



Sri
SAI RAM
ENGINEERING COLLEGE
INSTITUTE OF TECHNOLOGY
West Tambaram, Chennai - 44

Sairam
INSTITUTIONS



YEAR	SEM
II	IV

CS8392

OPERATING SYSTEMS
(Common to CSE, IT)

UNIT No. 1

1.1 Computer System Overview - Basic Elements

Version: 1.XX



Operating-System Generation

It is possible to design, code, and implement an operating system specifically for one machine at one site. More commonly, however, operating systems are designed to run on any of a class of machines at a variety of sites with a variety of peripheral configurations. The system must then be configured or generated for each specific computer site, a process sometimes known as system generation (**SYSGEN**).

The operating system is normally distributed on **disk** or **CD-ROM**. To generate a system, we use a special program. The **SYSGEN** program reads from a given file, or asks the operator of the system for information concerning the specific configuration of the hardware system, or probes the hardware directly to determine what components are there. The following kinds of information must be determined.

- **What CPU is to be used?** What options (extended instruction sets, floatingpoint arithmetic, and so on) are installed? For multiple CPU systems, each CPU must be described.
- **How much memory is available?** Some systems will determine this value themselves by referencing memory location after memory location until an "illegal address" fault is generated. This procedure defines the final legal address and hence the amount of available memory.
- **What devices are available?** The system will need to know how to address each device (the device number), the device interrupt number, the device's type and model, and any special device characteristics.
- **What operating-system options are desired, or what parameter values are to be used?** These options or values might include how many buffers of which sizes should be used, what type of CPU-scheduling algorithm is desired, what the maximum number of processes to be supported is, and so on. Once this information is determined, it can be used in several ways.

At one extreme, a system administrator can use it to modify a copy of the source code of the operating system. The operating system then is completely compiled. Data declarations, initializations, and constants, along with conditional compilation, produce an output object

version of the operating system that is tailored to the system described. At a slightly less tailored level, the system description can cause the creation of tables and the selection of modules from a precompiled library.

These modules are linked together to form the generated operating system. Selection allows the library to contain the device drivers for all supported I/O devices, but only those needed are linked into the operating system. Because the system is not recompiled, system generation is faster, but the resulting system may be overly general.

At the other extreme, it is possible to construct a system that is completely table driven. All the code is always part of the system, and selection occurs at execution time, rather than at compile or link time. System generation involves simply creating the appropriate tables to describe the system.

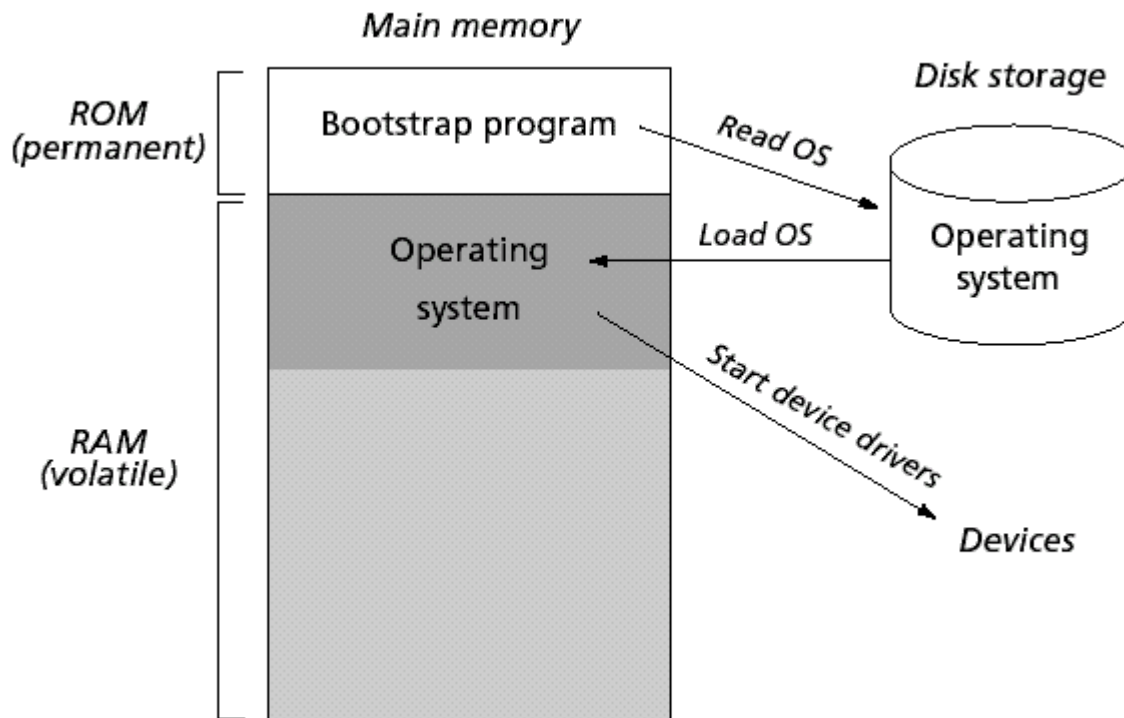
The major differences among these approaches are the size and generality of the generated system and the ease of modification as the hardware configuration changes. Consider the cost of modifying the system to support a newly acquired graphics terminal or another disk drive. Balanced against that cost, of course, is the frequency (or infrequency) of such changes.

System Boot

After an operating system is generated, it must be made available for use by the hardware. But how does the hardware know where the kernel is or how to load that kernel? The procedure of starting a computer by loading the kernel is known as booting the system. On most computer systems, a small piece of code known as the bootstrap program or bootstrap loader locates the kernel, loads it into main memory, and starts its execution. Some computer systems, such as PCs, use a two-step process in which a simple bootstrap loader fetches a more complex boot program from disk, which in turn loads the kernel.

When a CPU receives a reset event—for instance, when it is powered up or rebooted—the instruction register is loaded with a predefined memory location, and execution starts there. At that location is the initial bootstrap program. This program is in the form of read-only

memory (ROM), because the RAM is in an unknown state at system startup. ROM is convenient because it needs no initialization and cannot be infected by a computer virus.



The bootstrap program can perform a variety of tasks. Usually, one task is to run diagnostics to determine the state of the machine. If the diagnostics pass, the program can continue with the booting steps. It can also initialize all aspects of the system, from CPU registers to device controllers and the contents of main memory. Sooner or later, it starts the operating system.

Some systems—such as cellular phones, PDAs, and game consoles—store the entire operating system in ROM. Storing the operating system in ROM is suitable for small operating systems, simple supporting hardware, and rugged operation. A problem with this approach is that changing the bootstrap code requires changing the ROM hardware chips. Some systems resolve this problem by using erasable programmable read-only memory (EPROM), which is read only except when explicitly given a command to become writable. All forms of ROM are also known as firmware, since their characteristics fall somewhere between those of hardware and those of software. A problem with firmware in general is that executing code there is slower than executing code in RAM. Some systems store the

operating system in firmware and copy it to RAM for fast execution. A final issue with firmware is that it is relatively expensive, so usually only small amounts are available.

For large operating systems (including most general-purpose operating systems like Windows, Mac OS X, and UNIX) or for systems that change frequently, the bootstrap loader is stored in firmware, and the operating system is on disk. In this case, the bootstrap runs diagnostics and has a bit of code that can read a single block at a fixed location (say block zero) from disk into memory and execute the code from that boot block. The program stored in the boot block may be sophisticated enough to load the entire operating system into memory and begin its execution. More typically, it is simple code (as it fits in a single disk block) and only knows the address on disk and length of the remainder of the bootstrap program. All of the disk-bound bootstrap, and the operating system itself, can be easily changed by writing new versions to disk. A disk that has a boot partition is called a boot disk or system disk.

Now that the full bootstrap program has been loaded, it can traverse the file system to find the operating system kernel, load it into memory, and start its execution. It is only at this point that the system is said to be running.