PROCSERV(1)

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
2.8.0	06/28/2019		

Contents

1 NAME

procServ - Process Server with Telnet Console and Log Access

2 SYNOPSIS

procServ [*OPTIONS*] -P *endpoint... command args...*

procServ [*OPTIONS*] *endpoint command args...*

3 DESCRIPTION

procServ(1) creates a run time environment for a command (e.g. a soft IOC). It forks a server run as a daemon into the background, which creates a child process running *command* with all remaining *args* from the command line. The server provides control access (stdin/stdout) to the child process console by offering a telnet connection at the specified *endpoint*(s).

An *endpoint* can either be a TCP server socket (specified by the port number) or a UNIX domain socket (where available). See ENDPOINT SPECIFICATION below for details. For security reasons, control access is restricted to connections from localhost (127.0.0.1), so that a prior login in to the host machine is required. (See **--allow** option.)

The first variant allows multiple *endpoint* declarations and treats all non-option arguments as the command line for the child process. The second variant (provided for backward compatibility) declares one *endpoint* with its specification taken from the first non-option argument.

procServ can be configured to write a console log of all in- and output of the child process into a file using the **-L** (**--logfile**) option. Sending the signal SIGHUP to the server will make it reopen the log file.

To facilitate running under a central console access management (like conserver), the **-l** (**--logport**) option creates an additional endpoint, which is by default public (i.e. TCP access is not restricted to connections from localhost), and provides read-only (log) access to the child's console. The **-r** (**--restrict**) option restricts both control and log access to connections from localhost.

Both control and log endpoints allow multiple connections, which are handled transparently: all input from control connections is forwarded to the child process, all output from the child is forwarded to all control and log connections (and written to the log file). All diagnostic messages from the procServ server process start with "`@@@`" to be clearly distinguishable from child process messages. A name specified by the **-n** (**--name**) option will replace the command string in many messages for increased readability.

The server will by default automatically respawn the child process when it dies. To avoid spinning, a minimum time between child process restarts is honored (default: 15 seconds, can be changed using the **--holdoff** option). This behavior can be toggled online using the toggle command `^T`, the default may be changed using the **--noautorestart** option. You can restart a running child manually by sending a signal to the child process using the kill command `^X`. With the child process being shut down, the server accepts two commands: `^R` or `^X` to restart the child, and `^Q` to quit the server. The **-w** (**--wait**) option starts the server in this shut down mode, waiting for a control connection to issue a manual start command to spawn the child.

To facilitate running under system daemon management (systemd/supervisord), the **-o** (**--oneshot**) option will exit the procServ server after the child exits. In that mode, the system daemon must handle restarts (if required), and all clients will have to reconnect.

Any connection (control or log) can be disconnected using the client's disconnect sequence. Control connections can also be disconnected by sending the logout command character that can be specified using the **-x** (**--logoutcmd**) option.

To block input characters that are potentially dangerous to the child (e.g. `^D` and `^C` on soft IOCs), the **-i** (**--ignore**) option can be used to specify characters that are silently ignored when coming from a control connection.

To facilitate being started and stopped as a standard system service, the **-p** (**--pidfile**) option tells the server to create a PID file containing the PID of the server process. The **-I** (**--info-file**) option writes a file listing the server PID and a list of all endpoints.

The **-d** (**--debug**) option runs the server in debug mode: the daemon process stays in the foreground, printing all regular log content plus additional debug messages to stdout.

4 ENDPOINT SPECIFICATION

Both control and log endpoints may be bound to either TCP or UNIX sockets (where supported). Allowed endpoint specifications are:

<port>
Bind to either 0.0.0.0:<*port*> (any) or 127.0.0.1:<*port*> (localhost) depending on the type of endpoint and the setting of **-r** (**--restrict**) and **--allow** options.

<ifaceaddr>:<port>
Bind to the specified interface address and <*port*>. The interface IP address <*ifaceaddr*> must be given in numeric form. Uses 127.0.0.1 (localhost) for security reasons unless the **--allow** option is also used.

unix:</path/to/socket>
Bind to a named unix domain socket that will be created at the specified absolute or relative path. The server process must have permission to create files in the enclosing directory. The socket file will be owned by the uid and primary gid of the procServ server process with permissions 0666 (equivalent to a TCP socket bound to localhost).

unix:<user>:<group>:<perm>:</path/to/socket>
Bind to a named unix domain socket that will be created at the specified absolute or relative path. The server process must have permission to create files in the enclosing directory. The socket file will be owned by the specified <*user*> and <*group*> with <*perm*> permissions. Any of <*user*>, <*group*>, and/or <*perm*> may be omitted. E.g. "-P unix::grp:0660:/run/procServ/foo/control" will create the named socket with 0660 permissions and allow the "grp" group connect to it. This requires that procServ be run as root or a member of "grp".

unix:@</path/to/socket>
Bind to an abstract unix domain socket (Linux specific). Abstract sockets do not exist on the filesystem, and have no permissions checks. They are functionally similar to a TCP socket bound to localhost, but identified with a name string instead of a port number.

5 OPTIONS

--allow
Allow TCP control connections from anywhere. (Default: restrict control access to connections from localhost.) Creates a serious security hole, as telnet clients from anywhere can connect to the child's stdin/stdout and might execute arbitrary commands on the host if the child permits. Needs to be enabled at compile-time (see Makefile). Please do not enable and use this option unless you exactly know why and what you are doing.

--autorestartcmd=*char*
Toggle auto restart flag when *char* is sent on a control connection. Use ^ to specify a control character, `""` to disable. Default is `^T`.

--coresize=*size*
Set the maximum *size* of core file. See getrlimit(2) documentation for details. Setting *size* to 0 will keep child from creating core files.

-c, --chdir=*dir*
Change directory to *dir* before starting the child. This is done each time the child is started to make sure symbolic links are properly resolved on child restart.

-d, --debug
Enter debug mode. Debug mode will keep the server process in the foreground and enables diagnostic messages that will be sent to the controlling terminal.

-e, --exec=*file*
Run *file* as executable for child. Default is *command*.

-f, --foreground
Keep the server process in the foreground and connected to the controlling terminal.

-h, --help
Print help message.

--holdoff=*n*
Wait at least *n* seconds between child restart attempts. (Default is 15 seconds.)

-i, --ignore=*chars*
Ignore all characters in *chars* on control connections. This can be used to shield the child process from input characters that are potentially dangerous, e.g. `^D` and `^C` characters that would shut down a soft IOC. Use `^` to specify control characters, `^^` to specify a single `^` character.

***-I, --info-file <file>**
Write instance information to this file.

-k, --killcmd=*char*
Kill the child process (child will be restarted automatically by default) when *char* is sent on a control connection. Use `^` to specify a control character, `""` for no kill command. Default is `^X`.

--killsig=*signal*
Kill the child using *signal* when receiving the kill command. Default is 9 (SIGKILL).

-l, --logport=*endpoint*
Provide read-only log access to the child's console on *endpoint*. See ENDPOINT SPECIFICATION above. By default, TCP log endpoints allow connections from anywhere. Use the **-r** (**--restrict**) option to restrict TCP access to local connections.

-L, --logfile=*file*
Write a console log of all in and output to *file*. - selects stdout.

--logstamp[=*fmt*]
Prefix lines in logs with a time stamp, setting the time stamp format string to *fmt*. Default is "[<timefmt>] ". (See **--timefmt** option.)

-n, --name=*title*
In all server messages, use *title* instead of the full command line to increase readability.

--noautorestart
Do not automatically restart child process on exit.

-o, --oneshot
Once the child process exits, also exit the server.

-P, --port=*endpoint*
Provide control access to the child's console on *endpoint*. See ENDPOINT SPECIFICATION above. By default, TCP control endpoints are restricted to local connections. Use the **--allow** option to allow TCP access from anywhere.

-p, --pidfile=*file*
Write the PID of the server process into *file*.

--timefmt=*fmt*
Set the format string used to print time stamps to *fmt*. Default is "%c". (See strftime(3) documentation for details.)

-q, --quiet
Do not write informational output (server). Avoids cluttering the screen when run as part of a system script.

--restrict
Restrict TCP access (control and log) to connections from localhost.

-V, --version
Print program version.

-w, --wait
Do not start the child immediately. Instead, wait for a control connection and a manual start command.

-x, --logoutcmd=*char*
Log out (close client connection) when *char* is sent on an control connection. Use ^ to specify a control character. Default is empty.

6 USAGE

To start a soft IOC using procServ, change the directory into the IOC's boot directory. A typical command line would be

```
procServ -n "My SoftIOC" -i ^D^C 20000 ./st.cmd
```

To connect to the IOC, log into the soft IOC's host and connect to port 20000 using

```
telnet localhost 20000
```

To connect from a remote machine, ssh to a user account on procservhost and connect to port 20000 using

```
ssh -t user@procservhost telnet localhost 20000
```

You will be connected to the soft IOCs console and receive an informative welcome message. All output from the procServ server will start with "@@@" to allow telling it apart from messages that your IOC sends.

```
> telnet localhost 20000
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
@@@ Welcome to the procServ process server (procServ Version 2.1.0)
@@@ Use ^X to kill the child, auto restart is ON, use ^T to toggle auto restart
@@@ procServ server PID: 21413
@@@ Startup directory: /projects/ctl/lange/epics/ioc/test314/iocBoot/iocexample
@@@ Child "My SoftIOC" started as: ./st.cmd
@@@ Child "My SoftIOC" PID: 21414
@@@ procServ server started at: Fri Apr 25 16:43:00 2008
@@@ Child "My SoftIOC" started at: Fri Apr 25 16:43:00 2008
@@@ 0 user(s) and 0 logger(s) connected (plus you)
```

Type the kill command character `^X` to reboot the soft IOC and get server messages about this action.

Type the telnet escape character `^]` to get back to a telnet prompt then "`quit`" to exit telnet (and ssh when you were connecting remotely).

Though procServ was originally intended to be an environment to run soft IOCs, an arbitrary process might be started as child. It provides an environment for any program that requires access to its console, while running in the background as a daemon, and keeping a log by writing a file or through a console access and logging facility (such as `conserver`).

7 ENVIRONMENT VARIABLES

PROCSERV_PID
Sets the file name to write the PID of the server process into. (See **-p** option.)

PROCSERV_DEBUG
If set, procServ starts in debug mode. (See **-d** option.)

8 KNOWN PROBLEMS

None so far.

9 REPORTING BUGS

Please report bugs using the issue tracker at https://github.com/ralphlange/procServ/issues.

10 AUTHORS

Originally written by David H. Thompson (ORNL). Current author: Ralph Lange <ralph.lange@gmx.de>.

11 RESOURCES

GitHub project: https://github.com/ralphlange/procServ

12 COPYING