NAME

mi_switch, **cpu_switch**, **cpu_throw** - switch to another thread context

SYNOPSIS

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

void
mi_switch(*int flags*);

void
cpu_switch(*struct thread *oldtd, struct thread *newtd, struct mtx *lock*);

void
cpu_throw(*struct thread *oldtd, struct thread *newtd*);

DESCRIPTION

The **mi_switch**() function implements the machine-independent prelude to a thread context switch. It is the single entry point for every context switch and is called from only a few distinguished places in the kernel. The context switch is, by necessity, always performed by the switched thread, even when the switch is initiated from elsewhere; e.g. preemption requested via Inter-Processor Interrupt (IPI).

The various major uses of **mi_switch**() can be enumerated as follows:

1. From within a function such as sleepq_wait(9) or **turnstile_wait**() when the current thread voluntarily relinquishes the CPU to wait for some resource or lock to become available.

2. Involuntary preemption due to arrival of a higher-priority thread.

3. At the tail end of critical_exit(9), if preemption was deferred due to the critical section.

4. Within the TDA_SCHED AST handler, when rescheduling before the return to usermode was requested. There are several reasons for this, a notable one coming from **sched_clock**() when the running thread has exceeded its time slice.

5. In the signal handling code (see issignal(9)) if a signal is delivered that causes a process to stop.

6. In **thread_suspend_check**() where a thread needs to stop execution due to the suspension state of the process as a whole.

7. In kern_yield(9) when a thread wants to voluntarily relinquish the processor.

The *flags* argument to **mi_switch**() indicates the context switch type. One of the following must be passed:

SWT_OWEPREEMPT	Switch due to delayed preemption after exiting a critical section.
SWT_TURNSTILE	Switch after propagating scheduling priority to the owner of a resource.
SWT_SLEEPQ	Begin waiting on a sleepqueue(9).
SWT_RELINQUISH	Yield call.
SWT_NEEDRESCHED	Rescheduling was requested.
SWT_IDLE	Switch from the idle thread.
SWT_IWAIT	A kernel thread which handles interrupts has finished work and must wait for interrupts to schedule additional work.
SWT_SUSPEND	Thread suspended.
SWT_REMOTEPREEMPT	Preemption by a higher-priority thread, initiated by a remote processor.
SWT_REMOTEWAKEIDLE	Idle thread preempted, initiated by a remote processor.
SWT_BIND	The running thread has been bound to another processor and must be switched out.

In addition to the switch type, callers must specify the nature of the switch by performing a bitwise OR with one of the SW_VOL or SW_INVOL flags, but not both. Respectively, these flags denote whether the context switch is voluntary or involuntary on the part of the current thread. For an involuntary context switch in which the running thread is being preempted, the caller should also pass the SW_PREEMPT flag.

Upon entry to **mi_switch**(), the current thread must be holding its assigned thread lock. It may be unlocked as part of the context switch. After they have been rescheduled and execution resumes, threads will exit **mi_switch**() with their thread lock unlocked.

mi_switch() records the amount of time the current thread has been running before handing control over to the scheduler, via **sched_switch**(). After selecting a new thread to run, the scheduler will call **cpu_switch**() to perform the low-level context switch.

cpu_switch() is the machine-dependent function that performs the actual switch from the running thread *oldtd* to the chosen thread *newtd*. First, it saves the context of *oldtd* to its Process Control Block (PCB, *struct pcb*), pointed at by *oldtd->td_pcb*. The function then updates important per-CPU state such as the curthread variable, and activates *newtd's* virtual address space using its associated pmap(9) structure. Finally, it reads in the saved context from *newtd's* PCB. CPU instruction flow continues in the new thread context, on *newtd's* kernel stack. The return from **cpu_switch**() can be understood as a completion of the function call initiated by *newtd* when it was previously switched out, at some point in the distant (relative to CPU time) past.

The *mtx* argument to **cpu_switch**() is used to pass the mutex which will be stored as *oldtd's* thread lock at the moment that *oldtd* is completely switched out. This is an implementation detail of **sched_switch**().

cpu_throw() is similar to **cpu_switch**() except that it does not save the context of the old thread. This function is useful when the kernel does not have an old thread context to save, such as when CPUs other than the boot CPU perform their first task switch, or when the kernel does not care about the state of the old thread, such as in thread_exit(9) when the kernel terminates the current thread and switches into a new thread, *newtd*. The *oldtd* argument is unused.

SEE ALSO

critical_exit(9), issignal(9), kern_yield(9), mutex(9), pmap(9), sleepqueue(9), thread_exit(9)