IEEE P1003.2a Draft 8 – December 1991

5.24 `renice` — Set system scheduling priorities of running processes

5.24.1 Synopsis

`renice` [–n *increment*] [–g | –p | –u] *ID* . . .

Obsolescent Versions :

`renice` *nice_value* [–p] *pid* . . . [–g *gid* . . .] [–p *pid* . . .] [–u *user* . . .]

`renice` *nice_value* –g *gid* . . . [–g *gid* . . .] [–p *pid* . . .] [–u *user* . . .]

`renice` *nice_value* –u *user* . . . [–g *gid* . . .] [–p *pid* . . .] [–u *user* . . .]

5.24.2 Description

The `renice` utility shall request that the system scheduling priorities (see 2.2.2.212) of one or more running processes be changed. When a process ID is `renice`d (the default), the applicable processes are specified by their process IDs. When a process group is `renice`d, the request applies to all processes in the process group.

The system scheduling priority shall be bounded in an implementation-defined manner. If the requested *increment* (or *nice_value* in the obsolescent versions) would raise or lower the system scheduling priority of the executed utility beyond implementation-defined limits, then the limit whose value was exceeded shall be used.

When a user is `renice`d and {_POSIX_SAVED_IDS} (see 2.9.3 in POSIX.1 {8}) is defined, the request applies to all processes whose saved set-user-ID matches the user ID corresponding to the user. When a user is `renice`d and {_POSIX_SAVED_IDS} is not defined, the request applies to all processes whose real user ID matches the user ID corresponding to the user.

Regardless of which options are supplied or any other factor, `renice` shall not alter the system scheduling priorities of any process unless the user requesting such a change has appropriate privileges to do so for the specified process. If the user lacks appropriate privileges to perform the requested action, the utility shall return an error status.

On systems where {_POSIX_SAVED_IDS} is defined, the saved set-user-ID of the user's process shall be checked instead of its effective user ID when `renice` attempts to determine the user ID of the process in order to determine whether the user has appropriate privileges. On systems where {_POSIX_SAVED_IDS} is not defined, `renice` shall check the real or effective user ID when attempting to determine the user ID of the process and the user ID of the requester as part of the determination of whether the user has appropriate privileges to perform the requested action.

5.24.3 Options

The `renice` utility shall conform to the utility argument syntax guidelines described in 2.10.2. The obsolescent version shall conform with the following exceptions:

- The first operand, *nice_value* , shall precede the options and can have the appearance of a multidigit option.
- The `−g`, `−p`, and `−u` options can each take multiple option-arguments.
- The *pid* option-argument can be used without its `−p` option.

The following options shall be supported by the implementation:

`−g` Interpret all operands (or just the *gid* arguments in the obsolescent version) as unsigned decimal integer process group IDs.

`−n` *increment*

Specify how the system scheduling priority of the specified process(es) shall be adjusted. The *increment* option-argument is a positive or negative decimal integer that shall be used to modify the system scheduling priority of the specified process(es).

Positive *increment* values shall cause a lower system scheduling priority. Negative *increment* values may require appropriate privileges and shall cause a higher system scheduling priority.

`−p` Interpret all operands (or just the *pid* arguments in the obsolescent version) as unsigned decimal integer process IDs. The `−p` option is the default if no options are specified.

`−u` Interpret all operands (or just the *user* arguments in the obsolescent version) as users. If a user exists with a user name equal to the operand, then the user ID of that user shall be used in further processing. Otherwise, if the operand represents an unsigned decimal integer, it shall be used as the numeric user ID of the user.

5.24.4 Operands

The following operands shall be supported by the implementation:

ID A process ID, process group ID, or user name/user ID, depending on the option selected.

nice_value (Obsolescent.) The value specified shall be taken as the actual system scheduling priority, rather than as an increment to the existing system scheduling priority. Specifying a scheduling priority higher than that of the existing process may require appropriate privileges.

5.24.5 External Influences

5.24.5.1 Standard Input

None.

5.24.5.2 Input Files

None.

5.24.5.3 Environment Variables

The following environment variables shall affect the execution of `renice`:

LANG This variable shall determine the locale to use for the locale categories when both **LC_ALL** and the corresponding environment variable (beginning with **LC_**) do not specify a locale. See 2.6.

LC_ALL This variable shall determine the locale to be used to override any values for locale categories specified by the settings of **LANG** or any environment variables beginning with **LC_**.

LC_CTYPE This variable shall determine the locale for the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments).

LC_MESSAGES This variable shall determine the language in which messages should be written.

5.24.5.4 Asynchronous Events

Default.

5.24.6 External Effects

5.24.6.1 Standard Output

None.

5.24.6.2 Standard Error

Used only for diagnostic messages.

5.24.6.3 Output Files

None.

5.24.7 Extended Description

None.

5.24.8 Exit Status

The `renice` utility shall exit with one of the following values:

- 0 Successful completion.
- >0 An error occurred.

5.24.9 Consequences of Errors

Default.

5.24.10 Rationale. *(This subclause is not a part of P1003.2a)*

Usage, Examples

The *gid*, *pid*, and *user* specifications do not fit either the definition of operand or option-argument. However, for clarity, they have been included in the Options subclause, rather than the Operand subclause.

The following example would adjust the system scheduling priority in such a way that process IDs 987 and 32 would have a lower scheduling priority:

```
renice -n 5 -p 987 32
```

The next example would adjust the system scheduling priority in such a way that group IDs 324 and 76 would have a higher scheduling priority, if the user has the appropriate privileges to do so:

```
renice -n -4 -g 324 76
```

The next example would adjust the system scheduling priority in such a way that numeric user ID 8 and user `sas` would have a lower scheduling priority:

```
renice -n 4 -u 8 sas
```

Useful nice values on historical systems include 19 or 20 (the affected processes will run only when nothing else in the system attempts to run), 0 (the ‘‘base’’ scheduling priority), and any negative number (to make processes run faster).

History of Decisions Made

Originally this utility was written in the historical manner, using the term ‘‘nice value.’’ This was always a point of concern with users, because it was never intuitively obvious what this meant. With the new version of `renice`, which uses the term ‘‘system scheduling priority,’’ it is hoped that novice users will better understand what this utility was meant to do. It also made it much easier to document what the utility was meant to do.

The obsolescent version of `renice` allows pids, gids, and users to be mixed on the command line. This was seen as overly confusing, so when `renice` was rewritten, it was decided to allow only one kind of ID for each invocation. If no `−p`, `−g` or `−u` is specified, any ID operands are considered process IDs.

Strong input was received that the traditional syntax was very difficult to understand and it should be adjusted to comply with the utility syntax guidelines and be more reminiscent of the `nice` utility syntax.

Although this utility has use by system administrators (and in fact appears in the system administration portion of the BSD documentation), the developers of the standard felt that it was very useful for individual end users to control their own processes.

5.25 `split` — Split files into pieces

5.25.1 Synopsis

`split` [`−l` *line_count*] [`−a` *suffix_length*] [*file* [*name*]]

`split` `−b` *n*[`k`|`m`] [`−a` *suffix_length*] [*file* [*name*]]

Obsolescent Version :

`split` [−*line_count*] [`−a` *suffix_length*] [*file* [*name*]]

5.25.2 Description

The `split` utility shall read an input file and write one or more output files. The default size of each output file shall be 1000 lines. The size of the output files can be modified by specification of the `−b` or `−l` options. Each output file shall be created with a unique suffix. The suffix shall consist of exactly *suffix_length* lowercase letters from the POSIX Locale. The letters of the suffix shall be used as if they were a base-26 digit system, with the first suffix to be created consisting of all ‘`a`’’ characters, the second with a ‘`b`’’ replacing the last ‘`a`’’, etc., until a name of all ‘`z`’’s is created. By default, the names of the output files shall be ‘`x`’’, followed by a two-character suffix from the character set as described above, starting with `aa`, `ab`, `ac`, etc., and continuing until the suffix `zz`, for a maximum of 676 files.

If the number of files required exceeds the maximum allowed by the suffix length provided, such that the last allowable file would be larger than the requested size,
the `split` utility shall fail after creating the last file with a valid suffix; `split` 7
shall not delete the files it created with valid suffixes. If the file limit is not 7
exceeded, the last file created shall contain the remainder of the input file, and may be smaller than the requested size.

5.25.3 Options

The `split` utility shall conform to the utility argument syntax guidelines described in 2.10.2, except that the obsolescent version allows a multidigit option, −*line_count* .

The following options shall be supported by the implementation:

`−a` *suffix_length*
: Use *suffix_length* letters to form the suffix portion of the filenames of the split file. If `−a` is not specified, the default suffix length shall be two. If the sum of the *name* operand and the *suffix_length* option-argument would create a filename exceeding {NAME_MAX} bytes, an error shall result; `split` shall exit with a diagnostic message and no files shall be created.

`−b` *n* Split a file into pieces *n* bytes in size.

`−b` *n*`k` Split a file into pieces *n*∗1024 bytes in size.

`−b` *n*`m` Split a file into pieces *n*∗1 048 576 bytes in size.

`−l` *line_count*
−*line_count* (Obsolescent.)
Specify the number of lines in each resulting file piece. The *line_count* argument is an unsigned decimal integer. The default is 1000. If the input does not end with a `<newline>`, the partial line shall be included in the last output file.

5.25.4 Operands

The following operands shall be supported by the implementation:

file The pathname of the ordinary file to be split. If no input file is given or *file* is −, the standard input shall be used.

name The prefix to be used for each of the files resulting from the split operation. If no *name* argument is given, ‘`x`’’ shall be used as the prefix of the output files. The combined length of the basename of *prefix* and *suffix_length* cannot exceed {NAME_MAX} bytes; see 5.25.3.

5.25.5 External Influences

5.25.5.1 Standard Input

See Input Files.

5.25.5.2 Input Files

Any file can be used as input.

5.25.5.3 Environment Variables

The following environment variables shall affect the execution of `split`:

LANG This variable shall determine the locale to use for the locale categories when both **LC_ALL** and the corresponding environment variable (beginning with **LC_**) do not specify a locale. See 2.6.

LC_ALL This variable shall determine the locale to be used to override any values for locale categories specified by the settings of **LANG** or any environment variables beginning with **LC_**.

LC_CTYPE This variable shall determine the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments and input files).

LC_MESSAGES This variable shall determine the language in which messages should be written.

5.25.5.4 Asynchronous Events

Default.

5.25.6 External Effects

5.25.6.1 Standard Output

None.

5.25.6.2 Standard Error

Used only for diagnostic messages.

5.25.6.3 Output Files

The output files contain portions of the original input file, otherwise unchanged.

5.25.7 Extended Description

None.

5.25.8 Exit Status

The `split` utility shall exit with one of the following values:

0 Successful completion.

>0 An error occurred.

5.25.9 Consequences of Errors

Default.

5.25.10 Rationale. *(This subclause is not a part of P1003.2a)*

Usage, Examples

In the following examples `foo` is a text file that contains 5000 lines.

This example creates five files, `xaa`, `xab`, `xac`, `xad`, and `xae`:

```
split foo
```

This example also creates five files, but the suffixed portion of the created files consists of three letters, `xaaa`, `xaab`, `xaac`, `xaad`, and `xaae`:

```
split -a 3 foo
```

This example creates three files with four-letter suffixes and a supplied prefix, `bar_aaaa`, `bar_aaab`, and `bar_aaac`:

```
split -a 4 -l 2000 foo bar_
```

This example creates as many files as are necessary to contain at most 20∗1024 bytes, each with the default prefix of ‘`x`’’ and a five-letter suffix:

```
split -a 5 -b 20k foo
```

History of Decisions Made

The −`b` option was added to provide a mechanism for splitting files other than by lines. While most uses of the −`b` option will be for transmitting files over networks, some felt it would have additional uses.

The −`a` option was added to overcome the limitation of being able to create only 676 files.

The −*line_count* option was declared obsolescent and replaced with the −`l` option. This allows future standards to have a `split` utility that fully conforms to the syntax guidelines.

Consideration was given to deleting this utility using the rationale that the function provided by this utility is available via the `csplit` utility (see 5.6). Upon reconsideration of the purpose of the User Portability Extension, it was decided to retain both this utility and the `csplit` utility because users use both utilities and have historical expectations of their behavior. Furthermore, the splitting on byte boundaries in `split` cannot be duplicated with the historical `csplit`.

The text ‘‘`split` shall not delete the files it created with valid suffixes’’ would nor- 7
mally be assumed, but since the related utility, `csplit`, does delete files under 7
some circumstances, the historical behavior of `split` is made explicit to avoid 7
misinterpretation. 7

5.26 `strings` — Find printable strings in files

5.26.1 Synopsis

`strings` [−a][−t *format*][−n *number*][*file* ...]

Obsolescent Version :

`strings` [−][−t *format*][−*number*][*file* ...]

5.26.2 Description

The `strings` utility shall look for printable strings in regular files and write those strings to standard output. A printable string is any sequence of four (by default) or more printable characters terminated by a `<newline>` or NUL character. Additional implementation-defined strings may be written. (See `localedef` in 4.35.)

5.26.3 Options

The `strings` utility shall conform to the utility argument syntax guidelines described in 2.10.2, except that the obsolescent version uses − in a nonstandard way and allows a multidigit option, −*number* .

The following options shall be supported by the implementation:

`−a`
`-` (Obsolescent.)
Scan files in their entirety. If `−a` is not specified, it is implementation defined what portion of each file is scanned for strings.

`−n` *number*
−*number* (Obsolescent.)
Specify the minimum string length, where the *number* argument is a positive decimal integer. The default shall be 4.

`−t` *format* Write each string preceded by its byte offset from the start of the file. The format shall be dependent on the single character used as the *format* option-argument:

`d` The offset shall be written in decimal.

`o` The offset shall be written in octal.

`x` The offset shall be written in hexadecimal.

5.26.4 Operands

The following operand shall be supported by the implementation:

file A pathname of a regular file to be used as input. If no *file* operand is specified, the `strings` utility shall read from the standard input.

5.26.5 External Influences

5.26.5.1 Standard Input

See Input Files.

5.26.5.2 Input Files

The input files named by the utility arguments or the standard input shall be regular files of any format.

5.26.5.3 Environment Variables

The following environment variables shall affect the execution of `strings`:

LANG This variable shall determine the locale to use for the locale categories when both **LC_ALL** and the corresponding environment variable (beginning with **LC_**) do not specify a locale. See 2.6.

LC_ALL This variable shall determine the locale to be used to override any values for locale categories specified by the settings of **LANG** or any environment variables beginning with **LC_**.

LC_CTYPE This variable shall determine the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments and input files), and to identify printable strings.

LC_MESSAGES This variable shall determine the language in which messages should be written.

5.26.5.4 Asynchronous Events

Default.

5.26.6 External Effects

5.26.6.1 Standard Output

Strings found shall be written to the standard output, one per line.

When the `−t` option is not specified, the format of the output shall be:

`"%s"`, *<string>*

With the `−t o` option, the format of the output shall be:

`"%o %s"`, *<byte offset>*, *<string>*

With the `−t x` option, the format of the output shall be:

`"%x %s"`, *<byte offset>*, *<string>*

With the `−t d` option, the format of the output shall be:

`"%d %s"`, *<byte offset>*, *<string>*

5.26.6.2 Standard Error

Used only for diagnostic messages.

5.26.6.3 Output Files

None.

5.26.7 Extended Description

None.

5.26.8 Exit Status

The `strings` utility shall exit with one of the following values:

- 0 Successful completion.
- >0 An error occurred.

5.26.9 Consequences of Errors

Default.

5.26.10 Rationale. *(This subclause is not a part of P1003.2a)*

Usage, Examples

The `strings` utility is useful for identifying random object files and many other things.

History of Decisions Made

Apart from rationalizing the option syntax and slight difficulties with object and executable binary files, `strings` is specified to closely match existing practice. The `−a` and `−n` options were introduced to replace the nonconforming `−` and −*number* options.

Investigation revealed that `−o` historically means different things on different implementations. Some maintain that it means ''*offset* in decimal,'' while others maintain that it means ''offset in *octal*.'' Instead of trying to decide which way would be least objectionable, the `−t` option was added. It was originally named `−O` to mean ''offset,'' but was changed to `−t` to be consistent with `od`.

By default, the data area (as opposed to the text, ''bss,'' or header areas) of a binary executable file is scanned in historical implementations. Implementations are required to document which areas are scanned.

Some historical implementations do not require NUL or `<newline>` terminators for strings to permit those languages that do not use NUL as a string terminator to have their strings written.

The C Standard {7} function *isprint* () is restricted to a domain of *unsigned char* . This standard requires implementations to write strings as defined by the current locale.

5.27 `tabs` — Set terminal tabs

5.27.1 Synopsis

`tabs` [−*n*] [−`T` *type*]

`tabs` [−`T` *type*] *n1* [`,` *n2* `,` . . .]

5.27.2 Description

The `tabs` utility shall display a series of characters that first clears the hardware terminal tab settings and then initializes the tab stops at the specified positions.

The phrase ‘‘tab-stop position *N*’’ shall be taken to mean that, from the start of a line of output, tabbing to position *N* shall cause the next character output to be in the (N+1)th column position on that line. The maximum number of tab stops allowed is terminal dependent.

It need not be possible to implement `tabs` on certain terminals. If the terminal type obtained from the **TERM** environment variable or −`T` option represents such a terminal, an appropriate diagnostic message shall be written to standard error and `tabs` shall exit with a status greater than zero.

5.27.3 Options

The `tabs` utility shall conform to the utility argument syntax guidelines described in 2.10.2.

The following options shall be supported by the implementation:

−*n* Specify repetitive tab stops separated by a uniform number of column positions, *n*, where *n* is a single-digit decimal number. The default usage of `tabs` with no arguments shall be equivalent to `tabs −8`. When `−0` is used, the tab stops shall be cleared and no new ones set.

−`T` *type* Indicate the type of terminal. If this option is not supplied and
the **TERM** variable is unset or null, an unspecified default termi- 8
nal type shall be used. The setting of *type* shall take precedence 8
over the value in **TERM** .

5.27.4 Operands

The following operand shall be supported by the implementation:

n1 [`,` *n2* `,` . . .]
A single command-line argument that consists of tab-stop values separated using either commas or `<blank>`s. The tab-stop values shall be positive decimal integers in strictly ascending order. If

any number (except the first one) is preceded by a plus sign, it is taken as an increment to be added to the previous value. For example, the tab lists `1,10,20,30` and `1,10,+10,+10` are considered to be identical.

5.27.5 External Influences

5.27.5.1 Standard Input

None.

5.27.5.2 Input Files

None.

5.27.5.3 Environment Variables

The following environment variables shall affect the execution of `tabs`:

LANG This variable shall determine the locale to use for the locale categories when both **LC_ALL** and the corresponding environment variable (beginning with **LC_**) do not specify a locale. See 2.6.

LC_ALL This variable shall determine the locale to be used to override any values for locale categories specified by the settings of **LANG** or any environment variables beginning with **LC_**.

LC_CTYPE This variable shall determine the locale for the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments).

LC_MESSAGES This variable shall determine the language in which messages should be written.

TERM This variable shall be interpreted as the name of the terminal type. If this variable is unset or null, and if the `-T` 7
option is not specified, an unspecified default terminal 7
type shall be used. 7

5.27.5.4 Asynchronous Events

Default.

5.27.6 External Effects

5.27.6.1 Standard Output

If standard output is a terminal, the appropriate sequence to clear and set the tab stops may be written to standard output in an unspecified format. If standard output is not a terminal, undefined results occur.

5.27.6.2 Standard Error

Used only for diagnostic messages.

5.27.6.3 Output Files

None.

5.27.7 Extended Description

None.

5.27.8 Exit Status

The `tabs` utility shall exit with one of the following values:

0 Successful completion.

>0 An error occurred.

5.27.9 Consequences of Errors

Default.

5.27.10 Rationale. *(This subclause is not a part of P1003.2a)*

Usage, Examples

Some integrated display units might not have escape sequences to set tab stops, but may be set by internal system calls. On these terminals, `tabs` will work if standard output is directed to the terminal; if output is directed to another file, however, `tabs` will fail.

History of Decisions Made

Consideration was given to having the `tput` utility handle all of the functions described here. However, the separate `tabs` utility was retained because it seems more intuitive to use a command named `tabs` than `tput` with a new option. The POSIX. 2a `tput` does not support setting or clearing tabs and no known historical version of `tabs` supports the capability of setting arbitrary tab stops.

The System V `tabs` interface is very complex; the version in this standard has a reduced feature list. There was considerable sentiment for specifying only a means of resetting the tabs back to a known state—presumably the ''standard'' tabs every eight positions. The following features were omitted:

(1) Setting tab stops tailored for certain programming languages; the working group was afraid that there would be no end to the bickering about which languages to include and where the tabs should be (even C programmers cannot agree on this).

(2) Setting tab stops via the first line in a file, using −−*file*. Since even the *SVID* has no complete explanation of this feature, it is doubtful that it is in widespread use.

(3) Setting the left margin using `"+m`*n*`"`. As this does not work with all terminal types, it was omitted.

In an earlier draft, a `−t` *tablist* option was added for consistency with `expand`; this was later removed when inconsistencies with the historical list of tabs were identified.

Consideration was given to adding a `−p` option that would output the current tab settings so that they could be saved and then later restored. This was not accepted because querying the terminal's tab stops is not a capability in historical *terminfo* or *termcap* facilities and might not be supportable on a wide range of terminals.

5.28 `talk` — Talk to another user

5.28.1 Synopsis

`talk` *address* [*terminal*]

5.28.2 Description

The `talk` utility is a two-way, screen-oriented communication program.

When first invoked, `talk` shall send a message similar to:

```
Message from <unspecified string>
talk: connection requested by your_address
talk: respond with: talk your_address
```

to the specified *address* . At this point, the recipient of the message can reply by typing:

```
talk your_address
```

Once communication is established, the two parties can type simultaneously, with
their output displayed in separate regions of the screen. Characters shall be pro- 8
cessed as follows: 8

— Typing the `<alert>` character shall alert the recipient's terminal.

— Typing `control-L` shall cause the sender's screen regions to be refreshed.

— Typing the erase and kill characters shall affect the sender's terminal in the manner described by the POSIX.1 {8} *termios* interface.

— Typing the interrupt or end-of-file characters shall terminate the local `talk` utility. Once the `talk` session has been terminated on one side, the other side of the `talk` session shall be notified that the `talk` session has been terminated and shall be able to do nothing except exit.

— Typing characters from LC_CTYPE classifications `print` or `space` shall 8
cause those characters to be sent to the recipient's terminal.

— When and only when the `stty` *iexten* local mode (see 4.59.4.4) is enabled, 8
additional special control characters and multibyte or single-byte functions 8
shall be processed in an implementation-defined manner. (See POSIX.1 {8} 8
section 7.1.1.9.) 8

— Typing other nonprintable characters shall cause implementation-defined sequences of printable characters to be sent to the recipient's terminal.

Permission to be a recipient of a `talk` message can be denied or granted by use of
the `mesg` utility (see 5.17). However, a user's privilege may further constrain the 7
domain of accessibility of other users' terminals. The `talk` utility shall fail when 7
the user lacks the appropriate privileges to perform the requested action. 7

Certain block-mode terminals do not have all the capabilities necessary to support the simultaneous exchange of messages required for `talk`. When this type of exchange cannot be supported on such terminals, the implementation may support an exchange with reduced levels of simultaneous interaction or it may report an error describing the terminal-related deficiency.

5.28.3 Options

None.

5.28.4 Operands

The following operands shall be supported by the implementation:

address The recipient of the `talk` session. One form of *address* is the *<user name>*, as returned by the `who` utility (see 5.36). Other address formats and how they are handled are unspecified.

terminal If the recipient is logged in more than once, the *terminal* argument can be used to indicate the appropriate terminal name. If *terminal* is not specified, the `talk` message shall be displayed on one or more accessible terminals in use by the recipient. The format of *terminal* shall be the same as that returned by the `who` utility.

5.28.5 External Influences

5.28.5.1 Standard Input

Characters read from standard input shall be copied to the recipient's terminal in an unspecified manner. If standard input is not a terminal, talk shall write a diagnostic message and exit with a nonzero status.

5.28.5.2 Input Files

None.

5.28.5.3 Environment Variables

The following environment variables shall affect the execution of `talk`:

LANG This variable shall determine the locale to use for the locale categories when both **LC_ALL** and the corresponding environment variable (beginning with **LC_**) do not specify a locale. See 2.6.

LC_ALL This variable shall determine the locale to be used to override any values for locale categories specified by the settings of **LANG** or any environment variables beginning with **LC_**.

LC_CTYPE This variable shall determine the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments and input files). If the recipient's locale does not use an LC_CTYPE equivalent to the sender's, the results are undefined.

LC_MESSAGES This variable shall determine the language in which diagnostic messages should be displayed.

TERM This variable shall be interpreted as the name of the invoker's terminal type. If this variable is unset or null, 7
an unspecified default terminal type shall be used. 7

5.28.5.4 Asynchronous Events

When the `talk` utility receives a SIGINT signal, the utility shall terminate and exit with a zero status. It shall take the standard action for all other signals; see 2.11.5.4.

5.28.6 External Effects

5.28.6.1 Standard Output

If standard output is a terminal, characters copied from the recipient's standard input may be written to standard output. Standard output also may be used for diagnostic messages. If standard output is not a terminal, `talk` shall exit with a nonzero status.

5.28.6.2 Standard Error

None.

5.28.6.3 Output Files

None.

5.28.7 Extended Description

None.