minit

Felix von Leitner
felix-minit@fefe.de

September 2004

What is this all about?

This talk is about a new init program called minit. Several itches needed scratching:

1. Typical Linux distributions boot painfully slow
2. If a boot script fails, the whole boot process stops
3. If sshd is killed, it is not automatically restarted
4. The shutdown script does not know the PID of the daemon (ugly failure modes where you accidentally kill the wrong process)
5. lack of flexibility (start one more or one less service?)

Why is fast booting so important?

If an important server has an uptime guarantee of 99.99999%, that means only about 5 minutes total downtime per year. On typical PC servers a reboot is quite lengthy: the BIOS takes two minutes, the kernel takes 20 seconds, and the boot scripts take 30-90 seconds. The orderly shutdown takes another 20 seconds. That leaves only one minute for doing the actual maintenance (like changing a RAM or CPU module.

Reducing the boot script time to 5-15 seconds actually doubles the time for maintenance.

Prior art

All of these problems are really old and well-known. There is much prior art on the topic. Unfortunately, most of it is not in widespread use.

I personally used Dan Bernstein's daemontools to manage and monitor my daemons. Like most of Dan's work, it works like a charm and solves all the problems it set out to solve really well.

It works like this: (next slide)

Prior art – daemontools

1. `svscan` starts a `supervise` process for each directory under `/service`.
2. `supervise` starts `./run`, which starts the daemon so that it does not background itself.
3. The OS tells `supervise` when the daemon terminates. supervise then starts `./run` anew.
4. You can tell `supervise` to stop a daemon (`svc -d /service/sshd`) or to not run one in the first place (`touch /service/sshd/down`) at next boot.
5. `svc` can also send signals to processes (svc -t terminates, -a sends SIGALRM, -p sends SIGSTOP...).

Other nice properties of daemontools

- If `/service/sshd/log` exists, another supervise is started for it, and the stdout of `/service/sshd` will be piped to the stdin of `/service/sshd/log`.
- No log data is lost (syslogd failed, disk full) but daemon blocks.
- The logging service can than buffer the output without fear of losing buffered log messages if the main service segfaults.
- Adding a new service is easily automated.
- Services are restarted by supervise with supervise's environment, not like sysvinit with the admin's current shell environment.

Downside of daemontools

daemontools only has one real downside: all those supervise processes.

I like my system to run as few processes as possible, so I can find my processes without grep and less.

Also, daemontools has no mechanism for doing boot-time initialization. You can only start independent services.

If service A needs service B, you start both, service B then does not find service A, complains, and exits. This repeats until service A is up and running. This is ugly.

If you rm -rf /service/fnord, the supervise for it stays. And you can't kill it any longer because you don't know which supervise it is.

Improving on daemontools – initialization

I have always liked the `service/run` concept a lot.

- initialization is stuff like mount /, ifconfig, route, loadkeys.
- basically just like daemon services, only don't restart them
- normal Linux boot process is so slow because it's lots of /bin/sh code
- but we need mostly "ifconfig eth0 10.0.0.4 ..." and "mount /proc"
- allow command line parameters to be given to run! Run can then be a symlink instead of a shell script.

Improving on daemontools – initialization

```
# cd /etc/minit
# ls -l sshd
-rw-r--r--  1 root  root      4 Jul 28  2003 params
lrwxr-xr-x  1 root  root     19 Jul 28  2003 run ->
                                  /opt/diet/sbin/sshd
# cat sshd/params
-6D
# cat net/eth0/params
eth0
10.0.0.6
netmask
255.255.255.0
#
```

Improving on daemontools – initialization

- turns out there are more initializations than daemons
- so default should be ”do not restart”
- to tell minit to restart sshd, touch sshd/respawn
- installing a package should create a service directory for it
- however, it should not run the service immediately
- so the default should be not to run all the services
- what if route needs ifconfig to be run first?

Improving on daemontools – run order

- with shell scripts, it's implicit
- sysvinit names the scripts and uses globbing
- so called "run levels": collection of services to run
- all kind of ugly. Why not have a dependency tree?
- route needs ifconfig: `echo net/eth0 > net/route/depends`
- can do independent initializations in parallel

Improving on daemontools – run order

- two dependency types: soft and hard.
- soft: samba depends on route (but will start anyway).
- hard: route depends on ifconfig (will not start unless ran and terminated)
- Solution: `touch ifconfig/sync`

Improving on daemontools – which services to run

- touch sshd/runme? Starting minit would mean traversing the service directory (slow and ugly).
- So minit runs one default service, called `default`
- The default service starts the others
- How to tell minit which services to start?
- Solution: put them in default's dependencies.

Handling Ctrl-Alt-Del and Keyboard Request

init (the process with PID 1) can tell the kernel that it wants to handle Ctrl-Alt-Del and Keyboard Request (Alt-UpArrow).

Pressing Ctrl-Alt-Del then sends SIGINT to init (SIGWINCH for kbreq).

minit handles this by spawning the services `ctrlaltdel` or `kbreq`. I use ctrlaltdel for reboot and kbreq for shutdown.

Unanticipated problems

minit records the time stamp when the process was started. You can then use `msvc sshd` to ask for how long the service has been running.

Initialization involves setting the time zone on my notebook. This sets the time back by one hour for me. Thus, msvc output was wrong by one hour.

daemontools does not have this problem because it is started after initialization is complete. sysvinit does not have this problem because it does not record the time.

I ended up kludging: minit detects if the time moves back or jumps by more than 30 seconds. minit runs in a loop waiting for signals or commands from msvc, interrupted every 5 seconds.

How should msvc talk to minit?

The sysvinit way is to use a Unix Domain socket at `/dev/initctl`. Problem is: this requires kernel support for Unix domain sockets and a writable /dev directory.

I would like to support minimal embedded Linux, so I assume that the kernel has support disabled for everything it can and nothing is writable (e.g. boot from a CD-ROM with no ram disk or initrd).

This means: No Unix domain sockets. No `/proc` (otherwise could write something to `/proc/1/fd/3` and read answer from `/proc/1/fd/4`). Filesystem is not writable. `/dev` is not mounted yet. No IP support in kernel (i.e. no socket on 127.0.0.1).

How should msvc talk to minit?

Only option left: two FIFOs, `/etc/minit/in` and `/etc/minit/out`, which make install creates there.

msvc uses fcntl locking to avoid race conditions. These work even on FIFOs on read-only file systems.

How to boot in ”single-user mode”?

The obvious way is of course to use `init=/bin/sh` as kernel argument.

However, it turns out that init gets the kernel arguments as command line arguments!

So minit does this: if any of the command line arguments turn out to be runnable services, minit runs them and does not run the default service. Otherwise it runs the default service. `-sshd` can be used to not start sshd.

This means that you can have one grub selection start in server mode, and one start in X mode, just by passing ”default xmode” on the command line and then having the xmode service started additionally to default.

How do I know which services are running?

Since this is not often needed, minit has no built-in way to do it.

However, you can do

```
# find /etc/minit -type d | xargs -n 1 msvc
```

minit ships a script called `listpids` that does something like this to output all the running services.

Another way is to use `msvc -D default` (lists the services running because default depended on them) and then call msvc -D again for each of them, and then on their dependencies.

Why is it called minit?

Because it is really small, minimal even (even more so when linked with the diet libc), and it is an implementation of /sbin/init.

```
$ ls -l minit msvc
-rwxr-xr-x  1 leitner  users    6712 Apr  8 17:58 minit
-rwxr-xr-x  1 leitner  users    6120 Apr  8 17:58 msvc
$ ps awux | grep init
root       1  0.0 0.0   44  44 ?     S 21:16 0:03 /sbin/init
root   28907  0.0 0.1 3316 488 pts/2 S+23:40 0:00 grep init
$
```

Working Around Daemons that Background Themselves

Most daemons have a way to tell them not to background. If not, they should be fixed. Normally it's quite easy to do that.

Another ugly hack would be to write a small shared library implementing `daemon()` as a no-op and LD_PRELOADing it (BSD legacy code only).

daemontools comes with `fghack`, which creates a pipe, then runs the daemon, and reads from the pipe. If the daemon does not close file descriptors 4 to 33 for some bizarre voodoo reason, the read will block until the daemon terminates. This way you don't get the PID of the daemon (needed for sending signals), but at least you can restart it.

Working Around Daemons that Background Themselves

minit comes with pidfilehack.

pidfilehack runs the program and watches the pid file (given as command line argument).

Once the file appears, the contents of it are reported back to minit as the real PID of the daemon. minit checks with kill(PID,0) that the daemon is still running with that PID.

Future Directions

Currently, the `sync` mechanism is quite rudimentary. Basically minit calls `waitpid()` on the PID of the service if it is marked `sync`.

That basically means that two sync services can not run in parallel. And you cannot use `msvc` while minit waits for a sync service to finish. In short, it sucks.

Some crazy finn sent me a patch to fix this, but I didn't understand it immediately and thus put it on hold so far (*blush*).

More Future Directions

Someone sent me a really ugly kludge patch that just tells minit to exec itself. The patch even does some really basic state synchronization that breaks horribly if the size of the structs changed in between.

However, the hack value of having this convinced me to include the patch. Plus, it was really small (only 200 bytes or so).

You know what? Just forget that I even mentioned this.

In practice minit works quite well and is in production use at a few sites.

Finally! The Last Slide!

Thanks for listening!

For the slow writers: `http://www.fefe.de/minit/`

Any questions?