NAME

kern_yield, **maybe_yield**, **should_yield** - functions for yielding execution of the current thread

SYNOPSIS

void
kern_yield(*int prio*);

void
maybe_yield();

bool
should_yield();

DESCRIPTION

The **kern_yield**() function causes the currently running thread to voluntarily, but unconditionally, surrender its execution to the scheduler. The *prio* argument specifies the scheduling priority to be assigned before the context switch, which has an influence on when execution will resume. Note that the requested priority will take effect until the thread returns to usermode, after which its base user priority will be restored. Valid values for *prio* are any of the PRI_* values defined in <*sys/priority.h*>, as well as the following special values:

PRI_USER Schedule the thread with its base user priority; the value corresponding to setpriority(2) / nice(3).

PRI_UNCHANGED Yield the thread without changing its priority.

The **should_yield**() function checks if sufficient time has passed since the thread's last voluntary context switch that yielding would be a useful service to other threads. It returns *true* when this is the case. See *USAGE NOTES* for an elaboration of what this means.

The **maybe_yield**() function is a helper function for the common task of optionally yielding the processor. Internally, **kern_yield**(*PRI_USER*) will be called if **should_yield**() returns *true*.

USAGE NOTES

Although the kernel supports preemption, this is usually reserved for high-priority realtime or interrupt threads. Kernel worker threads and timesharing threads are not guaranteed to preempt each another. Thus, threads executing in the kernel are expected to behave cooperatively with respect to other threads in the system. The yield functions are mostly intended to be used by threads which perform a lot of work inside the kernel. For example: **maybe_yield**() is called by the vlnru process each time it reclaims a vnode.

The scheduler aims to identify threads which monopolize the CPU, and will schedule them with decreased priority. Threads which regularly yield the processor will be given the chance to run more often. The possibly surprising effect of this is that, depending on the disposition of other threads on the CPU's runqueue, a call to **kern_yield**() does not guarantee that the yielding thread will be taken off the CPU.

With the above considerations in mind, it is advised that code written using **kern_yield**() be measured to confirm that its use has a positive effect on relevant performance or responsiveness metrics. Switching thread contexts has a non-zero cost, and thus yielding the processor too eagerly could have a negative impact on performance.

Additionally, since yielding is a cooperative action, it is advised that the yielding thread release any locks that it may be holding, when possible. Otherwise, threads which have been given the chance to run could end up waiting on the yielding thread to release the lock, largely defeating the purpose of the yield.

SEE ALSO

setpriority(2), nice(3), mi_switch(9)

AUTHORS

This manual page was written by Mitchell Horne <*mhorne@FreeBSD.org*>.