NAME

coproc, cocreate, cosend, cocheck, coreceive, codestroy – communicate with a process

SYNOPSIS

cocreate [**–r** *rpath*] [**–w** *wpath*] [**–i** *id*] [**–R** *refname*] [**–s** *send_string*]
[**–e** *expect_string*] *command*

cosend [**–n**] *proc_id string*

cocheck *proc_id*

coreceive *proc_id*

codestroy [**–R** *refname*] *proc_id* [*string*]

DESCRIPTION

These co-processing functions provide a flexible means of interaction between FMLI and an independent process; especially, they enable FMLI to be responsive to asynchronous activity.

The **cocreate** function starts *command* as a co-process and initializes communications by setting up pipes between FMLI and the standard input and standard output of *command*. The argument *command* must be an executable and its arguments (if any). This means that *command* expects strings on its input (supplied by **cosend**) and sends information on its output that can be handled in various ways by FMLI.

The **cosend** function sends *string* to the co-process identified by *proc_id* via the pipe set up by **cocreate** (optionally *wpath*), where *proc_id* can be either the *command* or *id* specified in **cocreate**. By default, **cosend** blocks, waiting for a response from the co-process. Also by default, FMLI does not send a *send_string* and does not expect an *expect_string* (except a newline). That is, it reads only one line of output from the co-process. If **–e** *expect_string* was not defined when the pipe was created, then the output of the co-process is any single string followed by a newline: any other lines of output remain on the pipe. If the **–e** option was specified when the pipe was created, **cosend** reads lines from the pipe until it reads a line starting with *expect_string*. All lines except the line starting with *expect_string* become the output of **cosend**.

The **cocheck** function determines if input is available from the process identified by *proc_id*, where *proc_id* can be either the *command* or *id* specified in **cocreate**. It returns a Boolean value, which makes **cocheck** useful in **if** statements and in other backquoted expressions in Boolean descriptors. **cocheck** receives no input from the co-process; it simply indicates if input is available from the co-process. You must use **coreceive** to actually accept the input. The **cocheck** function can be called from a **reread** descriptor to force a frame to update when new data is available. This is useful when the default value of a field in a form includes **coreceive**.

The **coreceive** function is used to read input from the co-process identified by *proc_id*, where *proc_id* can be either the *command* or *id* specified in **cocreate**. It should only be used when it has been determined, using **cocheck**, that input is actually available. If the **–e** option was used when the co-process was created, **coreceive** will continue to return lines of input until *expect_string* is read. At this point, **coreceive** will terminate. The output of **coreceive** is all the lines that were read excluding the line starting with *expect_string*. If the **–e** option was not used in the **cocreate**, each invocation of **coreceive** will return exactly one line from the co-process. If no input is available when **coreceive** is invoked, it will simply terminate without producing output.

The **codestroy** function terminates the read/write pipes to *proc-id*, where *proc_id* can be either the *command* or *id* specified in **cocreate**. It generates a **SIGPIPE** signal to the (child) co-process. This kills the co-process, unless the co-process ignores the **SIGPIPE** signal. If the co-process ignores the **SIGPIPE**, it will not die, even after the FMLI process terminates (the parent process id of the co-process will be **1**).

The optional argument *string* is sent to the co-process before the co-process dies. If *string* is not supplied, a NULL string is passed, followed by the normal *send_string* (newline by default). That is, **codestroy** will call **cosend** *proc_id string*: this implies that **codestroy** will write any output generated by the co-process to *stdout*. For example, if an interactive co-process is written to expect a "quit" string when the communication is over, the **close** descriptor could be defined;

close=`codestroy ID 'quit' | message`

and any output generated by the co-process when the string **quit** is sent to it via **codestroy** (using **cosend**) would be redirected to the message line.

The **codestroy** function should usually be given the **−R** option, since you may have more than one process with the same name, and you do not want to kill the wrong one. **codestroy** keeps track of the number of *refnames* you have assigned to a process with **cocreate**, and when the last instance is killed, it kills the process (*id*) for you. **codestroy** is typically called as part of a **close** descriptor because **close** is evaluated when a frame is closed. This is important because the co-process will continue to run if **codestroy** is not issued.

When writing programs to use as co-processes, the following tips may be useful. If the co-process program is written in C language, be sure to flush output after writing to the pipe. (Currently, **awk**(1) and **sed**(1) cannot be used in a co-process program because they do not flush after lines of output.) Shell scripts are well-mannered, but slow. C language is recommended. If possible, use the default *send_string*, *rpath* and *wpath*. In most cases, *expect_string* will have to be specified. This, of course, depends on the co-process.

In the case where asynchronous communication from a co-process is desired, a co-process program should use **vsig** to force strings into the pipe and then signal FMLI that output from the co-process is available. This causes the **reread** descriptor of all frames to be evaluated immediately.

OPTIONS

cocreate options are:

–**r** *rpath* If –**r** is specified, *rpath* is the pathname from which FMLI reads information. This option is usually used to set up communication with processes that naturally write to a certain path. If –**r** is not specified, **cocreate** will choose a unique path in **/var/tmp**.

–**w** *wpath* If –**w** is specified, *wpath* is the pathname to which **cosend** writes information. This option is usually used so that one process can talk to many different FMLI processes through the same pipe. If –**w** is not specified, **cocreate** will choose a unique path in **/var/tmp**.

–**i** *id* If –**i** is specified, *id* is an alternative name for the co-processinitialized by this **cocreate**. If –**i** is not specified, *id* defaults to *command*. The argument *id* can later be used with the other co-processing functions rather than *command*. This option is typically used, since it facilitates the creation of two or more co-processes generated from the same *command*. (For example, **cocreate -i ID1 program args** and **cocreate -i ID2 program different_args**).

–**R** *refname* If –**R** is specified, *refname* is a local name for the co-process. Since the **cocreate** function can be issued more than once, a *refname* is useful when the same co-process is referenced a second or subsequent time. With the –**R** option, if the co-process already exists a new one will not be created: the same pipes will be shared. Then, *refname* can be used as an argument to the –**R** option to **codestroy** when you want to end a particular connection to a co-process and leave other connections undisturbed. (The co-process is only killed after **codestroy** –**R** has been called as many times as **cocreate** –**R** was called.)

–**s** *send_string* The –**s** option specifies *send_string* as a string that will be appended to all output sent to the co-process using **cosend**. This option allows a co-process to know when input from FMLI has completed. The default *send_string* is a newline if –**s** is not specified.

–**e** *expect_string* The –**e** option specifies *expect_string* as a string that identifies the end of all output returned by the co-process. (Note: *expect_string* need only be the initial part of a line, and there must be a newline at the end of the co-process output.) This option allows FMLI to know when output from the co-process has completed. The default *expect_string* is a newline if –**e** is not specified.

cosend options are:

–**n** If the –**n** option is specified, **cosend** will not wait for a response from the co-process. It simply returns, providing no output. If the –**n** option is not used, a co-process that does not answer will cause FMLI to permanently hang, waiting for input from the co-process.

EXAMPLES

```
   .
   .
   .
init=`cocreate –i BIGPROCESS initialize`
close=`codestroy BIGPROCESS`
   .
   .
   .
reread=`cocheck BIGPROCESS`

name=`cosend –n BIGPROCESS field1`
   .
   .
   .
name="Receive field"
inactive=TRUE
value=`coreceive BIGPROCESS`
```

SEE ALSO

awk(1), **cat**(1), **sed**(1), **vsig**(1F)

NOTES

If **cosend** is used without the **–n** option, a co-process that does not answer will cause FMLI to permanently hang.

The use of non-alphabetic characters in input and output strings to a co-process should be avoided because they may not get transferred correctly.