Runit

Warning: Arch Linux only has official support for **systemd**. When using runit as your init system, please mention so in support requests.

Related articles

init

Runit is a process supervisor. It includes `runit-init`, which can replace sysv's init as pid1, or can be run from inittab or your init system of choice. Runit's simple collection of tools can be used to build flexible dependency structures and distributed systems, or blazing fast parallel runlevel changes (including the initial boot). Runit can be used as a simple process supervisor, see the **#User Services** for an example.

Contents

Installation

Using runit alongside systemd

Using BusyBox's implementation

BusyBox provides a minimal implementation of runit that can be used for simple processing supervision needs. First, create symbolic links to the BusyBox binary for the necessary tools that are going to be needed:

```
# busybox --list | awk '/^runsv|^chpst$|^sv/' | xargs -I{} ln -sv /usr/bin/busybox /usr/local/bin/{}
```

Afterwards, a **systemd** unit file can be created in order to run BusyBox's runit when needed:

```
/etc/systemd/system/busybox-runit.service
```

```
[Unit]
Description=Runit service supervision - BusyBox implementation
Documentation=man:busybox(1) http://smarden.org/runit/

[Service]
```

```
Environment="PATH=/usr/local/sbin:/usr/local/bin:/usr/bin"
ExecStart=/usr/local/bin/runsvdir -P /var/service
KillSignal=SIGHUP
KillMode=process
Restart=always
SuccessExitStatus=111

[Install]
WantedBy=multi-user.target
```

```
${XDG_CONFIG_HOME:-$HOME/.config}/systemd/user/busybox-runit.service
```

```
[Unit]
Description=Runit service supervision - BusyBox implementation
Documentation=man:busybox(1) http://smarden.org/runit/

[Service]
Environment="HOME=%h" "PATH=/usr/local/sbin:/usr/local/bin:/usr/bin"
ExecStart=/usr/local/bin/runsvdir -P %h/service
KillSignal=SIGHUP
KillMode=process
Restart=always
SuccessExitStatus=111

[Install]
WantedBy=default.target
```

Note:

- This example unit file presupposes that the directory which is going to contain the enabled services is `/var/service` for the system instance or `$HOME/service` for user instance. This path can be changed according to each specific use case.
- The `SIGHUP` kill signal is used instead of the default `SIGTERM`, and only on the main *runsvdir* process (thanks to `KillMode=process`) so that processes being controlled by BusyBox's runit implementation are controllably stopped by *runsvdir* before terminating the supervisor. When *runsvdir* ends the processes that are being supervised after receiving a `SIGHUP` signal, it exits with an status code of 111, which needs to be interpreted as a success.

Be sure to create the directory which is going to be supervised by *runsvdir* according to the systemd unit file created. It is also recommended to create a directory in which runit services can be stored (usually `/etc/sv`), and only enabled when needed by creating a symbolic link directed to them from the directory being supervised. See **#General use** for more details.

When everything is correctly configured, `busybox-runit.service` can be **enabled and started**.

Using standard runit

It is possible to use runit as a simple process supervisor alongside the default Arch Linux's init system (**systemd**). For this purpose, install `runit-systemd (https://aur.archlinux.org/packages/runit-systemd/)`AUR, which provides a barebones runit installation without any stage scripts (`/etc/runit/{1..3}`) or runlevels (`/etc/runit/runsvdir/*`), which are generally only useful when using runit as the init system. The package provides a directory (`/var/service`) in which the desired runit services can be put and a systemd unit which starts runit monitoring that directory. Only the services configured in `/var/service` will be supervised by runit. Just **enable and start** `runit.service`.

Usage

The tools

- `sv` - used for controlling services, getting status of services, and dependency checking.
- `chpst` - control of a process environment, including memory caps, limits on cores, data segments, environments, user/group privileges, and more.
- `runsv` - supervises a process, and optionally a log service for that process.
- `svlogd` - a simple but powerful logger, includes auto-rotation based on different methods (time, size, etc), post-processing, pattern matching, and socket (remote logging) options. Say goodbye to logrotate and the need to stop your services to rotate logs.
- `runsvchdir` - changes service levels (runlevels, see below).
- `runsvdir` - starts a supervision tree

See the manpages for usage details not covered below.

Run levels and service directories

Runit uses directories of symlinks to specify runlevels, other than the 3 main ones, which are defined in /etc/runit/1, 2, and 3.

1 bootstraps the system, 2 starts runsvdir on /service, and 3 stops the system.

While in run level 2, you are not constrained to any amount of service levels (equivalent to runlevels in sysvinit). You can runschdir to any directory (full of service directory symlinks) you've made in /etc/runit/runsvdir/. This becomes very handy in cases where you have an HA (Failover) setup, and you have one machine that can take over services for many other machines, simply by runsvchdir <theservicedir>.

You can also run trees of dependent service levels by having user-level supervision directories. See User Level Services below.

By default, the runit-run package uses a very minimal service set, defined in /etc/runit/runsvdir/archlinux-default and symlinked to /etc/runit/runsvdir/default.

It only gives gettys on tty2 and tty3, so you will boot to just console scroll and a tidy 'runsvchdir: default: current'. This means when you start X it will be on tty4.

To go back to the standard arch consoles, remove the link /service/ngetty and link as many /etc/sv/*getty* services you like in /service, or edit the /etc/sv/ngetty/run file to get more getties. Better yet, create your own directory in /etc/runit/runsvdir and add the symlinks you want for just the services you desire. Remember to take any services you start with runit out of DAEMONS in /etc/rc.conf or systemctl disable them, they do not need to be started there, and runit will allow parallel startup without backgrounding them.

General use

In this explanation, `/var/service` is the chosen service directory being supervised by *runsvdir* and `/etc/sv` is the chosen directory for containing the services that can be enabled.

Tip: Specifying the whole path to the service directory can be avoided by setting the environment variable `SVDIR` to indicate the path of the service directory. For example, with `SVDIR=/var/service`, `sv status /var/service/`*`servicename`* becomes `sv status` *`servicename`*.

- Listing running services:

```
# sv status /var/service/*
run: /var/service/agetty-2: (pid 4120) 7998s
run: /var/service/agetty-3: (pid 4119) 7998s
run: /var/service/bougyman: (pid 4465) 7972s
run: /var/service/bougyx: (pid 4135) 7998s; run: log: (pid 4127) 7998s
run: /var/service/cron: (pid 4137) 7998s; run: log: (pid 4122) 7998s
run: /var/service/dialer: (pid 4121) 7998s
run: /var/service/qmail: (pid 4138) 7998s; run: log: (pid 4126) 7998s
run: /var/service/smtpd: (pid 4136) 7998s; run: log: (pid 4125) 7998s
run: /var/service/socklog-klog: (pid 4139) 7998s; run: log: (pid 4132) 7998s
run: /var/service/socklog-unix: (pid 4133) 7998s; run: log: (pid 4124) 7998s
run: /var/service/ssh: (pid 4134) 7998s; run: log: (pid 4123) 7998s
```

- Create and start a service:

```
# ln -s /etc/sv/ssh /var/service/ssh
```

- Stops a service immediately (would still start on next boot):

```
# sv down /var/service/ssh
```

- Starts a service which has been previously stopped, or which is configured to not start automatically:

```
# sv up /var/service/ssh
```

- Restarts a service:

```
# sv restart /var/service/ssh
```

- Reloads a service:

```
# sv reload /var/service/ssh
```

- Shows status of a service and it's log service:

```
# sv status /var/service/ssh
```

- Stops a service, and disables it (won't start next boot):

```
# rm /var/service/ssh
```

User Services

There are two ways of creating a user supervision tree: using a Systemd/User service or with runit itself. See #Using BusyBox's implementation for an example systemd user service. To use runit itself refer to the section in the Voidlinux Handbook (https://docs.voidlinux.org/config/services/user-services.html).

Example Service

Note: This example assumes your services are stored in `~/.condig/sv` and your runsvdir is `~/service` . You should change them to your configuration.

Create a directory where you will keep your services.

```
$ mkdir -p ~/.config/sv
```

Create a directory for your service.

```
$ mkdir ~/.config/sv/mpd
```

Make a `run` file in the service directory and make it executable.

```
$ touch ~/.config/sv/mpd/run && chmod +x ~/.config/sv/mpd/run
```

Edit the `run` file. You can find example run files in runit's site (http://smarden.org/runit/runscripts.html)

$HOME/.config/sv/mpd/run

```
#!/bin/sh
MPDCONF=${XDG_CONFIG_HOME:-$HOME/.config}/mpd/mpd.conf
exec mpd --no-daemon $MPDCONF
```

To enable the service make a symlink to your service directory. This will make the service start automatically when the runit starts. The service should start immediately.

```
$ ln -s ~/.config/sv/mpd ~/service/mpd
```

To stop the service:

```
$ SVDIR=~/service sv down mpd   # or
$ sv down ~/service/mpd
```

To start the service again:

```
$ SVDIR=~/service sv up mpd   # or
$ sv up ~/service/mpd
```

See also

- Runit's Homepage (http://smarden.org/runit/). G Pape's runit homepage.
- Running runit on Amazon Linux AMI (https://evasive.ru/50a3904206c52447aa1fa5d90a8382a3.html). Contains instructions related to setting up BusyBox's implementation of runit.
- Voidlinux Handbook - Services and Daemons - runit (https://docs.voidlinux.org/config/services/index.html). The section in the Voidlinux Handbook about runit.

Retrieved from "https://wiki.archlinux.org/index.php?title=Runit&oldid=655507"