NAME

socketpipe – zero overhead remote process plumbing

SYNOPSIS

socketpipe [**–b**] [**–i {** *input generation command* [*args ...*] **}**] **–l {** *login command* [*args ...*] **} –r {** *remote command* [*args ...*] **}** [**–o {** *output processing command* [*args ...*] **}**]

DESCRIPTION

Socketpipe connects over a TCP/IP socket the *remote command* specified to the local *input generation command* and/or the local *output processing command*. At least one of the two local commands must be specified. The input and output of the *remote command* are appropriately redirected so that the remote command's input will come from the local *input generation command* and the remote command's output will be sent to the local *output processing command*. The remote command is executed on the machine accessed through the *login command*. The *socketpipe* executable should be available through the execution path in the remote machine. The braces used for delimiting the commands and their arguments should be space-separated and can be nested. This feature allows you to setup complex and efficient topologies of distributed communicating processes.

Although the initial *socketpipe* communication setup is performed through client-server intermediaries such as *ssh(1)* or *rsh(1)*, the communication channel that *socketpipe* establishes is a direct socket connection between the local and the remote commands. Without the use of *socketpipe*, when piping remote data through *ssh(1)* or *rsh(1)*, each data block is read at the local end by the respective client, is sent to the remote daemon and written out again to the remote process. The use of *socketpipe* removes the inefficiency of the multiple data copies and context switches and can in some cases provide dramatic throughput improvements. On the other hand, the confidentiality and integrity of the data passing through *socketpipe*'s data channel is not protected; *socketpipe* should therefore be used only within a confined LAN environment. (The authentication process uses the protocol of the underlying login program and is no more or less vulnerable than using the program in isolation; *ssh*(1) remains secure, *rsh*(1) continues to be insecure.)

OPTIONS

–l { *login command* [*args ...*] **}**

Specify the remote login command (see previous section). Use arguments to this command to specify the host and authentication options (e.g. username). The remote login command should accept as further arguments a command and its arguments and execute it on the remote host. The remote login command is used to execute a server instance of *socketpipe* on the remote host. Typical examples of remote login commands are *ssh(1)* and *rsh(1)*.

–r { *remote command* [*args ...*] **}**

Specify the remote processing command (see previous section). The remote processing command is executed on the remote machine with its input, output, or both redirected for processing to local commands.

–i { *input generation command* [*args ...*] **}**

Specify the remote input generation command (see previous section). The output of the input generation command is redirected as input to the remote command.

–o { *output processing command* [*args ...*] **}**

Specify the output processing command (see previous section). The output of the remote command is redirected as input to the output processing command.

–b

Execute the remote login command in batch mode. This option should be used when no interaction is needed for authentication purposes with the remote login command. This is for example the case when user authentication is performed by means of private keys (*ssh(1)*) or (horror) the *.rhosts(5)* file (*rsh(1)*). The option circumvents two problems in OpenSSH_3.5p1 (and possibly also other remote login commands): the setting of our (shared) output to non-blocking I/O and attempts to read from the standard input. The first problem may manifest itself through an error message of the output processing command such as "stdout: Resource temporarily unavailable". The second problem will not allow you to put *socketpipe* instances in the background, stopping them with a tty input signal (SIGTTIN). The **-b** option will close the remote login command's standard output and redirect its standard input from /dev/null solving those problems. On the other hand this flag will disable I/O to/from the remote login command and may therefore interfere with any interaction required for the authentication process.

EXAMPLE

```
socketpipe -b -i { tar cf - / } -l { ssh remotehost } -r { dd
of=/dev/st0 bs=32k }
```

Backup the local host on a tape drive located on *remotehost*.

```
socketpipe -b -l { ssh remotehost } -r { dd if=/dev/st0 bs=32k } -o {
tar xpf - /home/luser }
```

Restore a directory using the tape drive on the remote host.

```
socketpipe -b -i { tar cf - / } -l { ssh remotehost } -r { bzip2 -c } -o
{ dd of=/dev/st0 bs=32k }
```

Backup the local disk on a local tape, compressing the data on the (presumably a lot more powerful) *remotehost*.

SEE ALSO

tcpcat(1), *zsh*(1)

AUTHOR

Diomidis Spinellis -- <http://www.spinellis.gr>

BUGS

The sockets used to connect the local and remote commands may read or write only parts of the data specified in a *read(2)* or *write(2)* operation. Although this is standard behavior, and is for example correctly handled by the *stdio(3)* library, some commands may not expect it and may exhibit strange bugs. Most examples in Stevens's "*Advanced Programming in the UNIX Environment*" (Addison-Wesley 1992) would fail reading from sockets; on the other hand Section 6.6 of Stevens's "*UNIX Network Programming*" (Prentice Hall 1990) provides code that deals with this problem.