NAME

kern_reboot, **shutdown_nice** - reboot, halt, or power off the system

SYNOPSIS

#include <sys/types.h>
#include <sys/systm.h>
#include <sys/reboot.h>

extern int rebooting;

void
kern_reboot(*int howto*);

void
shutdown_nice(*int howto*);

#include <sys/eventhandler.h>

EVENTHANDLER_REGISTER(*shutdown_pre_sync*, *shutdown_fn*, *private*, *priority*);

EVENTHANDLER_REGISTER(*shutdown_post_sync*, *shutdown_fn*, *private*, *priority*);

EVENTHANDLER_REGISTER(*shutdown_final*, *shutdown_fn*, *private*, *priority*);

DESCRIPTION

The **kern_reboot**() function handles final system shutdown, and either halts, reboots, or powers down the system. The exact action to be taken is determined by the flags passed in *howto*.

The relevant flags are:

RB_HALT	Halt the system in-place rather than restarting.
RB_POWEROFF	Power down the system rather than restarting.
RB_POWERCYCLE	Request a power-cycle in addition to restarting.
RB_NOSYNC	Do not sync filesystems during shutdown.
RB_DUMP	Dump kernel memory during shutdown.

The *howto* field, and its full list of flags are described in additional detail by reboot(2).

kern_reboot() performs the following actions:

1. Set the *rebooting* variable to 1, indicating that the reboot process has begun and cannot be

stopped.

2. Set the *kdb_active* variable to 0, indicating that execution has left the kernel debugger, if it was previously active.

3. Unless the RB_NOSYNC flag is set in *howto*, sync and unmount the system's disks by calling vfs_unmountall(9).

4. If rebooting after a panic (RB_DUMP is set in *howto*, but RB_HALT is not set), initiate a system crash dump via **doadump**().

5. Print a message indicating that the system is about to be halted or rebooted, and a report of the total system uptime.

6. Execute all registered shutdown hooks. See *SHUTDOWN HOOKS* below.

7. As a last resort, if none of the shutdown hooks handled the reboot, call the machine-dependent **cpu_reset**() function. In the unlikely case that this is not supported, **kern_reboot**() will loop forever at the end of the function. This requires a manual reset of the system.

kern_reboot() may be called from a typical kernel execution context, when the system is running normally. It may also be called as the final step of a kernel panic, or from the kernel debugger. Therefore, the code in this function is subject to restrictions described by the *EXECUTION CONTEXT* section of the panic(9) man page.

The **shutdown_nice**() function is the intended path for performing a clean reboot or shutdown when the system is operating under normal conditions. Calling this function will send a signal to the init(8) process, instructing it to perform a shutdown. When init(8) has cleanly terminated its children, it will perform the reboot(2) system call, which in turn calls **kern_reboot**().

If **shutdown_nice**() is called before the init(8) process has been spawned, or if the system has panicked or otherwise halted, **kern_reboot**() will be called directly.

SHUTDOWN HOOKS

The system defines three separate EVENTHANDLER(9) events, which are invoked successively during the shutdown procedure. These are *shutdown_pre_sync*, *shutdown_post_sync*, and *shutdown_final*. They will be executed unconditionally in the listed order. Handler functions registered to any of these events will receive the value of *howto* as their second argument, which may be used to decide what action to take.

The *shutdown_pre_sync* event is invoked before syncing filesystems to disk. It enables any action or state transition that must happen before this point to take place.

The *shutdown_post_sync* event is invoked at the point immediately after the filesystem sync has finished. It enables, for example, disk drivers to complete the sync by flushing their cache to disk. Note that this event still takes place before the optional kernel core dump.

The *shutdown_final* event is invoked as the very last step of **kern_reboot**(). Drivers and subsystems such as acpi(4) can register handlers to this event that will perform the actual reboot, power-off, or halt.

Notably, the *shutdown_final* event is also the point at which all kernel modules will have their shutdown (MOD_SHUTDOWN) hooks executed, and when the DEVICE_SHUTDOWN(9) method will be executed recursively on all devices.

All event handlers, like **kern_reboot**() itself, may be run in either normal shutdown context or a kernel panic or debugger context. Handler functions are expected to take care not to trigger recursive panics.

RETURN VALUES

The **kern_reboot**() function does not return.

The **shutdown_nice**() function will usually return to its caller, having initiated the asynchronous system shutdown. It will not return when called from a panic or debugger context, or during early boot.

EXAMPLES

A hypothetical driver, foo(4), defines a *shutdown_final* event handler that can handle system power-off by writing to a device register, but it does not handle halt or reset.

```
void
foo_poweroff_handler(struct void *arg, int howto)
{
	struct foo_softc *sc = arg;
	uint32_t reg;

	if ((howto & RB_POWEROFF) != 0) {
		reg = FOO_POWEROFF;
		WRITE4(sc, FOO_POWEROFF_REG, reg);
	}
}
```

The handler is then registered in the device attach routine:

```
int
foo_attach(device_t dev)
{
        struct foo_softc *sc;

        ...

        /* Pass the device's software context as the private arg. */
        EVENTHANDLER_REGISTER(shutdown_final, foo_poweroff_handler, sc,
            SHUTDOWN_PRI_DEFAULT);

        ...
}
```

This *shutdown_final* handler uses the RB_NOSYNC flag to detect that a panic or other unusual condition has occurred, and returns early:

```
void
bar_shutdown_final(struct void *arg, int howto)
{

        if ((howto & RB_NOSYNC) != 0)
                return;

        /* Some code that is not panic-safe. */
        ...
}
```

SEE ALSO

reboot(2), init(8), DEVICE_SHUTDOWN(9), EVENTHANDLER(9), module(9), panic(9), vfs_unmountall(9)