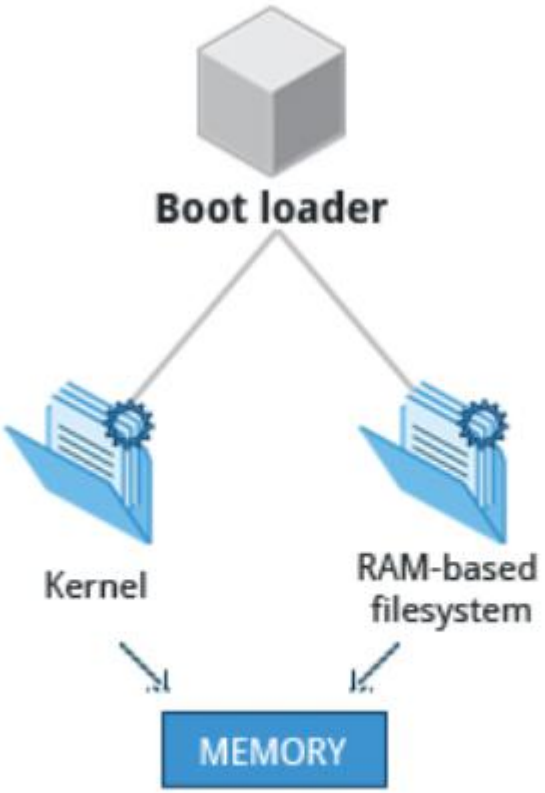
Kernel, init and Services

The Linux Kernel:

1. The boot loader loads both the kernel and an initial RAM–based file system (initramfs) into memory, so it can be used directly by the kernel.
2. When the kernel is loaded in RAM, it immediately initializes and configures the computer's memory and also configures all the hardware attached to the system.
3. This includes all processors, I/O subsystems, storage devices, etc.
4. The kernel also loads some necessary user space applications.

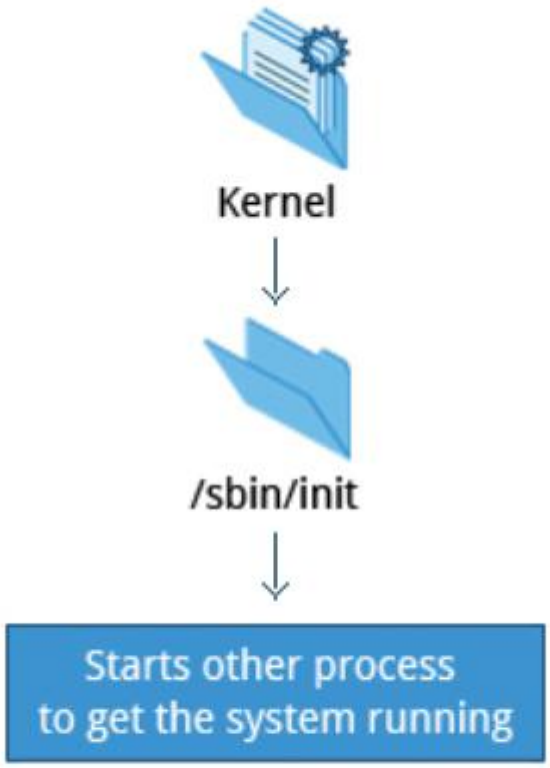


The Linux Kernel

/sbin/init and Services:

1. Once the kernel has set up all its hardware and mounted the root filesystem, the kernel runs /sbin/init.
2. This then becomes the initial process, which then starts other processes to get the system running.
3. Most other processes on the system trace their origin ultimately to init; exceptions include the so-called kernel processes.

4. These are started by the kernel directly, and their job is to manage internal operating system details.
5. Besides starting the system, init is responsible for keeping the system running and for shutting it down cleanly.
6. One of its responsibilities is to act when necessary as a manager for all non-kernel processes; it cleans up after them upon completion, and restarts user login services as needed when users log in and out, and does the same for other background system services.



/sbin/init and Services

7. Traditionally, this process startup was done using conventions that date back to the 1980s and the System V variety of UNIX.
8. This serial process had the system passing through a sequence of runlevels containing collections of scripts that start and stop services.
9. Each runlevel supported a different mode of running the system. Within each runlevel, individual services could be set to run, or to be shut down if running.
10. However, all major distributions have moved away from this sequential runlevel method of system initialization, although they usually emulate many System V utilities for compatibility purposes.

Startup Alternatives:

1. **SysVinit** viewed things as a serial process, divided into a series of sequential stages.
2. Each stage required completion before the next could proceed.
3. Thus, startup did not easily take advantage of the **parallel processing** that could be done on multiple processors or cores.
4. Furthermore, shutdown and reboot was seen as a relatively rare event; exactly how long it took was not considered important.
5. This is no longer true, especially with mobile devices and embedded Linux systems.
6. Some modern methods, such as the use of containers, can require almost instantaneous startup times.
7. Thus, systems now require methods with faster and enhanced capabilities.
8. Finally, the older methods required rather complicated startup scripts, which were difficult to keep universal across distribution versions, kernel versions, architectures, and types of systems.

   The two main alternatives developed were:

   Upstart:

   1. Developed by Ubuntu and first included in 2006
   2. Adopted in Fedora 9 (in 2008) and in RHEL 6 and its clones

   System:

   1. Adopted by Fedora first (in 2011).
   2. Adopted by RHEL 7 and SUSE
   3. Replaced Upstart in Ubuntu 16.04
9. While the migration to **systemd** was rather controversial, it has been adopted by all major distributions, and so we will not discuss the older System V(**sysvinit**) method or **Upstart**, which has become a dead end.

systemd Features:

1. Systems with systemd start up faster than those with earlier init methods.
2. This is largely because it replaces a serialized set of steps with aggressive parallelization techniques, which permits multiple services to be initiated simultaneously.
3. Complicated startup shell scripts are replaced with simpler configuration files, which enumerate what has to be done before a service is started, how to execute service startup, and what conditions the service should indicate have been accomplished when startup is finished.
4. One thing to note is that /sbin/init now just points to /lib/systemd/systemd; i.e. **systemd** takes over the **init** process.
5. One **systemd** command (**systemctl**) is used for most basic tasks.
6. Below is the bried listing of its use:
   - Starting, stopping, restarting a service (using httpd, the Apache web server, as an example) on a currently running system:
     $ sudo systemctl start|stop|restart httpd.service
   - Enabling or disabling a system service from starting up at system boot:
     $ sudo systemctl enable|disable httpd.service