**BIOS**

One of the most common uses of [Flash memory](http://computer.howstuffworks.com/flash-memory.htm) is for the **basic input/output system** of your computer, commonly known as the BIOS (pronounced "bye-ose"). On virtually every computer available, the BIOS makes sure all the other chips, hard drives, ports and CPU function together.

Every [desktop](http://computer.howstuffworks.com/pc.htm) and [laptop](http://computer.howstuffworks.com/laptop.htm) computer in common use today contains a [microprocessor](http://computer.howstuffworks.com/microprocessor.htm) as its central processing unit. The microprocessor is the **hardware** component. To get its work done, the microprocessor executes a set of instructions known as **software** (see [How Microprocessors Work](http://computer.howstuffworks.com/microprocessor.htm) for details). You are probably very familiar with two different types of software:

* The **operating system** - The [operating system](http://computer.howstuffworks.com/operating-system.htm) provides a set of services for the applications running on your computer, and it also provides the fundamental user interface for your computer. Windows 98 and Linux are examples of operating systems. (See [How Operating Systems Work](http://computer.howstuffworks.com/operating-system.htm) for lots of details.)
* The **applications** - Applications are pieces of software that are programmed to perform specific tasks. On your computer right now you probably have a browser application, a word processing application, an [e-mail](http://computer.howstuffworks.com/e-mail-messaging/email.htm) application and so on. You can also buy new applications and install them.

­It turns out that the BIOS is the third type of software your computer needs to operate successfully. In this article, you'll learn all about BIOS -- what it does, how to configure it and what to do if your BIOS needs updating.

**Processor:-** The computer you are using to read this page uses a **microprocessor** to do its work. The microprocessor is the heart of any normal computer, whether it is a [desktop machine](http://computer.howstuffworks.com/pc.htm), a [server](http://computer.howstuffworks.com/web-server.htm) or a [laptop](http://computer.howstuffworks.com/laptop.htm). The microprocessor you are using might be a Pentium, a K6, a PowerPC, a Sparc or any of the many other brands and types of microprocessors, but they all do approximately the same thing in approximately the same way.

**A microprocessor** -- also known as a **CPU** or central processing unit -- is a complete computation engine that is fabricated on a single chip. The first microprocessor was the Intel 4004, introduced in 1971. The 4004 was not very powerful -- all it could do was add and subtract, and it could only do that 4 [bits](http://computer.howstuffworks.com/bytes.htm) at a time. But it was amazing that everything was on one chip. Prior to the 4004, engineers built computers either from collections of chips or from discrete components ([transistors](http://electronics.howstuffworks.com/diode2.htm) wired one at a time). The 4004 powered one of the first portable electronic calculators

**PCB**

**Process Control Block** (PCB, also called **Task Controlling Block**, **Task Struct**, or **Switchframe**) is a data structure in the [operating system](http://en.wikipedia.org/wiki/Operating_system) [kernel](http://en.wikipedia.org/wiki/Kernel_%28computer_science%29) containing the information needed to manage a particular process. The PCB is "the manifestation of a process in an operating system".

If the mission of the operating system is to manage computing resources on behalf of processes, then it must be continuously informed about the status of each process and resource.

The approach commonly followed to represent this information is to create and update status tables for each relevant entity, like memory, I/O devices, files and processes.

Memory tables, for example, may contain information about the allocation of main and secondary (virtual) memory for each process, authorization attributes for accessing memory areas shared among different processes, etc. I/O tables may have entries stating the availability of a device or its assignment to a process, the status of I/O operations being executed, the location of memory buffers used for them, etc.

File tables provide info about location and status of files (of course, what else? more on this later). Finally, process tables store the data the OS needs to manage processes. At least part of the process control data structure is always maintained in main memory, though its exact location and configuration varies with the OS and the memory management technique it uses. In the following we'll refer by process image to the complete physical manifestation of a process, which includes instructions, program data areas (both static and dynamic - e.g. at least a stack for procedure calls and parameter passing) and the process management information. We'll call this last set the process control block (PCB).

The role of the PCBs is central in process management: they are accessed and/or modified by most OS utilities, including those involved with scheduling, memory and I/O resource access and performance monitoring. It can be said that the set of the PCBs defines the current state of the operating system. Data structuring for processes is often done in terms of PCBs. For example, pointers to other PCBs inside a PCB allow the creation of those queues of processes in various scheduling states ("ready", "blocked", etc.) that we previously mentioned.

In modern sophisticated multitasking systems the PCB stores many different items of data, all needed for correct and efficient process management. Though the details of these structures are obviously system-dependent, we can identify some very common parts, and classify them in three main categories:

* Process identification data
* Processor state data
* Process control data

*Process identification data* always include a unique identifier for the process (almost invariably an integer number) and, in a multiuser-multitasking system, data like the identifier of the parent process, user identifier, user group identifier, etc. The process id is particularly relevant, since it's often used to cross-reference the OS tables defined above, e.g. allowing to identify which process is using which I/O devices, or memory areas.

*Processor state data* are those pieces of information that define the status of a process when it's suspended, allowing the OS to restart it later and still execute correctly. This always include the content of the CPU general-purpose registers, the CPU process status word, stack and frame pointers etc.

*Process control information* is used by the OS to manage the process itself. This includes:

The process scheduling state (different from the task state above discussed), e.g. in terms of "ready", "suspended", etc., and other scheduling information as well, like a priority value, the amount of time elapsed since the process gained control of the CPU or since it was suspended. Also, in case of a suspended process, event identification data must be recorded for the event the process is waiting for. Process structuring information:process's children id's, or the id's of other processes related to the current one in some functional way, which may be represented as a queue, a ring or other data structures. Interprocess communication information: various flags, signals and messages associated with the communication among independent processes may be stored in the PCB. Process privileges, in terms of allowed/unallowed access to system resources.

**Comparison between Scheduler**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.N.** | **Long Term Scheduler** | **Short Term Scheduler** | **Medium Term Scheduler** |
| 1 | It is a job scheduler | It is a CPU scheduler | It is a process swapping scheduler. |
| 2 | Speed is lesser than short term scheduler | Speed is fastest among other two | Speed is in between both short and long term scheduler. |
| 3 | It controls the degree of multiprogramming | It provides lesser control over degree of multiprogramming | It reduces the degree of multiprogramming. |
| 4 | It is almost absent or minimal in time sharing system | It is also minimal in time sharing system | It is a part of Time sharing systems. |
| 5 | It selects processes from pool and loads them into memory for execution | It selects those processes which are ready to execute | It can re-introduce the process into memory and execution can be continued. |

**Context Switch**

A context switch is the mechanism to store and restore the state or context of a CPU in Process Control block so that a process execution can be resumed from the same point at a later time. Using this technique a context switcher enables multiple processes to share a single CPU. Context switching is an essential part of a multitasking operating system features.

When the scheduler switches the CPU from executing one process to execute another, the context switcher saves the content of all processor registers for the process being removed from the CPU, in its process descriptor. The context of a process is represented in the process control block of a process.

Context switch time is pure overhead. Context switching can significantly affect performance as modern computers have a lot of general and status registers to be saved. Content switching times are highly dependent on hardware support. Context switch requires ( n + m ) bxK time units to save the state of the processor with n general registers, assuming b are the store operations are required to save n and m registers of two process control blocks and each store instruction requires K time units.

Some hardware systems employ two or more sets of processor registers to reduce the amount of context switching time. When the process is switched, the following information is stored.

* Program Counter
* Scheduling Information
* Base and limit register value
* Currently used register
* Changed State
* I/O State
* Accounting

**SCHEDULING SELECTION FUNCTION**

When all processes are of the same priority, or processes at a given priority level must be decided amongst, the selection function determines which process to execute next.

The selection function typically makes its decision based on resource requirements, or on the past execution profile of all processes.

In addition, the decision mode decides the moments in time at which the decision function is invoked:

**non-pre-emptive**:

once a process is executing, it will continue to execute until

• it terminates, or

• it makes an I/O request which would block the process, or

• it makes an operating system call.

**pre-emptive:**

the same three conditions as above apply, and in addition the

process may be pre-empted by the operating system when

•

a new process arrives (perhaps at a higher priority), or

•

an interrupt or signal occurs, or

•

a (frequent) clock interrupt occurs.