NAME

init − process control initialization

SYNOPSIS

/etc/init

DESCRIPTION

Init is a general process spawner. Its primary role is to create processes from a script stored in the file **/etc/inittab** (see *inittab*(5)). This file usually has *init* spawn *getty*'s on each line that a user may log in on. It also controls autonomous processes required by any particular system.

Init considers the system to be in a *run level* at any given time. A *run level* can be viewed as a software configuration of the system where each configuration allows only a selected group of processes to exist. The processes spawned by *init* for each of these *run levels* is defined in the *inittab* file. *Init* can be in one of eight levels, **0-6** and **S** (**s**). The *run level* is changed by having a privileged user run **/etc/init** (which is linked to **/bin/telinit**). This user spawned *init* sends appropriate signals to the orginal *init* spawned by the operating system when the system was rebooted, telling it which level to change to.

Init is invoked inside UNIX as the last step in the boot procedure. The first thing *init* does is to look for **/etc/inittab** and see if there is an entry of the type *initdefault* (see *inittab*(5)). If there is, *init* uses the level specified in that entry as the initial *run level* to enter. If this entry is not in *inittab* or *inittab* is not found, *init* requests that the user enter a run level from the virtual system console, **/dev/syscon**. If an **S** (**s**). is entered, *init* goes into the *SINGLE USER* level. This is the only level that doesn't require the existence of a properly formated *inittab* file. If **/etc/inittab** doesn't exist, then by default the only legal level that *init* can enter is the *SINGLE USER* level. In the *SINGLE USER* level the virtual console terminal **/dev/syscon** is opened for reading and writing and the command **/bin/su** is invoked immediately. To exit from the *SINGLE USER run level* one of two options can be elected. First, if the shell is terminated (via an end-of-file), *init* will reprompt for a new run level. Second, the *init* or *telinit*(1M) command can signal *init* and force it to change the *run level* of the system.

When attempting to boot the system, failure of *init* to prompt for a new run level may be due to the fact that the device **/dev/syscon** is linked to a device other than the physical system teletype (**/dev/systty**). If this occurs, *init* can be forced to relink **/dev/syscon** by typing a delete on the system teletype which is colocated with the processor.

When *init* prompts for the new run level the operator may only enter one of the digits **0** through **6** or the letters **S** or **s**. If **S** is entered *init* operates as previously described in *SINGLE USER* mode with the additional result that **/dev/syscon** is linked to the user's terminal line, thus making it the virtual system console. A message is generated on the physical console, **/dev/systty**, saying where the virtual terminal has been relocated.

If a **0** through **6** is entered *init* enters the corresponding *run level*. Any other input will be rejected and the user will be reprompted. If this is the first time *init* has entered a *run level* other than *SINGLE USER*, *init* first scans *inittab* for special entries of the type *boot* and *bootwait*. These entries are performed, providing the level entered matches that of the entry before any normal processing of *inittab* takes place. In this way any special initialization of the operating system,such as mounting file systems, can take place before users are allowed onto the system. The *inittab* file is scanned to find all entries that are to be processed for that *run level*.

Run level 0 is usually defined by the user to contain all of the terminal processes and daemons that are spawned in the multi-user environment.

In a multi-user environment, the *inittab* file is usually set up so that *init* will create a process for each terminal on the system.

For terminal processes, ultimately the shell will terminate because of an end-of-file either typed explicitly or generated as the result of hanging up. When *init* receives a child death signal,

telling it that a process it spawned has died, it records the fact and the reason it died in **/etc/utmp** (and **/etc/wtmp** if it exits) (see *who*(1)). A history of the processes spawned is kept in **/etc/wtmp** if such a file exists.

To spawn each process in the *inittab* file, *init* reads each entry and for each entry which should be respawned, it forks a child process. After it has spawned all of the processes specified by the *inittab* file, *init* waits for one of its descendant processes to die, a powerfail signal, or until *init* is signaled by *init* or *telinit*(1M) to change the system's run level. When one of the above three conditions occurs, *init* re-examines the *inittab* file. New entries can be added to the *inittab* file at any time; however *init* still waits for one of the above three conditions to occur. To provide for an instantaneous response the **"init (telinit) Q"** command can wake *init* to reexamine the *inittab* file.

If *init* receives a *powerfail* signal (*SIGPWR* (*signal*(2))) and is not in *SINGLE USER* mode, it scans *inittab* for special powerfail entries. These entries are invoked (if the levels permit) before any further processing takes place. In this way *init* can perform various cleanup and recording functions whenever the operating system experiences a power failure.

When *init* is requested to change run levels (via *telinit*(1M)), *init* sends the warning signal (signal 15) to all processes that are undefined in the target *run level*. *Init* waits 20 seconds before forcibly terminating these processes via the kill signal (signal 9).

FILES

/etc/inittab,
/etc/utmp,
/etc/wtmp,
/dev/syscon,
/dev/systty,
/bin/sh,
/bin/su

SEE ALSO

getty(1M), login(1), sh(1), telinit(1M), inittab(5), utmp(5)

DIAGNOSTICS

If *init* finds that it is continuously respawning an entry from **/etc/inittab** more than 10 times in 2 minutes, it will assume that there is an error in the command string, and generate an error message on the system console, and refuse to respawn this entry until either 5 minutes has elapsed or it receives a signal from a user *init* (*telinit*). This prevents *init* from eating up system resources when someone makes a typographical error in the *inittab* file or a program is removed that is referenced in the *inittab*.