PROCSERV(1)

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

1 NAME

procServ - Process Server with Telnet Console and Log Access

2 SYNOPSIS

procServ [*OPTIONS*] *port 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 console access (stdin/stdout) to the child process by offering a telnet connection at the specified port. For security reasons, access is restricted to connections from localhost (127.0.0.1), so that logging into a valid account on the host machine is required.

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

Both access and log ports allow multiple connections, which are handled transparently: all input from access connections is forwarded to the child process, all output from the child is forwarded to all access and log connections (and written to the log file). All diagnostic messages from the server process start with "`@@@ `" to be clearly distinguished 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 a child process when it dies. To avoid spinning, a minimum time between child process restarts is honoured (default: 15 seconds, can be changed using the **--holdoff** option). This behaviour 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` to restart the child and `^Q` to quit the server. The **-w** (**--wait**) option starts the server in this shut down mode, waiting for the telnet connection to issue a manual start command to create the child.

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 console access port.

To facilitate being started and stopped as a standard system service, the **-p** (**--pidfile**) option tells the server to create a standard PID file containing the PID of the server process.

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 OPTIONS

--allow
: Allow control connections from anywhere. (Default: restrict control access to localhost.) Creates a serious security hole, as telnet clients from anywhere can connect to the child's stdin/stdout and 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 an access 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 child. This is done each time the child is started to make sure symbolic links are 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.

-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 access 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.

-k, --killcmd=*char*
Kill the child process (child will be restarted automatically by default) when *char* is sent on an access 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=*port*
Provide read-only access to the child's console on *port*. By default all hosts can connect to *port*, use the **-r** (**--restrict**) option to restrict access to localhost.

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

-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.

-p, --pidfile=*file*
Write the PID of the server process into *file* to facilitate integration into regular system service administration mechanisms.

--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 log connections to localhost.

-V, --version
Print program version.

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

5 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 and quit to exit telnet (and ssh when you were connecting remotely).

While procServ was originally intended to be an environment to run soft IOCs, any process might be started as child. It provides an environment for any program that requires access to its console and should be run in the background as a daemon. By writing a file log directly or through a console access and logging facility (such as `conserver`), the log of such a program's output can be used to correlate its messages to other events. E.g., in an EPICS environment, running `casw` (the beacon anomaly watcher) can provide useful log output to analyse and track down network issues (name resolution broadcast storms).

6 ENVIRONMENT VARIABLES

PROCSERV_PID
: This 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.)

7 KNOWN PROBLEMS

None so far.

8 REPORTING BUGS

Report bugs on the procServ Trac at http://sourceforge.net/apps/trac/procserv/ or to the authors.

9 AUTHORS

Written by David H. Thompson <thompsondh@ornl.gov> and Ralph Lange <Ralph.Lange@bessy.de>.

10 RESOURCES

SourceForge project: http://sourceforge.net/projects/procserv/

11 COPYING