# Thoughts about Init

Lutz Donnerhacke

2048/39F37F5D 1996/04/25

A4 C1 50 8F 00 D9 28 60

70 BB 0B 5D D9 3A 0B B6

# A conceptional clean system

Tried to build a conceptional clean system.
Starting with SysV-Init:

- errornous heuristics to determine the runlevel and the running programs
- dire heuristics and methods to bring the system down
- horrible heuristics in external programs to restart dead daemons
- `fastboot`: hardcoded system calls
- *tylor* (look for A E R O S M I T H)

## What must init handle ?

Init is started by the kernel as PID 1 to bring up the system.

Kernel signals external dangers (`SIGPWR`) to PID 1.

PID 1 is the fallback substitute for processes losing there immediate parent.

*What can be avoided ?*

*Fork and exec just an other startup tool.*

*How to find the correct PID of the daemon?*

*How to find the dead program to a PID? A daemon?*

# Daemons

Current daemons are (usually) started from a shell script. To nullify the consequences of the script dead, they:

- leave the process group
- leave the session group
- *try* to close open descriptors

To reconfigure, supervise and kill them:

- use `killall` $\Rightarrow$ bofhing users
- hope that `/var/run/<name>.pid` works

# Design principles of daemons

Assume, that:

- all unnecessary file descriptors are already closed.
- all signals are send intentionally.
- exiting this daemon process means end of provided services.

*Impossible ?*

# Design principles of init

- Start all daemons to get the corresponding PIDs.
- Stop/Reconfigure/... all daemons.
- Handle dead daemons correctly.
- Know which resource requires other resources and brings up the system as complete and as fast as possible without trying impossible actions
  $\Rightarrow$ parallelize actions.
- Signal handling depends on running services.

# Services

```
service multi needs net tty var
  on start do $ECHO "System is up.";
  on stop  do $ECHO "System is going down.";
  on signal(SIGPWR) do switch "halt";
end service multi

service tty needs var
  on start do
    sequence
      $GETTY = /sbin/getty getty;
      parallel
        $getty1 = pid($GETTY "tty1");
        $getty2 = pid($GETTY "tty2");
      end parallel
    end sequence
  on stop do $KILL "-SIGTERM" $getty1 $getty2;
  on dead($getty1) do $getty1 = pid($GETTY "tty1");
  on dead($getty2) do $getty2 = pid($GETTY "tty2");
end service tty

service var
  on start do
    parallel
      $ECHO = "/bin/echo" "echo";
      $SH   = "/bin/sh"   "sh";
      $KILL = "/bin/kill" "kill";
    end parallel
end service var
```

# Communication channel

To maintain init:

- a service uses >`start socket`< to create a unix domain socket (`600 root`)
- on request a client socket is generated (back channel)
- send ordinary commands
- get progress status (what's on)
- catch program `stdout` and `stderr` prepended by PID

⇒ A frontend to display the startup progress and problems incl. scrollable error output is possible.

# Commands

**switch:** stop all unnecessary services and start all necessary ones, *optional*

**start:** start all necessary services if not running

**stop:** stop this service only

**shutdown:** stop this service after stopping all dependent services

**reload:** cause the daemons to reload there config, fallback: stop, start

**restart:** shutdown followed by start of all involved services

# Parameters

```
service reboot needs var
  on call do
    sequence
      $WALL "System is going to reboot.";
      # Wait for panic abort
      /usr/bin/sleep sleep $1;
      # Set a new startup command (binary patch)
      /sbin/lilo lilo -R $2 $3 $4 $5 $6 $7 $8 $9 $10 $11
    end sequence
  on start do
    sequence
      $ECHO "System is rebooting.";
      reboot;
    end sequence
  on stop  do $WALL "Rebooting aborted.";
end service reboot
```

⇒ `init reboot 16 win2000`

To add error checking: start a shell script.

# Usage

**lutz:...:/bin/init -d -n 5 -c ~lutz/.init**

```
service X
  on start do
    sequence
      $x = pid(/usr/bin/X11/X X);
      parallel
        $xpmroot = pid(/usr/bin/X11/xpmroot xpmroot
                           ~/.X11/background.xpm);
        $mainxterm = pid(/usr/bin/xterm xterm
                             -geometry 80x42+8+8);
      end parallel
    end sequence
  on error    do switch console; # don't catch stop errors
  on dead($x) do switch console;
  on dead($mainxterm)
    $mainxterm = pid(/usr/bin/xterm xterm
                         -geometry 80x42+8+8);
  on signal(SIGHUP) do /usr/bin/X11/xmessage Stop working!;
  on signal(SIGTERM) do shutdown X;
  on stop do /bin/kill kill -SIGHUP $x;
[...]
```

# Reconfiguration

Every lexed service has an associated CRC. So comments or formatting changes does not harm.

If a change was detected, the running service is stopped using the old configuration and is started using the new one.

# Discussion

Any questions ?

Trial version (missing features & bugs)

`ftp.iks-jena.de:/pub/mitarb/lutz/unix/`