-- --

IEEE P1003.2 Draft 11.2 – September 1991

utility.

One reviewer suggested that an implementation of *system* () might want to use an environment variable such as **SHELL** to determine which command interpreter to use. The supposed implementation would use the default command interpreter if the one specified by the environment variable was not available. This would allow a user, when using an application that prompts for command lines to be processed using *system* (), to specify a different command interpreter. Such an implementation is discouraged. If the alternate command interpreter did not follow the command line syntax specified in POSIX.2, then changing **SHELL** would render *system* () nonconformant. This would affect applications that expected the specified behavior from *system* (), and since this standard does not mention that **SHELL** affects *system* (), the application would not know that it needed to unset **SHELL** .

B.3.2 C Binding for Pipe Communications with Programs

Functions: *popen* (), *pclose* ()

B.3.2.1 Synopsis

```
#include <stdio.h>

FILE *popen(const char *command , const char *mode );

int pclose(FILE *stream );
```

B.3.2.2 Description

The *popen* () function shall execute the command specified by the string *command* . It shall create a pipe between the calling program and the executed command, and return a pointer to a C Standard {7} stream that can be used to either read from or write to the pipe. The *pclose* () function shall close the stream, wait for the command to terminate, and return the termination status from the command language interpreter.

The environment of the executed command shall be as if a child process were created within the *popen* () call using the *fork* () function, and the child invoked the `sh` utility using the call:

```
execl(<shell path >, "sh", "-c", command , (char *)0);
```

where *<shell path>* is an unspecified pathname for the `sh` utility. However, 1
popen () shall ensure that any streams from previous *popen* () calls that remain 1
open in the parent process are closed in the new child process. 1

The *mode* argument to *popen* () is a string that specifies I/O mode:

(1) If *mode* is `"r"`, when the child process is started its file descriptor STDOUT_FILENO shall be the writable end of the pipe, and the file descriptor *fileno* (*stream*) in the calling process, where *stream* is the stream pointer returned by *popen* (), shall be the readable end of the pipe.

(2) If *mode* is `"w"`, when the child process is started its file descriptor STDIN_FILENO shall be the readable end of the pipe, and the file descriptor *fileno*(*stream*) in the calling process, where *stream* is the stream pointer returned by *popen*(), shall be the writable end of the pipe.

(3) If *mode* is any other value, the result is undefined.

A stream opened by *popen*() should be closed by *pclose*(). As stated above, *pclose*() shall return the termination status from the command language interpreter. However, if the application has called any of the following:

(1) *wait*(),

(2) *waitpid*() with a *pid* argument less than or equal to zero or equal to the process ID of the command line interpreter, or

(3) any other function not defined in POSIX.1 {8} or POSIX.2 that could do one of the above

and one of those calls caused the termination status to be unavailable to *pclose*(), then *pclose*() shall return −1 with *errno* set to [ECHILD] to report this situation. In any case, *pclose*() shall not return before the child process created by *popen*() has terminated.

If the command language interpreter cannot be executed, the child termination status returned by *pclose*() shall be as if the command language interpreter terminated using *exit*(127) or *_exit*(127). If it can be executed, the *exit*() value shall be as described for the `sh` utility.

The *pclose*() function shall not affect the termination status of any child of the calling process other than the one created by *popen*() for the associated stream.

If the argument *stream* to *pclose*() is not a pointer to a stream created by *popen*(), the result of *pclose*() is undefined.

After *popen*(), both the parent and the child process shall be capable of executing independently before either terminates. See 2.9.1.2.

B.3.2.3 Returns

The *popen*() function shall return a **NULL** pointer if the pipe or subprocess cannot be created. Otherwise, it shall return a stream pointer as described above.

Upon successful return, *pclose*() shall return the termination status of the command language interpreter. Otherwise, *pclose*() shall return −1 and set *errno* to indicate the error.

B.3.2.4 Errors

If any of the following conditions are detected, the *popen*() function shall return **NULL** and set *errno* to the corresponding value:

[EINVAL] The *mode* argument is invalid.

The *popen*() function may also set *errno* values as described by the POSIX.1 {8} *fork*() or *pipe*() functions.

If any of the following conditions are detected, the *pclose*() function shall return −1 and set *errno* to the corresponding value:

[ECHILD] The status of the child process could not be obtained, as described above.

B.3.2.5 Rationale. *(This subclause is not a part of P1003.2)*

Examples, Usage

Because open files are shared, a mode `"r"` command can be used as an input filter and a mode `"w"` command as an output filter.

The behavior of *popen*() is specified for *mode*s of `"r"` and `"w"`. Other modes such as `"rb"` and `"wb"` might be supported by specific implementations, but these would not be portable features. Note that historical implementations of *popen*() only check to see if the first character of *mode* is `r`. Thus, a *mode* of `"robert the robot"` would be treated as *mode* `"r"`, and a *mode* of `"anything else"` would be treated as *mode* `"w"`.

If the application calls *waitpid*() with a *pid* argument greater than zero, and it still has a *popen*()ed stream open, it must ensure that *pid* does not refer to the process started by *popen*().

History of Decisions Made

There is a requirement that *pclose*() not return before the child process terminates. This is intended to disallow implementations that return [EINTR] if a signal is received while waiting. If *pclose*() returned before the child terminated, there would be no way for the application to discover which child used to be associated with the stream, and it could not do the cleanup itself.

If the stream pointed to by *stream* was not created by *popen*(), historical implementations of *pclose*() return −1 without setting *errno*. To avoid requiring *pclose*() to set *errno* in this case, this standard makes the behavior undefined. An application should not use *pclose*() to close any stream that wasn't created by *popen*().

Wording was added in Draft 10 requiring that the parent and child processes be able to execute independently. This behavior has been the intent all along, and the specific words were taken from the current draft of the POSIX.1a revision to POSIX.1 {8}. Rationale about this wording appears in B.3.1.1 of POSIX.1a.

Some historical implementations either block or ignore the signals SIGINT, SIGQUIT, and SIGHUP while waiting for the child process to terminate. Since this behavior is not described in POSIX.2, such implementations are not conforming. Also, some historical implementations return [EINTR] if a signal is received, even though the child process has not terminated. Such implementations are also considered nonconforming.

Consider, for example, an application that uses

```
popen("command", "r")
```

to start *command* , which is part of the same application. The parent writes a prompt to its standard output (presumably the terminal) and then reads from the *popened* stream. The child reads the response from the user, does some transformation on the response (pathname expansion, perhaps) and writes the result to its standard output. The parent process reads the result from the pipe, does something with it, and prints another prompt. The cycle repeats. Assuming that both processes do appropriate buffer flushing, this would be expected to work.

Modified in Draft 6 to reflect the availability of the *waitpid* () function in POSIX.1 {8}. To conform to this standard, *pclose* () must use *waitpid* (), or some similar function, instead of *wait* ().

Figure B-2 illustrates how the *pclose* () function might be implemented on a POSIX.1 {8} system.

```
int pclose(FILE *stream)                                                1
{
        int     stat;
        pid_t   pid;

        pid = <pid for process created for stream by popen()>
        (void) fclose(stream);

        while (waitpid(pid, &stat, 0) == -1) {
                if (errno != EINTR) {
                        stat = -1;
                        break;
                }
        }
        return(stat);
}
```

Figure B-2 – Sample *pclose* () Implementation

B.4 C Binding for Access Environment Variables

Function: *getenv* ()

The C language binding to the service described in 7.2 shall be the POSIX.1 {8} *getenv* () function.

B.5 C Binding for Regular Expression Matching

Functions: *regcomp* (), *regexec* (), *regfree* (), *regerror* ()

B.5.1 Synopsis

```
#include <sys/types.h>
#include <regex.h>

int regcomp(regex_t *preg, const char *pattern, int cflags);

int regexec(const regex_t *preg, const char *string,
        size_t nmatch, regmatch_t pmatch[], int eflags);

size_t regerror(int errcode, const regex_t *preg,
        char *errbuf, size_t errbuf_size);

void regfree(regex_t *preg);
```

B.5.2 Description

These functions shall interpret basic and extended regular expressions, as described in 2.8.

The header `<regex.h>` shall define the structure types *regex_t* and *regmatch_t*. The structure type *regex_t* shall include at least the member shown in Table B-6.

The structure type *regmatch_t* shall contain at least the members shown in
Table B-7. The type *regoff_t*, which shall be defined in `<regex.h>`, shall be a 1
signed arithmetic type that can hold the largest value that can be stored in either 1
an *off_t* or a *ssize_t*. 1

The *regcomp* () function shall compile the regular expression contained in the 1
string pointed to by the *pattern* argument and place the results in the structure 1
pointed to by *preg*. The *cflags* argument shall be the bitwise inclusive OR of zero or more of the flags shown in Table B-8, which shall be defined in the header `<regex.h>`.

The default regular expression type for *pattern* shall be a Basic Regular Expression. The application can specify Extended Regular Expressions using the REG_EXTENDED *cflags* flag.

If the function *regcomp* () succeeds, it shall return zero; otherwise it shall return nonzero, and the content of *preg* shall be undefined.