sleep() vs blocking of a process

I get confused with the sleep(), blocking call, preemption concept very often. As I understand preemption is completely done by the scheduler irrespective of what the process was doing. Except when the process is in some critical section or executing an atomic instruction, the scheduler can preempt the process based on the scheduling algorithm and put it in the waiting process list.

On the other hand sleep() calls the scheduler to block it for the specified time interval; passed as the argument to the sleep().

Blocking call is like waiting for an operation to complete like I/O operation such as disk read/write, signal from another device, etc.

Can someone give me explain me the working of these in a more comprehensive way or point me out to some solid resources? Thanks.

operating-system linux-kernel

asked Sep 21 '12 at 1:14

Nike
140 2 13

Everything you said was right. What else would you like to know? – qdii Sep 21 '12 at 1:19

When the process does a blocking call, it waits for the operation to complete. So when this operation is complete, does the blocked process immediately start running or scheduler takes control of it after it is blocked and schedules as per its wish? – Nike Sep 21 '12 at 1:50

2 Answers

> When the process does a blocking call, it waits for the operation to complete.

I am not entirely sure you understood this correctly. Say your program issues an I/O command, for instance `read()`. Your process does not wait for the operation to complete: it does not hold the processor until the I/O operation is done. When `read()` is called, the control of the processor is yield to the OS, which will prompt I/O operations. Those operations are extremely slow in comparison to the CPU speed, and they are executed by dedicated hardware, which means that the CPU will be free to execute something else in the meantime until the hardware signals that it is done (by a hardware interruption, for instance).

From the point of view of the program that run `read()`, all it knows is that `read()` has been called: the register that pointed to the current instruction has not changed, its virtual memory is the same as before. The process is "blocked". It does not mean that the CPU is not running.

From the point of view of the OS, the program is in waiting mode until the hardware signals that it has carried out its task. In the meantime, the scheduler has waken up another process, restored its context (i.e. the value of the CPU registers have been set to what they were just before it felt asleep, etc.), and started executing its code.

When the hardware is done, an interruption happens and the OS acknowledges it by marking the process as executable. Depending on the scheduler policy, it can restore the process context and start executing it right away, or wait until the current process has finished its time slice before switching context.

To read more about the implementation of the linux scheduler: Understanding the linux kernel is a very good book.

edited Sep 21 '12 at 2:32

answered Sep 21 '12 at 2:24

qdii
5,340 5 32 78

I get a better picture now.. Thanks. – Nike Sep 21 '12 at 2:33

Your view is absolutely correct.
http://en.wikipedia.org/wiki/Preemption_(computing)

answered Sep 21 '12 at 1:19

Serge
5,231 8 24