## NAME

explain_execvp − explain execvp(3) errors

## SYNOPSIS

#include <libexplain/execvp.h>

const char *explain_execvp(const char *pathname, char *const *argv);
const char *explain_errno_execvp(int errnum, const char *pathname, char *const *argv);
void explain_message_execvp(char *message, int message_size, const char *pathname, char *const *argv);
void explain_message_errno_execvp(char *message, int message_size, int errnum, const char *pathname, char *const *argv);

## DESCRIPTION

These functions may be used to obtain explanations for errors returned by the *execvp*(3) system call.

### explain_execvp

const char *explain_execvp(const char *pathname, char *const *argv);

The **explain_execvp** function is used to obtain an explanation of an error returned by the *execvp*(3) 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:

```
if (execvp(pathname, argv) < 0)
{
    fprintf(stderr, "%s\n", explain_execvp(pathname, argv));
    exit(EXIT_FAILURE);
}
```

The above code example is available pre-packaged as the *explain_execvp_or_die*(3) function.

*pathname*
: The original pathname, exactly as passed to the *execvp*(3) system call.

*argv*
: The original argv, exactly as passed to the *execvp*(3) 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_execvp

const char *explain_errno_execvp(int errnum, const char *pathname, char *const *argv);

The **explain_errno_execvp** function is used to obtain an explanation of an error returned by the *execvp*(3) 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:

```
if (execvp(pathname, argv) < 0)
{
    int err = errno;
    fprintf(stderr, "%s\n", explain_errno_execvp(err,
        pathname, argv));
    exit(EXIT_FAILURE);
}
```

The above code example is available pre-packaged as the *explain_execvp_or_die*(3) function.

*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 *execvp*(3) system call.

*argv* The original argv, exactly as passed to the *execvp*(3) 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_execvp**
void explain_message_execvp(char *message, int message_size, const char *pathname, char *const *argv);

The **explain_message_execvp** function may be used to obtain an explanation of an error returned by the *execvp*(3) 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:

```
if (execvp(pathname, argv) < 0)
{
    char message[3000];
    explain_message_execvp(message, sizeof(message), pathname, argv);
    fprintf(stderr, "%s\n", message);
    exit(EXIT_FAILURE);
}
```

The above code example is available pre-packaged as the *explain_execvp_or_die*(3) function.

*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 *execvp*(3) system call.

*argv* The original argv, exactly as passed to the *execvp*(3) system call.

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

The **explain_message_errno_execvp** function may be used to obtain an explanation of an error returned by the *execvp*(3) 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:

```
if (execvp(pathname, argv) < 0)
{
    int err = errno;
    char message[3000];
    explain_message_errno_execvp(message, sizeof(message),
        err, pathname, argv);
    fprintf(stderr, "%s\n", message);
    exit(EXIT_FAILURE);
}
```

The above code example is available pre-packaged as the *explain_execvp_or_die*(3) function.

*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 *execvp*(3) system call.

*argv* The original argv, exactly as passed to the *execvp*(3) system call.

## SEE ALSO

*execvp*(3)
execute a file

*explain_execvp_or_die*(3)
execute a file and report errors

## COPYRIGHT