spawnd

Marc Huber

COLLABORATORS			
	TITLE : spawnd		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Marc Huber	December 14, 2024	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

1 Introduction

spawnd is a broker with load-balancing functionality that listens for incoming TCP (or SCTP) connections on IP, UNIX or possibly IPv6 sockets, accepts them and finally forwards them (using ancillary messages over UNIX domain sockets) to the spawned worker processes.

Support for receiving and forwarding UDP packets to the worker processes is available, too (**tac_plus-ng** uses that for processing RADIUS/UDP)

1.1 Download

You can download the source code from the GitHub repository at https://github.com/MarcJHuber/event-driven-servers/. On-line documentation is available via https://projects.pro-bono-publico.de/event-driven-servers/doc/, too.

2 Operation

spawnd is now actually implemented as a shared library, and the programs that had to be invoked by it are now utilizing that library and are, as such, standalone. This in no way implies that **spawnd** configuration would be obsolete; only the binary is.

2.1 Command line syntax

Command line syntax is:

```
spawnd [ -b | -f ] [ -p pidfile-name ] [ -P ] [ -d level ] configuration-file [ id ]
```

The path to the configuration file is the only command line argument mandatory. If compiled with CURL support, *configuration-file* may be an URL.

id defaults to `spawnd`. It may be used to select a non-default section of the configuration file.

The `-b` switch will tell the daemon to release its controlling terminal on startup and fork itself to the background (just like `background = yes` in the configuration does, but with higher precedence). Likewise, `-f` keeps the daemon from forking to the background.

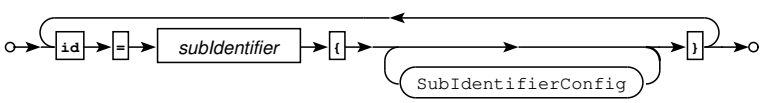
The `-p` *pidfile-name* option is equivalent to the `pidfile =` *pidfile-name* configuration directive.

The `-P` option enables *config parse mode*. Keep this one in mind; it is imperative that the configuration file supplied is syntactically correct, as the daemon won't start if there are any parsing errors at start-up.

The `-d` switch enables debugging. You most likely don't want to use this. Read the source if you need to.

3 Configuration file syntax

A typical **spawnd** configuration file consists of multiple `id` sections: one for **spawnd** itself, and one for the spawned server process (e.g. **tac_plus** or **ftpd**. The actual configuration section used is, by default, the one named after the program evaluating the configuration file. However, a different section may be selected by specifying an `id` parameter via command line or **spawnd** directive.



Railroad diagram: Config

For example, have a look at the following configuration snippet:

```
id = spawnd { exec = /path/to/ftpd }
id = spawnd2 { exec = /path/to/ftpd id = myftpd }
id = myftpd { }
id = ftpd { }
```

A **spawnd** started with this will default to evaluating the `id = spawnd` section, and any **ftpd** instance started will default to the corresponding `ftpd` stanza. However, starting **spawnd** with an additional argument, e.g.

```
/path/to/spawnd /path/to/configuration_file spawnd2
```

will choose the `spawnd2` section instead, which in turn tells the **ftpd** to evaluate the `myftpd` part of the configuration.

Comments within configuration files start with `#`. At top-level, other configuration files may be included using `include =` *file* syntax. Glob pattern matching applies (typical `sh`(1) wildcards are evaluated).

All the configuration directives given below need to be enclosed in an appropriate `id { ... }` section.

- (`permit` | `deny`) [`not`] *cidr*

  Accept or reject request from specific IP address ranges. This directive may appear multiple times. Matches are tried in order. IPv6 ACLs are supported. Default is to accept everything.

  Example:

  ```
  permit 127.0.0.1/8
  deny 192.168.5.0/8
  permit 192.168.0.0/16
  acl accept ::1
  ```

- `background =` (`yes` | `no`)

  If set, the daemon will release its controlling terminal on startup and fork itself to the background (default: `no`).

- `listen { ... }`

  This directives determines the connection end points the daemon is listening on. For IP, valid configuration directives inside the curly brackets are:

  – `port =` *Port*
  Port can be either numerical or a service name.

  – `flag =` (`access` | `accounting`)
  RADIUS/UDP supports Status Server queries. If you want to use these reliably you should use a dedicated UDP port for RADIUS accounting packets and set `flag = accounting`. The `accounting` flag will be forwarded to and evaluated by **tac_plus-ng** to choose the correct Server Status response.

  – `address =` *IPAddress*
  This is optional; by default the daemon listens on all available IP addresses, both v4 and v6.

  – `protocol =` (`TCP` | `UDP` | `SCTP`)
  Default protocol is TCP.

  – `bind retry delay =` *Seconds*
  On *bind(2)* failure, wait the specified number of *Seconds*, then try again. Default: 0 seconds (no retries).

  For listening on UNIX domain sockets, a different syntax applies:

  – `path =` *Path*
  This directive specifies the path to the UNIX domain socket.

  – `mode =` *Mode*
  This sets the file creation mode.

  – `userid =` *UserID*
  This specifies the user ID for socket creation.

– `groupid` = *GroupID*
  This specifies the group ID for socket creation.

Directives which may be used in both cases are:

– `tls` = (`yes` | `no`)
  This directive tells a child process if the connection is encrypted using TLS.
– `haproxy` = (`yes` | `no`)
  This directive tells a child process whether the connection is permitted to use the *haproxy* protocol (currently **tac_plus-ng** only).
– `backlog` = *Number*
  This sets the maximum number of pending connections (default: 128); see `listen`(2) for details.
– `overload backlog` = *Number*
  This sets the maximum number of pending connections in overload situations (default: 128); see `listen`(2) for details.
– `realm` = *String*
  Sends *String* to the client process when forwarding a connection. The client process may be able to use this value to differentiate between connection endpoints. Use this option only with clients that actually support it (currently: `tac_plus`).
– `vrf` = *id*
  Sets the VRF to *id* on systems that support it. *id* is the VRF name on Linux, or the numeric VRF id on OpenBSD.
– `bind retry delay` = *Seconds*
  On `bind`(2) failure, wait the specified number of *Seconds*, then try again. Defaults to the global `bind retry delay` value.
– `tcp keepalive` (`count` | `idle` | `interval`) = *Number*
  Sets various options for TCP keepalive probes, if supported by the operating system.
– `tcp bufsize` *Number*
  Overrides the system default input/output buffer sizes (SO_SNDBUF, SO_RCVBUF) for communication with child processes.

The *listen* directive is mandatory unless the daemon is started via an `inetd`(8) (or compatible) process, in which case

– the *inetd* `wait` option needs to be used
– `argv[0]` needs to be the absolute path to the binary

For standard `inetd`(8), configuration syntax for `ftpd` or any other `spawnd` compliant application would look like:

```
ftp  stream  tcp  wait root /usr/local/sbin/ftpd /usr/local/sbin/ftpd /usr/local/etc/ftpd. ↩
    cfg
```

or, with explicit specification of the spawned program's name in the configuration,

```
ftp  stream  tcp  wait root /usr/local/sbin/spawnd ftpd -f /usr/local/etc/ftpd.cfg
```

The equivalent `xinetd`(8) would (or could) be:

```
service ftp
{
    flags       = NAMEINARGS NOLIBWRAP
    socket_type = stream
    protocol    = tcp
    wait        = yes
    user        = root
    server      = /usr/local/sbin/ftpd
    server_args = /usr/local/sbin/ftpd -f /usr/local/etc/ftpd.cfg
    instances   = 1
}
```

- `pidfile` = *file*
  The process id will be written to *file*.
- `spawn { ... }`
  The `spawn` section defines various aspects related to the actual server processes:
  - `exec` = Path
    Defines the path the server process. This is mandatory when running the standalone **spawnd** process, but may be omitted else.
    Magic cookie substitution applies. The available conversions are
    * `%o` - run-time OS type
    * `%O` - compile-time OS type
    The "OS type" string inserted is identical to the output of:
    ```
    uname -srm | tr ' [:upper:]' '\-[:lower:]'
    ```
    Example: For "Linux 2.3.35 i686",
    ```
    exec = /some/where/%O/ftpd
    ```
    resolves to
    ```
    exec = /some/where/linux-2.3.35-i686/ftpd
    ```
  - `id` = ID
    Optionally defines a different ID for configuration file parsing. Defaults to the executables basename.
  - `config` = ConfigurationFile
    Optionally assigns a configuration file. Defaults to the configuration file **spawnd** is started with.
  - `instances` (`min` | `max`) = Number
    Sets the minimum or maximum number of server processes to start. Defaults to 2 and 8.
  - `sticky cache period` = Seconds
    This option tells the daemon to try to forward all connections from a particular source address to the same worker process. Defaults to 0 (disabled).
  - `sticky cache size` = Number
    This option sets the maximum number of entries in the "sticky" cache. Defaults to 1024.
  - `users` (`min` | `max`) = Number
    This directive limits the number of users per process. The distribution algorithm attempts to assign at least *min* (default: 5) and at most *max* (default: 40) users to each process, while attempting to keep the total number of processes at a reasonable limit.
    The spawned processes may have their own idea about the maximum number of users permitted and may lower the specified maximum number of users to a more suitable value.
  - `userid` = UserID
    Change UID to *UserID* for spawned processes.
  - `groupid` = GroupID
    Change GID to *GroupID* for spawned processes.
  - `working directory` = Directory
    Change directory to *Directory* for spawned processes.
  - `ipc key` = Number
    If this is set and the program was compiled with IPC support, then the configuration file will be cached in a shared memory segment and will only be loaded once. This may be of advantage if the configuration file given as an URL that will be retrieved using CURL.
- `overload` = (`close` | `queue` | `reset`)
  If the maximum number of users is reached, either close, reset or queue new connections. The latter is the default.

- `tcp keepalive` (`count` | `idle` | `interval`) = Number

  Sets various options for TCP keepalive probes, if supported by the operating system.

- `syslog` ((`ident` = Ident) | (`severity` = Level) | (`facility` = Facility))

  Selects syslog *ident*, *severity* and *facility*. Defaults to:

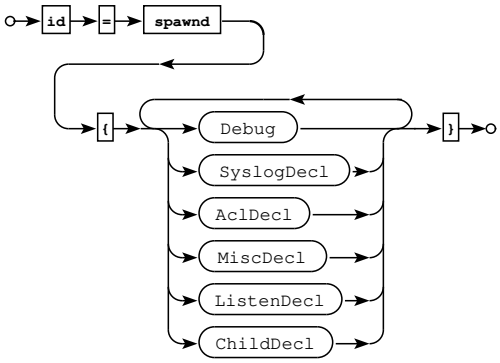
```
syslog ident = program-name
syslog facility = UUCP
syslog severity = INFO
```

- `single process` = (`yes` | `no`)

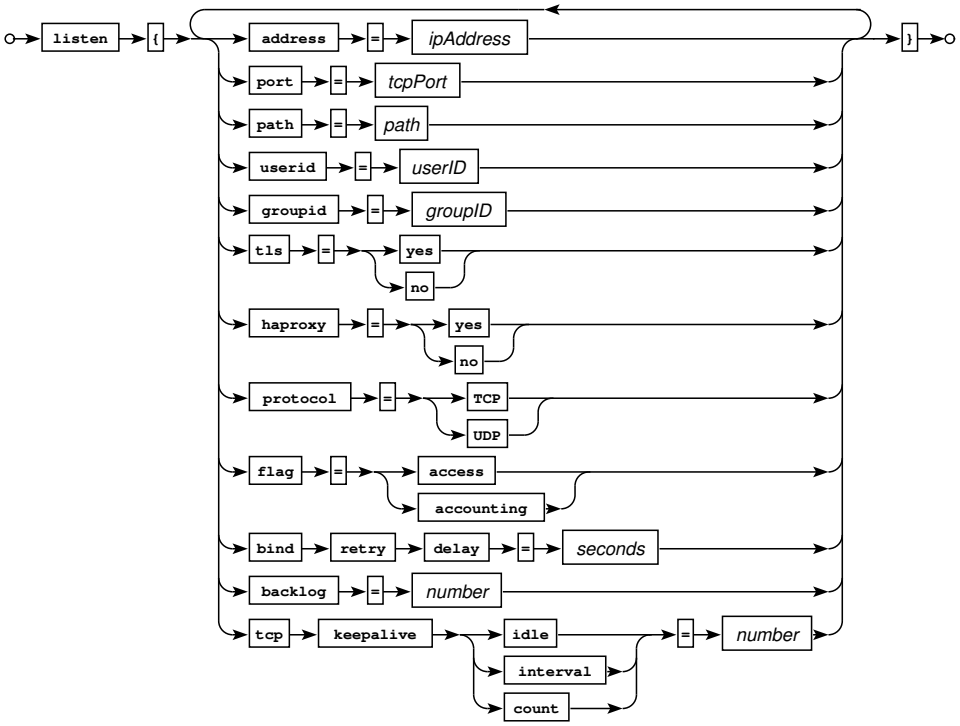
  Changes the execution model to *single process* mode. Connections will be accepted and processed by one single instance of the process, and not, as it's the default, be forwarded to child processes. Useful for systems that lack file descriptor passing capabilities.

  Default: `yes` (and not changeable) on Cygwin, `no` everywhere else.

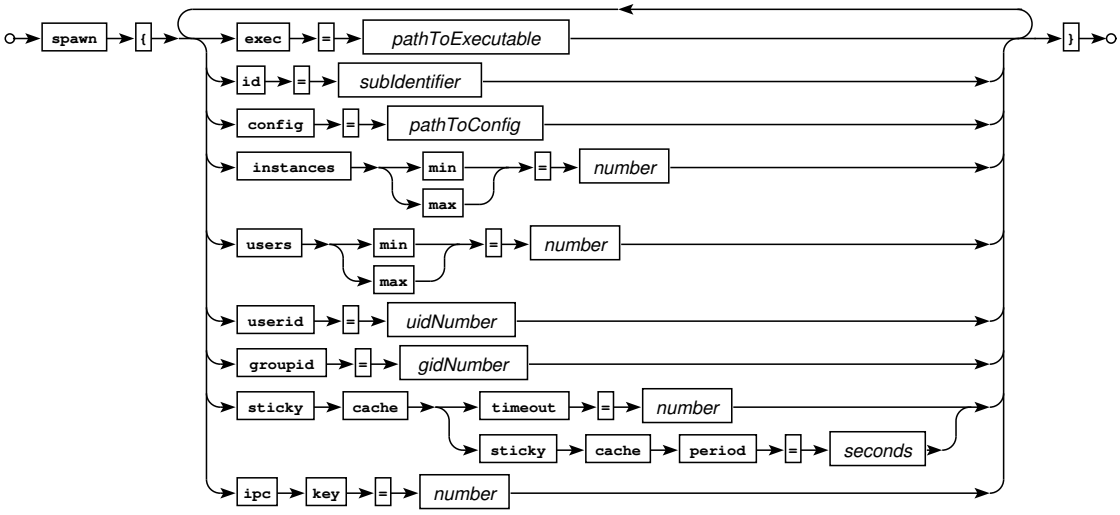
3.1 Railroad Diagrams



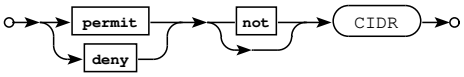
Railroad diagram: SpawndConfig



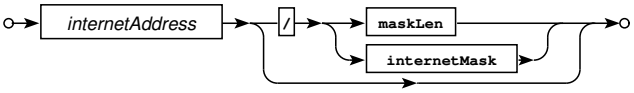
Railroad diagram: ListenDecl



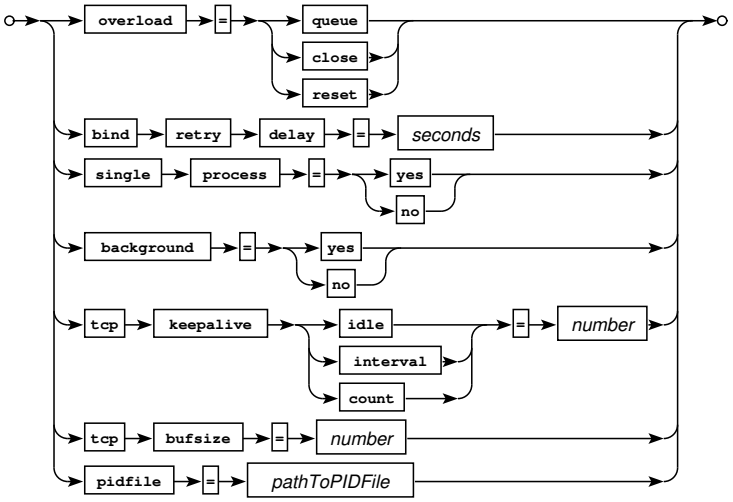
Railroad diagram: Child



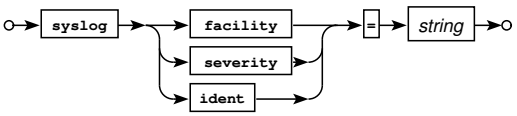
Railroad diagram: AclDecl



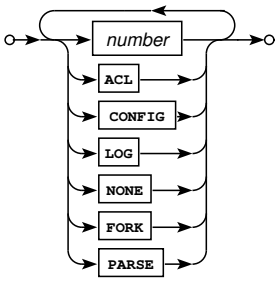
Railroad diagram: CIDR



Railroad diagram: MiscDecl



Railroad diagram: SyslogDecl



Railroad diagram: Debug

4 Signals

spawnd will terminate upon receiving a SIGTERM or SIGINT signal. SIGHUP will cause **spawnd** to restart itself from scratch.

The daemon is only checking for signals every couple of seconds, so actions aren't necessarily immediate.

5 Load balancing algorithm

spawnd allows configuration of upper and lower limits for users and processes. The distribution algorithm will try to assign new connections to one of the running servers with less than *users_min* connections. If all servers already have at least *users_min* active connections and the total number of servers doesn't exceed *servers_max*, an additional server process is started, and the connection is assigned to that process. If no more processes may be started, the connection is assigned to the server process with less than *users_max* users, which serves the lowest number of connections. Otherwise, the connection will stall until an existing connection terminates.

If the *sticky* feature is enabled, **spawnd** will try to assign connections to server processes based on the remote IP address of the peer. Please not that this will not work in combination with HAProxy.

6 Event mechanism selection

Several level-triggered event mechanisms are supported. By default, the one best suited for your operating system will be used. However, you may use the environment variable `IO_POLL_MECHANISM` to select a specific one.

The following event mechanisms are supported (in order of preference):

- port (Sun Solaris 10 and higher only, `IO_POLL_MECHANISM=32`)
- kqueue (*BSD and Darwin only, `IO_POLL_MECHANISM=1`)
- /dev/poll (Sun Solaris only, `IO_POLL_MECHANISM=2`)
- epoll (Linux only, `IO_POLL_MECHANISM=4`)
- poll (`IO_POLL_MECHANISM=8`)
- select (`IO_POLL_MECHANISM=16`)

Environment variables can be set in the configuration file at top-level:

```
setenv IO_POLL_MECHANISM = 4
```

7 Sample configuration

```
id = spawnd {
    listen { port 21 }
    listen { address = ::0 port = 2121 tls }
    spawn {
        users minimum = 10
        users maximum = 100
        instances minimum = 10
        instances maximum = 100
        exec = /usr/local/libexec/ftpd
        id = ftpd
        config = /usr/local/etc/ftpd.conf
    }
    background = true
}
```

8 Startup examples

spawnd (either standalone, or utilized via the `MAVIS` library, which is what **ftpd**, **tac_plus** and **tcprelay** do) is a long-running process. It may be started either manually, on demand, or at system boot time.

The examples in this section focus on **tac_plus**, but are easily adaptable to the **ftpd** and **tcprelay** daemons.

8.1 Manual startup

Starting a daemon manually is fine for testing, but, generally, undesireable for production. A configuration file that specifies at least a port the daemon should listen to is required:

```
id = spawnd {
    listen {
        port = 49
    }
}
id = tac_plus {
  ...
}
```

Copy this to, e.g., `./tac_plus.cfg`, then start the daemon:

```
# /usr/local/sbin/tac/plus ./tac_plus.cfg
```

The daemon will now run in the foreground, blocking your shell until interrupted or being send to the background. If you want to run the daemon in the background, you can either add

```
    background = yes
```

to the `spawnd` section, or use the `-b` command line option:

```
# /usr/local/sbin/tac/plus -b /usr/local/etc/tac_plus.cfg
```

8.2 Startup on demand

The daemon may be started on demand by `inetd`(8) or compatible applications. The configuration file should **not** specify a port to bind to, as `inetd` will pass an already bound socket to the daemon:

```
id = spawnd {
    listen {
    }
}
id = tac_plus {
  ...
}
```

8.2.1 inetd

For stock `inetd`, adding the following line to `/etc/inetd.conf` and sending a `HUP` to `inetd` will activate the daemon:

```
tacacs  stream  tcp  wait root /usr/local/sbin/tac_plus /usr/local/sbin/tac_plus /usr/local ↩
    /etc/tac_plus.cfg
```

8.2.2 xinetd

The equivalent `xinetd`(8) configuration:

```
service tacacs
{
    flags       = NAMEINARGS NOLIBWRAP
    socket_type = stream
    protocol    = tcp
    wait        = yes
    user        = root
    server      = /usr/local/sbin/tac_plus
    server_args = /usr/local/sbin/tac_plus /usr/local/etc/tac_plus.cfg
    instances   = 1
}
```

Depending on your setup this could either be added to `/etc/xinetd.conf` or be written to `/etc/xinetd.d/tacacs`.

8.2.3 launchd

Mac OS X comes with `launchd`(8), and here's a suitable `/Library/LaunchDaemons/de.pro-bono-publico.tac_plus.`

```
<xml version="1.0" encoding="UTF-8"?>
<DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/ ↩
    PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>Label<key>
    <string>de.pro-bono-publico.tac_plus<string>
    <key>ProgramArguments<key>
    <array>
        <string>/usr/local/sbin/tac_plus<string>
        <string>-p<string>
        <string>/var/run/tac_plus.pid<string>
        <string>/usr/local/etc/tac_plus.cfg<string>
    <array>
    <key>KeepAlive<key> <true/>
    <key>Sockets<key>
        <dict>
            <key>Listeners<key>
            <dict> <key>SockServiceName<key> <string>tacacs<string> <dict>
        <dict>
```

```
    <key>inetdCompatibility<key>
        <dict> <key>Wait<key> <true/> <dict>
    <key>KeepAlive<key>
        <dict> <key>NetworkState<key> <true/> <dict>
<dict>
<plist>
```

This needs to be activated using

```
# sudo launchctl load -w /Library/LaunchDaemons/de.pro-bono-publico.tac_plus.plist
```

The daemon will write its process id to `/var/run/tac_plus.pid`, and

```
# sudo kill `cat /var/run/tac_plus.pid`
```

will cause it to restart (and, implicitly, to re-read the configuration file).

8.3 Startup at system boot

The daemons may be started at system boot time. Alas, that's very specific to your system. You should definitely know what you're doing, or you may render your system unbootable.

8.3.1 Init scripts

The distribution comes with a couple of System V style init scripts, e.g. `tac_plus/doc/etc_init.d_tac_plus`. Copy this script to a location appropriate to your system (e.g. `/etc/init.d/tac_plus` and create the relevant symbolic or hardlinks. See your systems documentation for details.

8.3.2 launchd

On MacOS, create a file `/Library/LaunchDaemons/de.pro-bono-publico.tac_plus.plist` that consists of:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/ ↩
    PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>Label</key>
    <string>de.pro-bono-publico.tac_plus</string>
    <key>ProgramArguments</key>
    <array>
        <string>/usr/local/sbin/tac_plus</string>
        <string>-f</string>
        <string>-p</string>
        <string>/var/run/tac_plus.pid</string>
        <string>/usr/local/etc/tac_plus.cfg</string>
    </array>
    <key>KeepAlive</key>
        <dict> <key>NetworkState</key> <true/> </dict>
</dict>
</plist>
```

Then tell `launchd` about this configuration:

```
# sudo launchctl load -w /Library/LaunchDaemons/de.pro-bono-publico.tac_plus.plist
```

The daemon will write its process id to `/var/run/tac_plus.pid`, and you may make it re-read its configuration file by issuing

```
# sudo kill -HUP `cat /var/run/tac_plus.pid`
```

8.3.3 systemd

For `systemd` you'll have to create an appropriate *configuration unit*. Copy

```
[Unit]
Description=TACACS+ Service
After=syslog.target

[Service]
ExecStart=/usr/local/sbin/tac_plus -f /usr/local/etc/tac_plus.cfg
KillMode=process
Restart=always
ExecReload=/bin/kill -HUP $MAINPID

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

to `/etc/systemd/system/tac_plus.service`, then enable and start the service:

```
# sudo systemctl enable tac_plus.service
# sudo systemctl start tac_plus.service
```

9 Copyrights and Acknowledgements

Please see the source for copyright and licensing information of individual files.

- **Portions of the parsing code are taken from Cisco's tac_plus developers kit which is distributed under the following license:**

  Copyright (c) 1995-1998 by Cisco systems, Inc.

  Permission to use, copy, modify, and distribute this software for any purpose and without fee is hereby granted, provided that this copyright and permission notice appear on all copies of the software and supporting documentation, the name of Cisco Systems, Inc. not be used in advertising or publicity pertaining to distribution of the program without specific prior permission, and notice be given in supporting documentation that modification, copying and distribution is by permission of Cisco Systems, Inc.

  Cisco Systems, Inc. makes no representations about the suitability of this software for any purpose. THIS SOFTWARE IS PROVIDED ``AS IS'' AND WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

- **The code written by Marc Huber is distributed under the following license:**

  Copyright (C) 1999-2022 Marc Huber (Marc.Huber@web.de). All rights reserved.

  Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

  1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
  2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
  3. The end-user documentation included with the redistribution, if any, must include the following acknowledgment:

     This product includes software developed by Marc Huber (Marc.Huber@web.de).

     Alternately, this acknowledgment may appear in the software itself, if and wherever such third-party acknowledgments normally appear.

  THIS SOFTWARE IS PROVIDED ``AS IS'' AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL ITS AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT,