Supervising a boot-time service using the *screen* screen manager

Edward McGuire

April 12, 2023

Abstract

This paper shows how a boot-time service can be started with the following properties:

- respawns in case of failure
- has a virtual terminal
- virtual terminal can be attached to

Contents

1 Introduction **2**

2 Starting a service supervised by *screen* **2**

3 Arranging for the service to start at boot time **3**

4 Working with the session on an operator terminal **3**

5 Discussion **3**

1 Introduction

Suppose you start a service at multiuser boot time. Normally such a service runs as a detached process. An operator cannot interact with it, except by signals and logfiles.

But what if you wanted to interact with it? Terminal multiplexers such as `screen` and `tmux` make this possible.

In this scenario, the executable file `/usr/local/bin/foo` is the service. The `screen` terminal multiplexer provides a virtual terminal. The `init` process supervisor starts, and if necessary, restarts, the service.

This was tested using Slackware 15.0.

2 Starting a service supervised by *screen*

The command needed to start the service is:

```
/usr/bin/screen -D -m -S foo /usr/local/bin/foo
```

Let's break this down.

- The `-m` option makes a session manager.
- The `-D` option, in combination with `-m`, configures it as a detached session, but leaves the session manager running in the foreground. That is, the command does not return unless the session gets killed. This dovetails nicely with the `respawn` feature of `inittab`. If the service dies, then `screen` kills its window. Because it is the only window, `screen` then exits. And that is the input that the `respawn` feature is watching for.
- The `-S foo` option gives the session a recognizable name. This will be useful later when the operator wants to attach to it.
- the `/usr/local/bin/foo` argument is the name of an executable file. `screen` will create a window for it, and run it. The executable file should not detach (fork and exit). It should run as a foreground process that only exits in case of fatal error.

3 Arranging for the service to start at boot time

A `screen` session manager started as above can be monitored by `init`:

```
fu:3:respawn:env HOME=/root /usr/bin/screen -D -m -S foo /usr/local/bin/foo
```

Again, let's break this down.

- `fu` names the service for `init` – see the manual.
- `3` identifies the runlevel at which it should start. (If you have set your default runlevel to 4, use that instead.)
- `respawn` tells `init` to monitor the command. `init` will re-run the command when it dies.
- `env HOME=/root` adds `HOME` to the environment. This tells `screen` to create its socket under `/root/.screen`. If you omit this, it creates its socket under `/.screen`. This creates a problem when you try to attach to it.
- `/usr/bin/screen -D -m -S foo /usr/local/bin/foo` is just the command discussed in the previous section.

4 Working with the session on an operator terminal

`screen -ls` shows the sessionid.

`screen -r sessionid` attachs to the session.

`Ctrl+a d` or `Ctrl+a Ctrl+d` detaches from the session.

Naturally you will need to be superuser for any of this to work.

5 Discussion

Some things to keep in mind.

When the service dies, the `screen` session exits and `init` respawns one. You will not be able to attach and see the last error message.

As explained in the `init` manual, if the screen process flaps – defined as respawning more than 10 times in 2 minutes – init will suspend respawning for 5 minutes.

Because `init` runs as superuser, the `screen` and service do too. You could use `su`, `sudo`, or `daemon` to run it differently. That is left as an exercise.