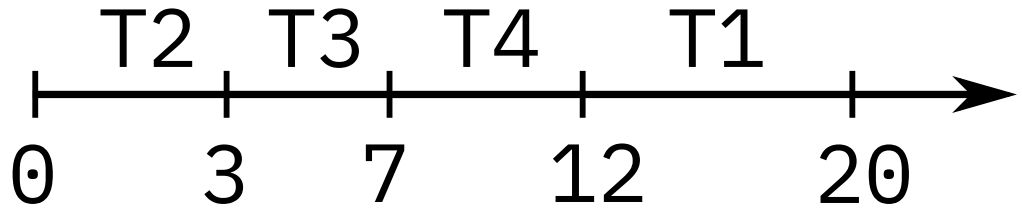
First-Come First-Served (FCFS)

In this strategy, we just take the tasks in order they *arrive* (or are provided to us). Arrange them on a timeline so we can see when each task begins and ends.



Shortest Job First (SJF)

Another strategy is to select tasks in order of their expected duration.



Quantify scheduling performance

Is there a reason to prefer one timeline over the other? Ultimately, we complete the same tasks in the same amount of time (20 seconds), so maybe it doesn't make much difference. But if we take the point of view of individual tasks, we can record their **turn-around times** and then take the average. Turn-around time measures the interval between when a task is **ready to run** and when it is **complete.** In this simplified setting, all of our tasks were ready to run from the start (time zero), so we'll just record completion times.

For FCFS, the calculation is:

$$\frac{8 + 11 + 15 + 20}{4} = 13.5$$

Whereas for SJF:

$$\frac{3 + 7 + 12 + 20}{4} = 10.5$$

A lower turn-around time is better, so SJF may be a preferable strategy.