Context switch

From Wikipedia, the free encyclopedia

A **context switch** is the computing process of storing and restoring the state (context) of a CPU such that multiple processes can share a single CPU resource. The context switch is an essential feature of a multitasking operating system. Context switches are usually computationally intensive and much of the design of operating systems is to optimize the use of context switches. A context switch can mean a register context switch, a task context switch, a thread context switch, or a process context switch. What constitutes the context is determined by the processor and the operating system. Switching from one process to another requires a certain amount of time for doing the administration - saving and loading registers and memory maps, updating various tables and list etc.

Contents

When to switch?

There are three scenarios where a context switch needs to occur. They are:

Multitasking

Most commonly, within some scheduling scheme, one process needs to be switched out of the CPU so another process can run. Within a preemptive multitasking operating system, the scheduler allows every task to run for some certain amount of time, called its *time slice*.

If a process does not voluntarily yield the CPU (for example, by performing an I/O operation), a timer interrupt fires, and the operating system schedules another process for execution instead. This ensures that the CPU cannot be monopolized by any one processor-intensive application.

Interrupt handling

Some architectures (like the Intel x86 architecture) are interrupt driven. This means that if the CPU requests data from a disk, for example, it does not need to busy-wait until the read is over, it can issue the request and continue with some other execution; when the read is over, the CPU can be *interrupted* and presented with the read. For interrupts, a program called an *interrupt handler* is installed, and it is the interrupt handler that handles the interrupt from the disk.

The kernel services the interrupts in the context of the interrupted process even though it may not have caused the interrupt. The interrupted process may have been executing in user mode or in kernel mode. The kernel saves enough information so that it can later resume execution of the interrupted process and services the interrupt in kernel mode. The kernel does not spawn or schedule a special process to handle interrupts.

User and kernel mode switching

When a transition between user mode and kernel mode is required in an operating system, a context switch is not necessary; a mode transition is *not* by itself a context switch. However, depending on the operating system, a context switch may also take place at this time.

Context switch: steps

In a context switch, the state of the first process must be saved somehow, so that, when the scheduler gets back to the execution of the first process, it can restore this state and continue.

The state of the process includes all the registers that the process may be using, especially the program counter, plus any other operating system specific data that may be necessary. This data is usually stored in a data structure called a process control block (PCB), or switchframe.

Now, in order to switch processes, the PCB for the first process must be created and saved. The PCBs are sometimes stored upon a per-process stack in kernel memory (as opposed to the user-mode stack), or there may be some specific operating system defined data structure for this information.

Since the operating system has effectively suspended the execution of the first process, it can now load the PCB and context of the second process. In doing so, the program counter from the PCB is loaded, and thus execution can continue in the new process. New processes are chosen from a queue or queues. Process and thread priority can influence which process continues execution, with processes of the highest priority checked first for ready threads to execute.

Software vs hardware context switching

Context switching can be performed primarily by software or hardware. Some processors, like the Intel 80286 and higher CPUs, have hardware support for context switches, by making use of a special data segment designated the Task State Segment or TSS. When a task switch occurs (implicitly due to a CALL instruction, referring to a task gate, or explicitly due to an interrupt or exception) the CPU can automatically load the new state from the TSS. With other tasks performed in hardware, one would expect this to be rather fast; however, mainstream operating systems, including Windows, do not use this feature. This is due mainly to two reasons: that hardware context switching does not save all the registers (only general purpose registers, not floating point registers--although the TS bit is automatically turned on in the CR0 control register, resulting in a fault when executing floating point instructions and giving the OS the opportunity to do saving and restoring of the floating point state), and associated performance issues.

Some architectures contain logic to allow several hardware contexts to exist simultaneously, eliminating the need to store and restore the CPU context to memory on context switch. The extreme case is the barrel processor architecture, which switches between threads of execution on every cycle.

External links

- Context Switch Definition (http://www.linfo.org/context_switch.html) - by The Linux Information Project (LINFO)
- Context Switches (http://msdn.microsoft.com/en-us/library/ms682105(VS.85).aspx) - from the Microsoft Developer Network (MSDN)

Retrieved from "http://en.wikipedia.org/wiki/Context_switch"
Categories: Process (computing)