The lifecycle of a computer involves 5 steps.

Booting, Kernel Initialization, Device Management Initialization, full operation and shutdown.

For booting the operating system has a loader which is a program that moves bits from disk to memory and then transfers CPU control to those newly loaded bits. Then there is the bootloader which is a program that loads the “first program” (kernel)

The bootloader is kept in the Boot PROM, which is persistent code that is already loaded on power up. The boot manager is a program that lets you choses the first program to load.

The bootloader is the program that lets you load the first program. It is usually staged in a primary and secondary stage. It requires firmware support (“a hardware bootstrap”) kept in firmware.

BIOS differs from a PROM monitor in that the BIOS is limited to be accesed only during the booting of the system. While the monitor can be continuously accesed by the command interpreter even if the system has already been loaded.

Boot Manager allow the user to pick the operating system he wants to run, and they basically copy themselves the (MBR) Master boot record which specifies the program that will run first during the startup of the operating system.

When booting a PC the first step is the POST, then a interrupt 19h calls the bootstrap loader that allows you to select the device you want to boot from, it will then load the boot sector of the device you pick which will allow you to load the bootloader of the operating system. The boot loader is kept in the MBR ofyour harddisk. After the boot sector is loaded the operating system performs a check with a magic number that determines whether it is executable or not, and then allows the PC to execute the boot sector or (primary bootloader). A way to accelerate this process is to have the kernel compressed, in which then will consist of two pieces a head and a compressed tail, the head will be the code that will allow the tail to be decompressed.

Setup.S performs real-mode hardware initialization, and lets you load more information of the kernel and execute as it is retrieved from harddisk.

What are these?

More user/group id types increase

flexibility – also increases complexity.

􀂄 uid, gid - plain user/group id

􀂄 euid, egid - effective user/group id

􀂄 suid, sgid - set user/group id

􀂄 fsuid, fsgid - filesystem user/group

id

Relations between processes are recorded via references that include the original parent, the parent, the youngest child, the oldest and youngest siblings.

w/ this refs u can traverse all levels of related tasks.

The task structure is one of the data structures that are kept in the operating system. It includes information about the state of the process, the flags and signals of the process.

Also information such as the address\_limit is kept which describes the address space that is possible to access. Exec\_domain is also kept as it provides a description of another platform that could be being emulated to execute the process.

There’s a lock for synchronization for the task structure.

Priority may also be kept for scheduling algorithms that are based on the priority of the task. Additionally, for scheduling , a counter can be kept to count the number of clock cycles the process can run before scheduling it again. Also information of policy is kept to deal with multi-level queueing which allows multiple scheduling algorithms to be used in the operating system.

Information for memory management is also kept in the task structure, the struct for associated to this is the mem\_manage (struct \*) which describes the positions of all parts of the process in memory. This struct includes start\_code, end\_code, start\_data, and end\_data positions of the program. It also has a start\_stack for the stack ,and a start\_mmap for the heap, and an arg\_start, arg\_end, and env\_start and env\_end.

Information about files is also kept because the process can use them as resources. A files\_info pointer to the files informs structure is kept that contains a count of the number of processes that are reading/writing from a given file.

The task structure also keeps track of the per\_cpu\_utime and per\_cpu\_stime, which is the time that a process has taken to execute in user mode and kernel mode respectively.

Finally semaphore information is also kept, a semsleep reference is kept in order to determine what resources the process is sleeping on. A semhold reference is kept in order to determine what semaphores neeed to be realeased after the process finishes using the resource.

The Process table is the data structure that keeps tasks within the system organized in a doubly-linked list. That way, the tasks are moved between different lists that have different meanings/states. The tasks can be accessed via the next\_task and prev\_task in the task structure. A reference init\_task is kept in order to access the first task in the list. One can also access the current task using the get\_current macro, and the current task can be in any state.

The maximum number of tasks in the system is restricted to a global value: max\_threads. This helps maintain some aspects of security in the system.

For files two data structures are kept, on is the iNode data structure and the other is the file data structure. The inode data structure contains the view of the “system” on the file. This means there is exactly one inode per file used. The information kept in this structure is primarily the physical location, meaning which blocks are comprised in the file, and what type of file it is.