NAME

choosethread, **procrunnable**, **remrunqueue**, **setrunqueue** - manage the queue of runnable processes

SYNOPSIS

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

extern struct rq itqueues[];
extern struct rq rtqueues[];
extern struct rq queues[];
extern struct rq idqueues[];

*struct thread **
choosethread(*void*);

int
procrunnable(*void*);

void
remrunqueue(*struct thread *td*);

void
setrunqueue(*struct thread *td*);

DESCRIPTION

The run queue consists of four priority queues: *itqueues* for interrupt threads, *rtqueues* for realtime priority processes, *queues* for time sharing processes, and *idqueues* for idle priority processes. Each priority queue consists of an array of NQS queue header structures. Each queue header identifies a list of runnable processes of equal priority. Each queue also has a single word that contains a bit mask identifying non-empty queues to assist in selecting a process quickly. These are named *itqueuebits*, *rtqueuebits*, *queuebits*, and *idqueuebits*. The run queues are protected by the *sched_lock* mutex.

procrunnable() returns zero if there are no runnable processes other than the idle process. If there is at least one runnable process other than the idle process, it will return a non-zero value. Note that the *sched_lock* mutex does *not* need to be held when this function is called. There is a small race window where one CPU may place a process on the run queue when there are currently no other runnable processes while another CPU is calling this function. In that case the second CPU will simply travel through the idle loop one additional time before noticing that there is a runnable process. This works because idle CPUs are not halted in SMP systems. If idle CPUs are halted in SMP systems, then this race condition might have more serious repercussions in the losing case, and **procrunnable**() may have to

require that the *sched_lock* mutex be acquired.

choosethread() returns the highest priority runnable thread. If there are no runnable threads, then the idle thread is returned. This function is called by **cpu_switch**() and **cpu_throw**() to determine which thread to switch to. **choosethread**() must be called with the *sched_lock* mutex held.

setrunqueue() adds the thread *td* to the tail of the appropriate queue in the proper priority queue. The thread must be runnable, i.e. *p_stat* must be set to SRUN. This function must be called with the *sched_lock* mutex held.

remrunqueue() removes thread *td* from its run queue. If *td* is not on a run queue, then the kernel will panic(9). This function must be called with the *sched_lock* mutex held.

SEE ALSO

cpu_switch(9), scheduler(9), sleepqueue(9)