NAME

explain_execve − explain execve(2) errors

SYNOPSIS

#include <libexplain/execve.h>

const char *explain_execve(const char *pathname, const char *const *argv, const char *const *envp);
const char *explain_errno_execve(int errnum, const char *pathname, const char *const *argv, const char *const *envp);
void explain_message_execve(char *message, int message_size, const char *pathname, const char *const *argv, const char *const *envp);
void explain_message_errno_execve(char *message, int message_size, int errnum, const char *pathname, const char *const *argv, const char *const *envp);

DESCRIPTION

These functions may be used to obtain explanations for errors returned by the *execve*(2) system call.

explain_execve

const char *explain_execve(const char *pathname, const char *const *argv, const char *const *envp);

The **explain_execve** function is used to obtain an explanation of an error returned by the *execve*(2) system call. The least the message will contain is the value of `strerror(errno)`, but usually it will do much better, and indicate the underlying cause in more detail.

The *errno* global variable will be used to obtain the error value to be decoded.

This function is intended to be used in a fashion similar to the following example:

```
execve(pathname, argv, envp);
fprintf(stderr, "%s\n", explain_execve(pathname, argv, envp));
exit(EXIT_FAILURE);
```

pathname
The original pathname, exactly as passed to the *execve*(2) system call.

argv The original argv, exactly as passed to the *execve*(2) system call.

envp The original envp, exactly as passed to the *execve*(2) system call.

Returns: The message explaining the error. This message buffer is shared by all libexplain functions which do not supply a buffer in their argument list. This will be overwritten by the next call to any libexplain function which shares this buffer, including other threads.

Note: This function is **not** thread safe, because it shares a return buffer across all threads, and many other functions in this library.

explain_errno_execve

const char *explain_errno_execve(int errnum, const char *pathname, const char *const *argv, const char *const *envp);

The **explain_errno_execve** function is used to obtain an explanation of an error returned by the *execve*(2) system call. The least the message will contain is the value of `strerror(errnum)`, but usually it will do much better, and indicate the underlying cause in more detail.

This function is intended to be used in a fashion similar to the following example:

```
execve(pathname, argv, envp);
int err = errno;
fprintf(stderr, "%s\n", explain_errno_execve(err, pathname, argv, envp));
exit(EXIT_FAILURE);
```

errnum The error value to be decoded, usually obtained from the *errno* global variable just before this function is called. This is necessary if you need to call **any** code between the system call to be explained and this function, because many libc functions will alter the value of *errno*.

pathname
The original pathname, exactly as passed to the *execve*(2) system call.

argv The original argv, exactly as passed to the *execve*(2) system call.

envp The original envp, exactly as passed to the *execve*(2) system call.

Returns: The message explaining the error. This message buffer is shared by all libexplain functions which do not supply a buffer in their argument list. This will be overwritten by the next call to any libexplain function which shares this buffer, including other threads.

Note: This function is **not** thread safe, because it shares a return buffer across all threads, and many other functions in this library.

explain_message_execve

void explain_message_execve(char *message, int message_size, const char *pathname, const char *const *argv, const char *const *envp);

The **explain_message_execve** function may be used to obtain an explanation of an error returned by the *execve*(2) system call. The least the message will contain is the value of `strerror(errno)`, but usually it will do much better, and indicate the underlying cause in more detail.

The *errno* global variable will be used to obtain the error value to be decoded.

This function is intended to be used in a fashion similar to the following example:

```
execve(pathname, argv, envp);
char message[3000];
explain_message_execve(message, sizeof(message), pathname, argv, envp);
fprintf(stderr, "%s\n", message);
exit(EXIT_FAILURE);
```

message The location in which to store the returned message. If a suitable message return buffer is supplied, this function is thread safe.

message_size
The size in bytes of the location in which to store the returned message.

pathname
The original pathname, exactly as passed to the *execve*(2) system call.

argv The original argv, exactly as passed to the *execve*(2) system call.

envp The original envp, exactly as passed to the *execve*(2) system call.

explain_message_errno_execve

void explain_message_errno_execve(char *message, int message_size, int errnum, const char *pathname, const char *const *argv, const char *const *envp);

The **explain_message_errno_execve** function may be used to obtain an explanation of an error returned by the *execve*(2) system call. The least the message will contain is the value of `strerror(errnum)`, but usually it will do much better, and indicate the underlying cause in more detail.

This function is intended to be used in a fashion similar to the following example:

```
execve(pathname, argv, envp);
int err = errno;
char message[3000];
explain_message_errno_execve(message, sizeof(message), err,
    pathname, argv, envp);
fprintf(stderr, "%s\n", message);
exit(EXIT_FAILURE);
```

message The location in which to store the returned message. If a suitable message return buffer is supplied, this function is thread safe.

message_size
The size in bytes of the location in which to store the returned message.

errnum The error value to be decoded, usually obtained from the *errno* global variable just before this function is called. This is necessary if you need to call **any** code between the system call to be explained and this function, because many libc functions will alter the value of *errno*.

pathname
The original pathname, exactly as passed to the *execve*(2) system call.

argv The original argv, exactly as passed to the *execve*(2) system call.

envp The original envp, exactly as passed to the *execve*(2) system call.

SEE ALSO

execve(2)
execute program

explain_execve_or_die(3)
execute program and report errors

COPYRIGHT

libexplain version 1.4