NAME

init - process control initialization

SYNOPSIS

init
init [**0** | **1** | **6** | **c** | **q**]

DESCRIPTION

The **init** utility is the last stage of the boot process. It normally runs the automatic reboot sequence as described in rc(8), and if this succeeds, begins multi-user operation. If the reboot scripts fail, **init** commences single-user operation by giving the super-user a shell on the console. The **init** utility may be passed parameters from the boot program to prevent the system from going multi-user and to instead execute a single-user shell without starting the normal daemons. The system is then quiescent for maintenance work and may later be made to go to multi-user by exiting the single-user shell (with ^D). This causes **init** to run the */etc/rc* start up command file in fastboot mode (skipping disk checks).

If the *console* entry in the ttys(5) file is marked "insecure", then **init** will require that the super-user password be entered before the system will start a single-user shell. The password check is skipped if the *console* is marked as "secure". Note that the password check does not protect from variables such as *init_script* being set from the loader(8) command line; see the *SECURITY* section of loader(8).

If the system security level (see security(7)) is initially nonzero, then **init** leaves it unchanged. Otherwise, **init** raises the level to 1 before going multi-user for the first time. Since the level cannot be reduced, it will be at least 1 for subsequent operation, even on return to single-user. If a level higher than 1 is desired while running multi-user, it can be set before going multi-user, e.g., by the startup script rc(8), using sysctl(8) to set the *kern.securelevel* variable to the required security level.

If **init** is run in a jail, the security level of the "host system" will not be affected. Part of the information set up in the kernel to support a jail is a per-jail security level. This allows running a higher security level inside of a jail than that of the host system. See jail(8) for more information about jails.

In multi-user operation, **init** maintains processes for the terminal ports found in the file ttys(5). The **init** utility reads this file and executes the command found in the second field, unless the first field refers to a device in */dev* which is not configured. The first field is supplied as the final argument to the command. This command is usually getty(8); **getty** opens and initializes the tty line and executes the login(1) program. The **login** program, when a valid user logs in, executes a shell for that user. When this shell dies, either because the user logged out or an abnormal termination occurred (a signal), the cycle is restarted by executing a new **getty** for the line.

The **init** utility can also be used to keep arbitrary daemons running, automatically restarting them if they

die. In this case, the first field in the ttys(5) file must not reference the path to a configured device node and will be passed to the daemon as the final argument on its command line. This is similar to the facility offered in the AT&T System V UNIX */etc/inittab*.

Line status (on, off, secure, getty, or window information) may be changed in the ttys(5) file without a reboot by sending the signal SIGHUP to **init** with the command "kill -HUP 1". On receipt of this signal, **init** re-reads the ttys(5) file. When a line is turned off in ttys(5), **init** will send a SIGHUP signal to the controlling process for the session associated with the line. For any lines that were previously turned off in the ttys(5) file and are now on, **init** executes the command specified in the second field. If the command or window field for a line is changed, the change takes effect at the end of the current login session (e.g., the next time **init** starts a process on the line). If a line is commented out or deleted from ttys(5), **init** will not do anything at all to that line.

The **init** utility will terminate multi-user operations and resume single-user mode if sent a terminate (TERM) signal, for example, "kill -TERM 1". If there are processes outstanding that are deadlocked (because of hardware or software failure), **init** will not wait for them all to die (which might take forever), but will time out after 30 seconds and print a warning message.

The **init** utility will cease creating new processes and allow the system to slowly die away, if it is sent a terminal stop (TSTP) signal, i.e. "kill -TSTP 1". A later hangup will resume full multi-user operations, or a terminate will start a single-user shell. This hook is used by reboot(8) and halt(8).

The **init** utility will terminate all possible processes (again, it will not wait for deadlocked processes) and reboot the machine if sent the interrupt (INT) signal, i.e. "kill -INT 1". This is useful for shutting the machine down cleanly from inside the kernel or from X when the machine appears to be hung.

The **init** utility will do the same, except it will halt the machine if sent the user defined signal 1 (USR1), or will halt and turn the power off (if hardware permits) if sent the user defined signal 2 (USR2).

When shutting down the machine, **init** will try to run the */etc/rc.shutdown* script. This script can be used to cleanly terminate specific programs such as **innd** (the InterNetNews server). If this script does not terminate within 120 seconds, **init** will terminate it. The timeout can be configured via the sysctl(8) variable *kern.init_shutdown_timeout*.

init passes "single" as the argument to the shutdown script if return to single-user mode is requested. Otherwise, "reboot" argument is used.

After all user processes have been terminated, **init** will try to run the */etc/rc.final* script. This script can be used to finally prepare and unmount filesystems that may have been needed during shutdown, for instance.

The role of **init** is so critical that if it dies, the system will reboot itself automatically. If, at bootstrap time, the **init** process cannot be located, the system will panic with the message "panic: init died (signal %d, exit %d)".

If run as a user process as shown in the second synopsis line, **init** will emulate AT&T System V UNIX behavior, i.e., super-user can specify the desired *run-level* on a command line, and **init** will signal the original (PID 1) **init** as follows:

Run-level	Signal	Action
0	SIGUSR1	Halt
0	SIGUSR2	Halt and turn the power off
0	SIGWINCH	Halt and turn the power off and then back on
1	SIGTERM	Go to single-user mode
6	SIGINT	Reboot the machine
c	SIGTSTP	Block further logins
q	SIGHUP	Rescan the ttys(5) file

KERNEL ENVIRONMENT VARIABLES

The following kenv(2) variables are available as loader(8) tunables:

init_chroot

If set to a valid directory in the root file system, it causes **init** to perform a chroot(2) operation on that directory, making it the new root directory. That happens before entering single-user mode or multi-user mode (but after executing the *init_script* if enabled). This functionality has generally been eclipsed by rerooting. See reboot(8) **-r** for details.

init_exec

If set to a valid file name in the root file system, instructs **init** to directly execute that file as the very first action, replacing **init** as PID 1.

init_script

If set to a valid file name in the root file system, instructs **init** to run that script as the very first action, before doing anything else. Signal handling and exit code interpretation is similar to running the */etc/rc* script. In particular, single-user operation is enforced if the script terminates with a non-zero exit code, or if a SIGTERM is delivered to the **init** process (PID 1). This functionality has generally been eclipsed by rerooting. See reboot(8) **-r** for details.

init_shell

Defines the shell binary to be used for executing the various shell scripts. The default is "/bin/sh". It is used for running the *init_exec* or *init_script* if set, as well as for the */etc/rc*,

/etc/rc.shutdown, and */etc/rc.final* scripts. The value of the corresponding kenv(2) variable is evaluated every time **init** calls a shell script, so it can be changed later on using the kenv(1) utility. In particular, if a non-default shell is used for running an *init_script*, it might be desirable to have that script reset the value of *init_shell* back to the default, so that the */etc/rc* script is executed with the standard shell */bin/sh*.

FILES

/dev/console	system console device
*/dev/tty**	terminal ports found in ttys(5)
/etc/ttys	the terminal initialization information file
/etc/rc	system startup commands
/etc/rc.shutdown	system shutdown commands
/etc/rc.final	system shutdown commands (after process termination)
/var/log/init.log	log of rc(8) output if the system console device is not available

DIAGNOSTICS

getty repeating too quickly on port %s, sleeping. A process being started to service a line is exiting quickly each time it is started. This is often caused by a ringing or noisy terminal line. *Init will sleep for 30 seconds, then continue trying to start the process.*

some processes would not die; ps axl advised. A process is hung and could not be killed when the system was shutting down. This condition is usually caused by a process that is stuck in a device driver because of a persistent device error condition.

SEE ALSO

kill(1), login(1), sh(1), ttys(5), security(7), getty(8), halt(8), jail(8), rc(8), reboot(8), shutdown(8), sysctl(8)

HISTORY

An **init** utility appeared in Version 1 AT&T UNIX.

CAVEATS

Systems without sysctl(8) behave as though they have security level -1.

Setting the security level above 1 too early in the boot sequence can prevent fsck(8) from repairing inconsistent file systems. The preferred location to set the security level is at the end of */etc/rc* after all multi-user startup actions are complete.