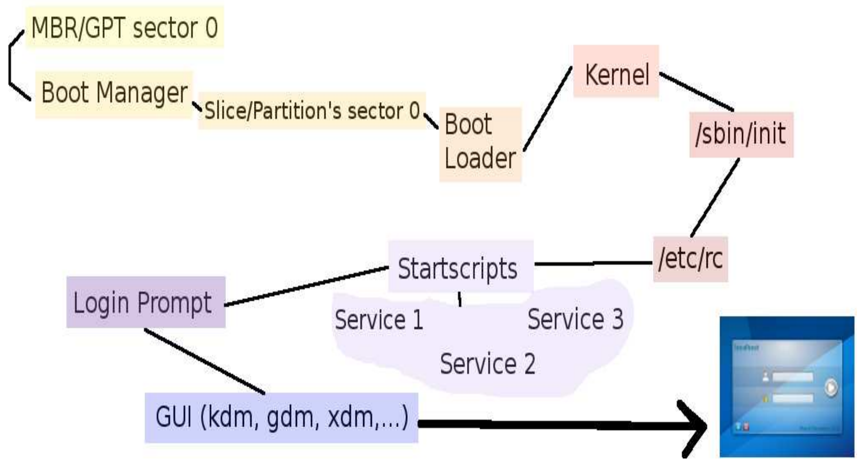
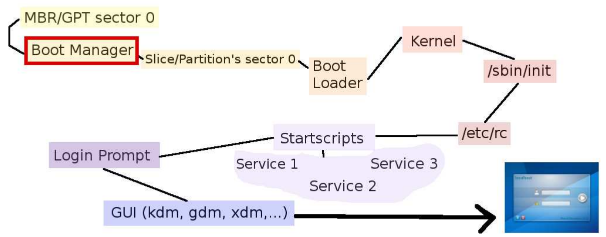
Booting



Starting to Boot (stage 0 boot)

- BIOS = basic input/output system, ROM...EEPROM...Flash
- BIOS locates MBR / GPT
- MBR/GPT code =*boot manager*, 512 bytes, boot menu
  - boot0, standard FreeBSD boot manager
  - GRUB,
  - standard PC MBR (searches active slice)
  - NTLDR, Vista MBR (Windows systems)
- MBR code reads boot loader (BIOS I/O)

Boot Manager / Boot Loader



a *boot manager* is independent from OS (on MBR)

a *boot loader* is OS specific (on slice)

Boot Manager: Select Partition with a Root FS

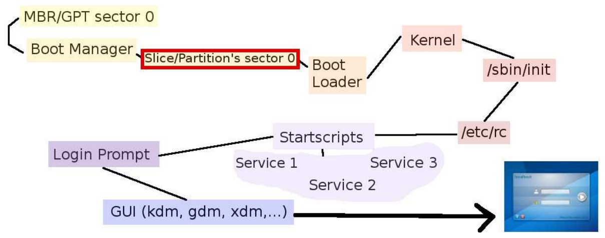
FreeBSD `boot0` start screen (file `/boot/boot0`, 512 bytes)

```
F1 DOS
F2 FreeBSD
F3 Linux
F4 ??
F5 Drive 1

Default: F2
```

source code directory `/usr/src/sys/boot/i386/boot0`

Prepare Loading of Boot Loader



FreeBSD `boot1` (file `/boot/boot1`, 512 bytes)

Located in boot sector of bootable slice ⇝ 512 bytes.

Knows `bsdlabel` data structure.

Finds and loads `boot2` (in the following 15 sectors)

Locate Boot Loader on Partition

FreeBSD `boot2` screenshot (file `/boot/boot2`, 7K bytes)

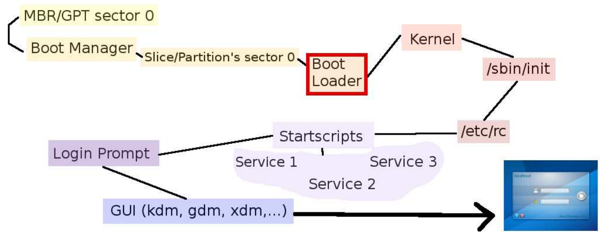
```
>> FreeBSD/i386 BOOT
Default: 0:ad(0,a)/boot/loader
boot:
```

Knows how to find files on a UFS filesystem on it

until now, everything coded in machine language directly

Finds and loads `/boot/loader`, (217K)

Boot Loader: Prepare Loading of OS



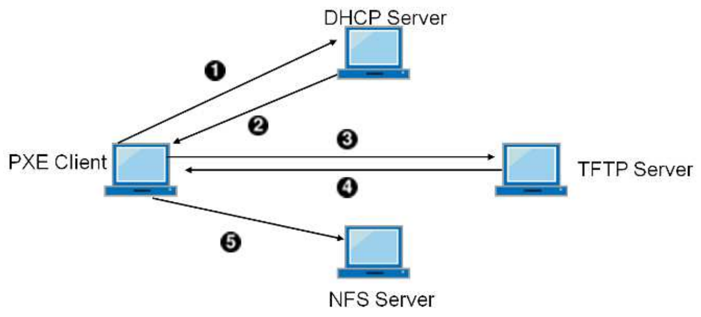
`/boot/loader`

programmed in C, can do:

- probe for a console
- probe for disks,
- figure out what disk it is booting from
- load kernel/modules

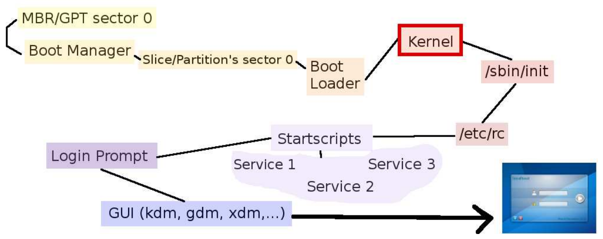
Side note: PXEBOOT

preboot-exec-environment (Intel), on ethernet card



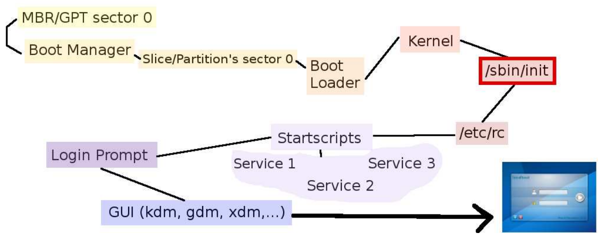
⇝ diskless machines.

Kernel



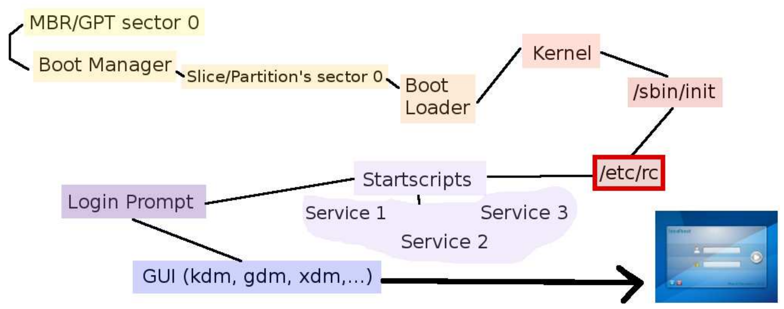
- initialize device drivers
- detect type of root filesystem, if unknown then STOP
- mount root filesystem read–only
- start process `/sbin/init` with PID=1
- init: `/etc/rc`, rc = resource configuration

/sbin/init



- maintains processes for the terminal ports (getty)
- `SIGHUP` $\rightsquigarrow$ re–read `/etc/ttys`
- `SIGTERM` $\rightsquigarrow$ single-user-mode
- `SIGINT` $\rightsquigarrow$ kill all processes, i.e. shutdown
- `if init terminates` $\rightsquigarrow$ reboot

/etc/rc



/etc/rc, the original idea

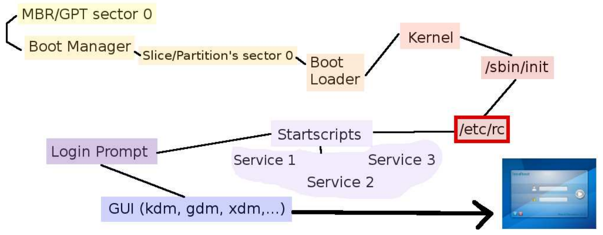
simple shell script, typical start sequences:

```
...
if [ "X${httpd_flags}" != X"NO" ]; then
        # Clean up left-over httpd locks
        rm -f /var/www/logs/{ssl_mutex,httpd.lock,accept.lock}.*
        echo -n ' httpd';    /usr/sbin/httpd ${httpd_flags}
fi

if [ "X${ftpd_flags}" != X"NO" ]; then
        echo -n ' ftpd';     /usr/libexec/ftpd ${ftpd_flags}
fi

if [ "X${identd_flags}" != X"NO" ]; then
        echo -n ' identd';   /usr/libexec/identd ${identd_flags}
fi
...
```

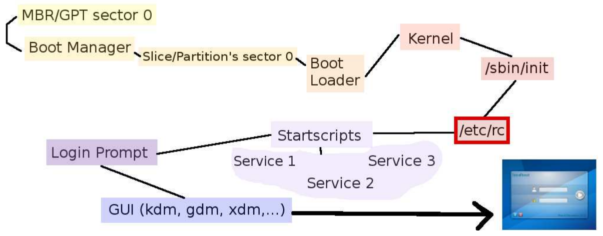
/etc/rc configuration (1)



variables . . .

```
httpd_flags="NO"
ftpd_flags="-t 120"
```

/etc/rc configuration (2)



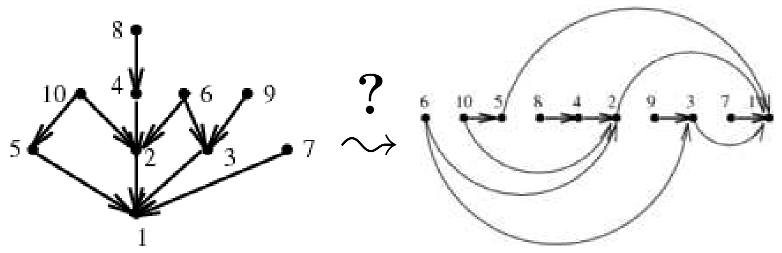
. . . are configured in startup config files

```
/etc/rc.conf
/etc/rc.conf.local
```

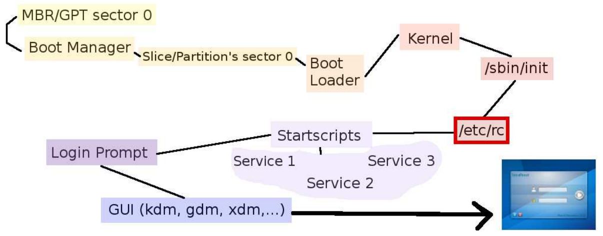
. . . and loaded in rc as follows . . .

```
. /etc/rc.conf
```

Problem: Dependencies between Services



/etc/rc, Enhanced Version



used in FreeBSD/NetBSD

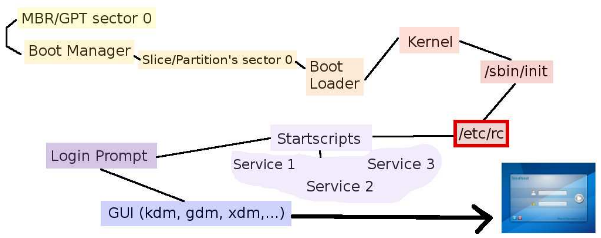
- rc: order all startup scripts by `rcorder`
- /etc/rc.d/*: startup scripts contain dependencies
- rc: starts scripts in `/etc/rc*.d` according to output of **topological sort**

example: RPC service `rpcbind`

```
#!/bin/sh
#

# PROVIDE: rpcbind
# REQUIRE: NETWORKING ntpdate syslogd named
```

/etc/rc, SYSVINIT Version



from UNIX system V, used in Linux, Solaris

⇝`/etc/inittab` exists, configures ,,runlevels"

runlevel: state of a system (which set of services is active)

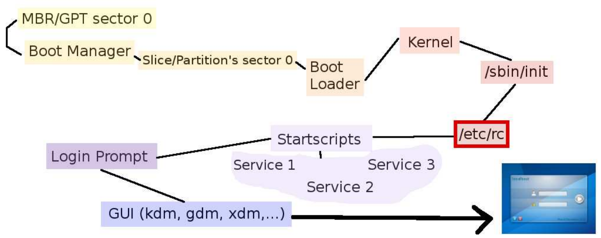
/etc/rc, SYSVINIT Version

- running (runlevels 2, 3, 5)
- shutdown (runlevels 0, 6)
- single user (runlevels 1, S)

normal operation: runlevels 2 or 3 (or 5)

determine set of scripts to be executed

/etc/rc, SYSVINIT Version

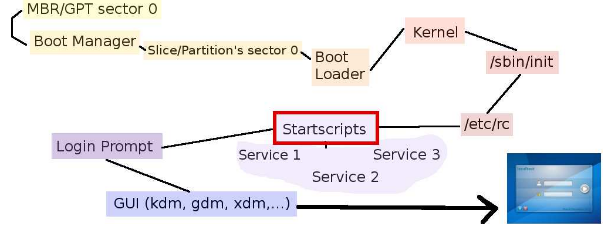


per runlevel there is a directory of softlinks

example `/etc/init.d/rc2.d`

```
...
lrwxrwxrwx 1 root root S05network -> ../network
lrwxrwxrwx 1 root root S06syslog -> ../syslog
lrwxrwxrwx 1 root root S07splash_early -> ../splash_early
lrwxrwxrwx 1 root root S10alsasound -> ../alsasound
lrwxrwxrwx 1 root root S10cups -> ../cups
...
```

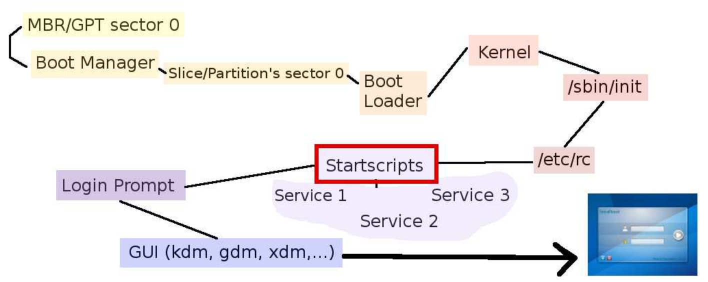
Startscripts (1)



each daemon/service has a start script

- checks configuration files
- determines if service may be started
- starts service (usually in `/usr/sbin`)

Startscripts (1)

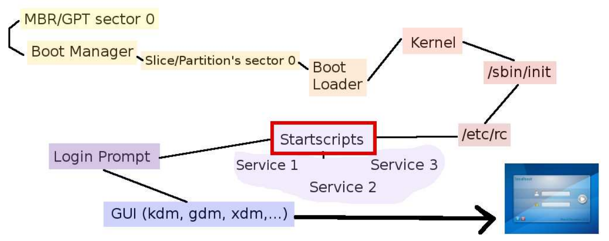


also control shutdown of service

should implement parameters

`start` `stop` `restart` `reload` `status`

Startscripts (2, FreeBSD, NetBSD)



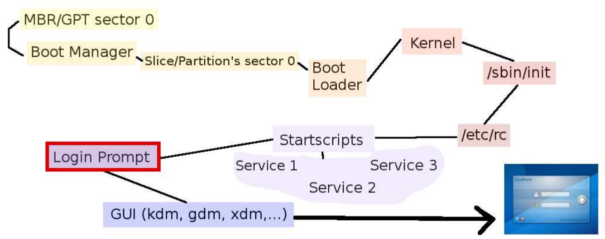
each startscript is located in `/etc/rc.d`

uses script infrastructure from `/etc/rc.subr`

points to service that must be started

```
name="sshd"
rcvar=`set_rcvar`
command="/usr/sbin/${name}"
start_precmd="sshd_precmd"
pidfile="/var/run/${name}.pid"
extra_commands="keygen reload"
```

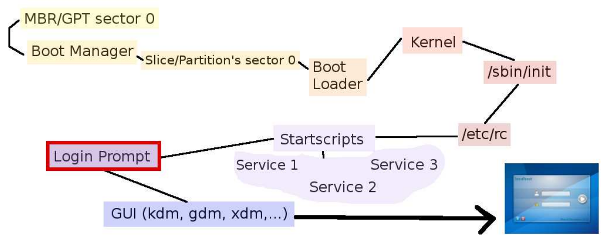
Single User Mode, Definition



- only root is allowed to log in
- only root filesystem is mounted

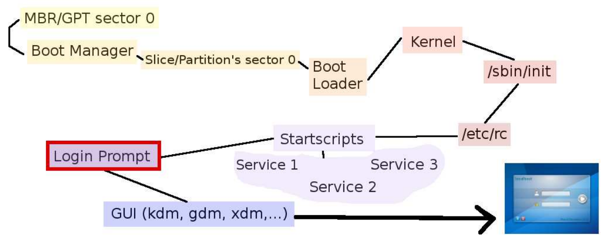
use this mode only for special tasks

Single User Mode, Examples



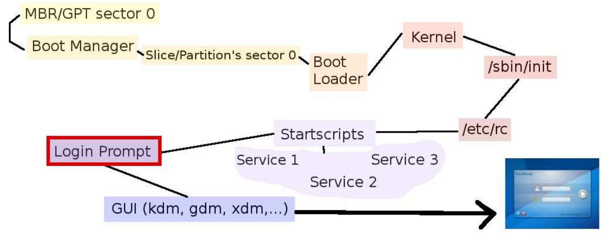
- upgrade system (kernel, system lib, tools)
- repair filesystems after system crash
- forensics/clean-up after system break–in
- fix problems in critical system files
  - `/etc/fstab`
  - `/etc/inittab` (if SYSVINIT system)
- restore files from backup

Invoking Single User Mode



- Use `shutdown` without `-h` or `-r`.
- On `loader` prompt use `boot -s`
- On `loader` menu use *single user*

Login Prompt (text–oriented)

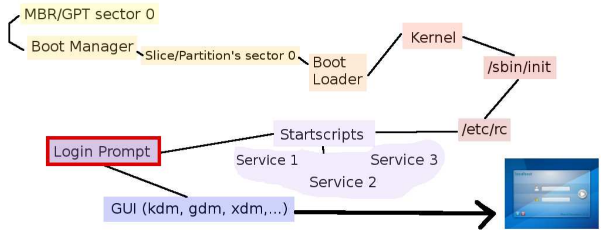


several text–oriented login screens

via `/etc/ttys`

```
# name   getty                       type      status
ttyv0    "/usr/libexec/getty Pc"     cons25l1  on  secure
ttyv1    "/usr/libexec/getty Pc"     cons25l1  on  secure
ttyv2    "/usr/libexec/getty Pc"     cons25l1  on  secure
...
```

Side note: Console

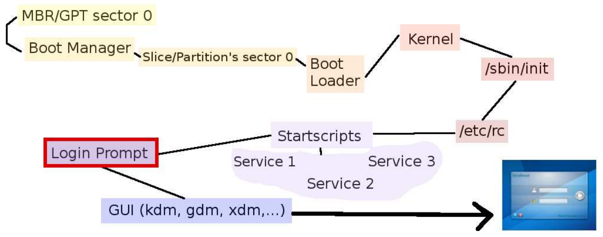


a text-mode terminal (usually 80x25)

- usually used for `root` login
- direct connection to the motherboard
  - PS/2
  - serial
  - ...

- may be used to control root access to the machine (physical presence required)
- change resolution with
  - `vidcontrol` (FreeBSD) (even 1024x768 resolution with MODE_279)
  - kernel boot parameter (Linux)

Side note: Console (2)



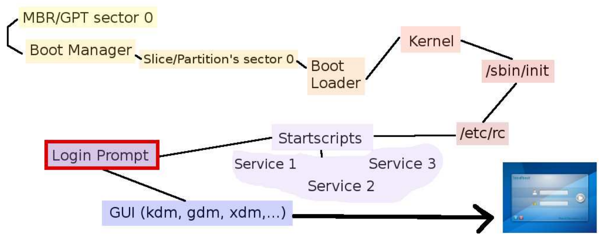
boot and have root ? FreeBSD–Version

see `/etc/ttys` on a FreeBSD-system

```
# If console is marked "insecure",
# then init will ask for the root password
# when going to single-user mode.

console none      unknown on insecure
```

Side note: Console (3)



boot and have root ? Linux–Version

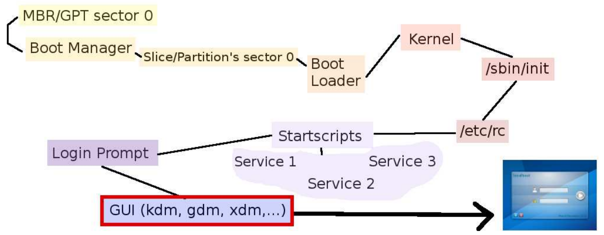
start from GRUB in single user mode

(append `single` on kernel–line and init=/bin/bash)

first process is root shell (no password needed)

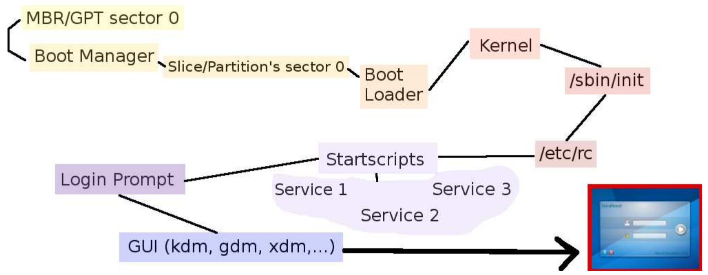
⇝must set password for GRUB/LILO

Login Prompt (for GUI)

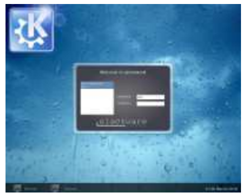


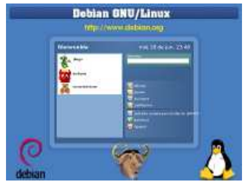
- depends on Xorg
  (GUI base system, formerly X11)
- requires root privileges (graphics card)
  - insecure: SETUID `/usr/local/bin/X` from terminal,
  - more secure: display manager
    (xdm, kdm, gdm, slim, . . . as root)

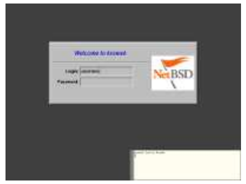
Login Prompt (for GUI)

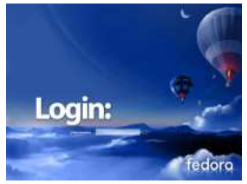


Login Prompt (Examples)


KDM

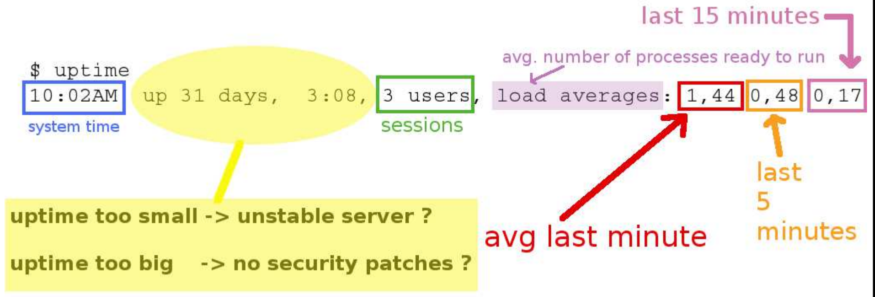

GDM


XDM


SLIM

System Up and Running

Load Average: How Busy the System Is



```
$ uptime
10:02AM  up 31 days,  3:08, 3 users, load averages: 1,44 0,48 0,17
```

last 15 minutes
avg. number of processes ready to run
system time
sessions
uptime too small -> unstable server ?
uptime too big -> no security patches ?
avg last minute
last 5 minutes

We are now going to shut down the system

System Halt (1)

the command `shutdown` halts the system

this command is reserved to the super–user

- halt with `shutdown -h` (`-p` power off)
- reboot with `shutdown -r`
- shutdown requires a time (when to shutdown)
- shutdown notifies all users via the `wall` command

Examples:

- `shutdown -h 11:15`
- `shutdown -r +20`
- `shutdown -c` (Linux: cancel running shutdown)

System Halt, Respect Your Users

not immediately

not throwing out users

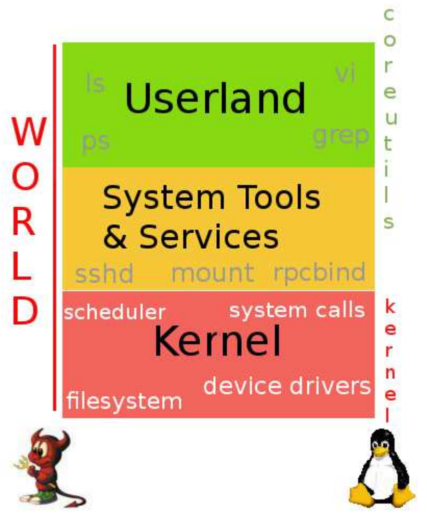
not, if load > 0

$\rightsquigarrow$make sure: no users, no processes, advance notice

System Halt (2)

- kills all processes
  - first per `TERM` signal
  - then per `KILL` signal
- writes all buffered data to disk (`sync`)

8. Kernel



Installing a New Kernel

Usually **not necessary**, except you want

- install security patches
- faster boot-up
- less memory usage
- support for extra hardware components

Installing a New Kernel (FreeBSD)

usually preceded by `make buildworld` ⇝ userland tools

- kernel sources ⇝ `/usr/src/sys`
- configuring kernel through options below `conf`
- `make kernel` installs to `/boot/kernel/kernel`
- reboot

Note: path is fixed, save previous version by changing name

Note: uses system compiler (gcc 4.2.1 or clang 3.4.1)

Options in a New Kernel (Example, FreeBSD)

```
options SCHED_ULE     # ULE scheduler
options PREEMPTION    # Enable kernel thread preemption
options INET          # InterNETworking
options INET6         # IPv6 communications protocols
options SCTP          # Stream Control Transmission Protocol
options FFS           # Berkeley Fast Filesystem
options UFS_ACL       # Support for access control lists
options QUOTA         # Enable disk quotas for UFS
options NFSCL         # New Network Filesystem Client
options NFSD          # New Network Filesystem Server
...
```

Options in a New Kernel (Example, FreeBSD)

```
device  em            # Intel PRO/1000 Gigabit Ethernet
device  igb           # Intel PRO/1000 PCIE Server Gigabit
...
device  uhci          # UHCI PCI->USB interface
device  ohci          # OHCI PCI->USB interface
device  umass         # Disks/Mass storage -> option scbus
...
device  sound         # Generic sound driver (required)
device  snd_via8233   # VIA VT8233x Audio
```

Installing a New Kernel (Linux, 3.0 kernel)

- get the kernel source from `ftp.kernel.org`
  example: `linux-3.14.12.tar.xz`
- unpack the kernel source (unxz, tar)
- configure the kernel source `make menuconfig`
- build kernel and modules `make`
- install kernel and modules `make modules_install install`
- insert section into bootloader configuration
- reboot

Note: needs gcc-3.2, can choose install dir with make–option