NAME

critical_enter, **critical_exit** - enter and exit a critical region

SYNOPSIS

#include <sys/param.h>
#include <sys/systm.h>

void
critical_enter(*void*);

void
critical_exit(*void*);

CRITICAL_ASSERT(*struct thread *td*);

DESCRIPTION

These functions are used to prevent preemption in a critical region of code. All that is guaranteed is that the thread currently executing on a CPU will not be preempted. Specifically, a thread in a critical region will not migrate to another CPU while it is in a critical region, nor will the current CPU switch to a different thread. The current CPU may still trigger faults and exceptions during a critical section; however, these faults are usually fatal.

The CPU might also receive and handle interrupts within a critical section. When this occurs the interrupt exit will not result in a context switch, and execution will continue in the critical section. Thus, the net effect of a critical section on the current thread's execution is similar to running with interrupts disabled, except that timer interrupts and filtered interrupt handlers do not incur a latency penalty.

The **critical_enter**() and **critical_exit**() functions manage a per-thread counter to handle nested critical sections. If a thread is made runnable that would normally preempt the current thread while the current thread is in a critical section, then the preemption will be deferred until the current thread exits the outermost critical section.

Note that these functions do not provide any inter-CPU synchronization, data protection, or memory ordering guarantees, and thus should *not* be used to protect shared data structures.

These functions should be used with care as an unbound or infinite loop within a critical region will deadlock the CPU. Also, they should not be interlocked with operations on mutexes, sx locks, semaphores, or other synchronization primitives, as these primitives may require a context switch to operate. One exception to this is that spin mutexes include a critical section, so in certain cases critical sections may be interlocked with spin mutexes.

Critical regions should be only as wide as necessary. That is, code which does not require the critical section to operate correctly should be excluded from its bounds whenever possible. Abuse of critical sections has an effect on overall system latency and timer precision, since disabling preemption will delay the execution of threaded interrupt handlers and callout(9) events on the current CPU.

The **CRITICAL_ASSERT**() macro verifies that the provided thread *td* is currently executing in a critical section. It is a wrapper around KASSERT(9).

SEE ALSO

callout(9), KASSERT(9), locking(9)

HISTORY

These functions were introduced in FreeBSD 5.0.