

Re: Polling, Interrupts, DMA, Synchronous, Asynchronous I/O Definitions

Source: <http://coding.derkeiler.com/Archive/Assembler/alt.lang.asm/2004-07/0920.html>

From: Beth (*BethStone21_at_hotmail.NOSPICEDHAM.com*)

Date: 07/22/04

Date: Thu, 22 Jul 2004 11:21:25 GMT

Randy wrote:

> *Frank asked:*

> > *In any case, this discussion prompts a philosophical question for the*

> > *LuxAsm team... If it were possible to speed up LuxAsm by being*

> > *"inconsiderate" and hogging resources, would this be an acceptable*

> > *method of speeding up LuxAsm? (I suppose the answer depends on "how*

> > *much?"...)*

>

> *Of course not. That's for the user to do. If they want Luxasm to run*

> *faster, they need to bump the priority up, that's all.*

Bumping up the priority of a task does not necessarily make it run any faster...

Under certain circumstances, you may actually create problems doing this, such as "starvation" (high priority threads failing to yield so that lower priority threads run rarely, if at all)...which could even lead to "priority inversion"...where the higher priority thread is locking out a lower priority thread BUT that it actually depends on that lower priority thread to do work on its behalf...creating a logical "catch-22" situation, not greatly dissimilar to that of "deadlock": The lower priority thread is waiting for the higher priority thread to relinquish control or it won't get any CPU time, the higher priority thread is waiting for the lower priority thread to respond and we're stuck in a logical paradox of requirements from the threads...this is another potential "lock up" situation and, as it's a potential issue in many instances, schedulers often employ particular methods to avoid it (similar to the idea of avoiding "deadlock" by such methods as assigning priorities to resources and then refusing to grant access unless the resources

are requested in numerical order...or, of course, by quickly "testing" for deadlock conditions – granting it "theoretically" in a resource matrix and then probing for the possibility of the condition arising – before any resource allocations are actually physically granted and then refusing if a deadlock condition is detected)...

Even when "bumped up" with causing any of these ill-effects, then, clearly, increasing the thread's priority does not somehow make the CPU execute faster...nor does it make the disks spin any faster...nor, if the application calls such a function, does it prevent it waiting and blocking on such things...nor does it inject the user with amphetamines and make them suddenly rush around with the mouse and keyboard any faster, when input is needed...

Altering the "priority", also in full accordance to the Plain English semantics of the word, simply refers to the relative importance of threads, where higher priority threads are dealt with first before lower priority threads...

It's like having a "V.I.P. list" for an otherwise public nightclub...the bouncer at the door makes everyone queue to get in from the public...but should one of the "V.I.P." guests arrive then they are given "priority" and allowed to by-pass the queue, walking straight in (if many V.I.P.s turn up at the same time, of relatively equal priority, then they form their own small queue but the V.I.P. queue is always dealt with first before the queue for the ordinary public is allowed back in again...a V.I.P. does NOT stand around queuing waiting for members of the general public to go first (they may have to wait for other V.I.P.s before them to go through first but NOT any ordinary member of the public in the other queue ;)...

"Priority" is only about "who goes first?", not about execution speed directly...any "speed up" that might happen from raising the thread priority is actually more a result of NOT allowing anything else to run, not a case of speeding the thread up in any way (the speed of the code depends on the instructions used and the CPU speed, nothing else can change that externally: You "speed up" code by writing it more optimally, priorities are only a "bias" to the OS scheduler to always select your task before any other lower-priority tasks)...

In certain circumstances, raising the priority could actually be detrimental to the application's performance...for instance, all the V.I.P.s arrive at the club but the V.I.P. at the front of the V.I.P. queue says "No, I'm not going in until my friend – near the end of the queue of non-V.I.P.s members of the general public – comes with me" (that is, the higher priority thread is waiting and depending on a lower priority thread to do some

work)...then we have "priority inversion" and it causes pandemonium to the system: The V.I.P.s can't carry on until the V.I.P. at the front goes in...the V.I.P. at the front won't go in until their "lower priority" friend comes in with them...the "lower priority" queue isn't allowed to go in until all the V.I.P.s have been dealt with first...oops, looks like no-one going anywhere, unless the "friend" is temporarily given a higher priority (e.g. the bouncer saying "Okay, let the V.I.P's friend through") or the V.I.P. at the front of the V.I.P. queue is temporarily lowered in priority (e.g. the bouncer saying: "Could you please stand aside, then, while you wait for your friend so that everyone else can be dealt with?")...

[Many schedulers, in fact, deliberately include means and measures designed to make such "priority inversion" be avoided...such as having threads "inherit" priority from the calling / waiting task (e.g. the "friend" is temporarily granted the same priority as the V.I.P. waiting for them)...or to add in a touch of randomness into the selection...or by having the priority "decay" as time goes forward (after all, a higher priority thread should be selected first...therefore, if it's having to wait a long time, this suggests there's some kind of "priority inversion" problem)...Windows uses some kind of "temporary priority boost" scheme in this capacity...]

Well, to end on a song:

"Polling, polling, polling...keep those apps polling" ;)

Beth :)