NAME

pipe, pipe2 − create pipe

SYNOPSIS

#include <unistd.h>

/* On Alpha, IA-64, MIPS, SuperH, and SPARC/SPARC64; see NOTES */
struct fd_pair {
  long fd[2];
};
struct fd_pair pipe();

/* On all other architectures */
int pipe(int *pipefd***[2]);**

#define _GNU_SOURCE   /* See feature_test_macros(7) */
#include <fcntl.h>   /* Obtain O_* constant definitions */
#include <unistd.h>

int pipe2(int *pipefd***[2], int** *flags***);**

DESCRIPTION

pipe() creates a pipe, a unidirectional data channel that can be used for interprocess communication. The array *pipefd* is used to return two file descriptors referring to the ends of the pipe. *pipefd[0]* refers to the read end of the pipe. *pipefd[1]* refers to the write end of the pipe. Data written to the write end of the pipe is buffered by the kernel until it is read from the read end of the pipe. For further details, see **pipe**(7).

If *flags* is 0, then **pipe2**() is the same as **pipe**(). The following values can be bitwise ORed in *flags* to obtain different behavior:

O_CLOEXEC
  Set the close-on-exec (**FD_CLOEXEC**) flag on the two new file descriptors. See the description of the same flag in **open**(2) for reasons why this may be useful.

O_DIRECT (since Linux 3.4)
  Create a pipe that performs I/O in "packet" mode. Each **write**(2) to the pipe is dealt with as a separate packet, and **read**(2)s from the pipe will read one packet at a time. Note the following points:

  *   Writes of greater than **PIPE_BUF** bytes (see **pipe**(7)) will be split into multiple packets. The constant **PIPE_BUF** is defined in *<limits.h>*.

  *   If a **read**(2) specifies a buffer size that is smaller than the next packet, then the requested number of bytes are read, and the excess bytes in the packet are discarded. Specifying a buffer size of **PIPE_BUF** will be sufficient to read the largest possible packets (see the previous point).

  *   Zero-length packets are not supported. (A **read**(2) that specifies a buffer size of zero is a no-op, and returns 0.)

  Older kernels that do not support this flag will indicate this via an **EINVAL** error.

  Since Linux 4.5, it is possible to change the **O_DIRECT** setting of a pipe file descriptor using **fcntl**(2).

O_NONBLOCK
  Set the **O_NONBLOCK** file status flag on the open file descriptions referred to by the new file descriptors. Using this flag saves extra calls to **fcntl**(2) to achieve the same result.

RETURN VALUE

On success, zero is returned. On error, −1 is returned, *errno* is set appropriately, and *pipefd* is left unchanged.

On Linux (and other systems), **pipe**() does not modify *pipefd* on failure. A requirement standardizing this behavior was added in POSIX.1-2016. The Linux-specific **pipe2**() system call likewise does not modify *pipefd* on failure.

ERRORS

EFAULT

pipefd is not valid.

EINVAL

(**pipe2**()) Invalid value in *flags*.

EMFILE

The per-process limit on the number of open file descriptors has been reached.

ENFILE

The system-wide limit on the total number of open files has been reached.

ENFILE

The user hard limit on memory that can be allocated for pipes has been reached and the caller is not privileged; see **pipe**(7).

VERSIONS

pipe2() was added to Linux in version 2.6.27; glibc support is available starting with version 2.9.

NOTES

The SystemV ABI on some architectures allows the use of more than one register for returning multiple values; several architectures (namely, Alpha, IA-64, MIPS, SuperH, and SPARC/SPARC64) (ab)use this feature in order to implement the **pipe**() system call in a functional manner: the call doesn't take any arguments and returns a pair of file descriptors as the return value on success. The glibc **pipe**() wrapper function transparently deals with this. See **syscall**(2) for information regarding registers used for storing second file descriptor.

CONFORMING TO

pipe(): POSIX.1-2001, POSIX.1-2008.

pipe2() is Linux-specific.

EXAMPLE

The following program creates a pipe, and then **fork**(2)s to create a child process; the child inherits a duplicate set of file descriptors that refer to the same pipe. After the **fork**(2), each process closes the file descriptors that it doesn't need for the pipe (see **pipe**(7)). The parent then writes the string contained in the program's command-line argument to the pipe, and the child reads this string a byte at a time from the pipe and echoes it on standard output.

Program source

```
#include <sys/types.h>
#include <sys/wait.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

int
main(int argc, char *argv[])
{
    int pipefd[2];
    pid_t cpid;
    char buf;

    if (argc != 2) {
        fprintf(stderr, "Usage: %s <string>\n", argv[0]);
        exit(EXIT_FAILURE);
    }
```

```
       if (pipe(pipefd) == -1) {
           perror("pipe");
           exit(EXIT_FAILURE);
       }

       cpid = fork();
       if (cpid == -1) {
           perror("fork");
           exit(EXIT_FAILURE);
       }

       if (cpid == 0) {    /* Child reads from pipe */
           close(pipefd[1]);          /* Close unused write end */

           while (read(pipefd[0], &buf, 1) > 0)
               write(STDOUT_FILENO, &buf, 1);

           write(STDOUT_FILENO, "\n", 1);
           close(pipefd[0]);
           _exit(EXIT_SUCCESS);

       } else {            /* Parent writes argv[1] to pipe */
           close(pipefd[0]);          /* Close unused read end */
           write(pipefd[1], argv[1], strlen(argv[1]));
           close(pipefd[1]);          /* Reader will see EOF */
           wait(NULL);                /* Wait for child */
           exit(EXIT_SUCCESS);
       }
   }
```

SEE ALSO

> **fork**(2), **read**(2), **socketpair**(2), **splice**(2), **tee**(2), **vmsplice**(2), **write**(2), **popen**(3), **pipe**(7)

COLOPHON

> This page is part of release 5.02 of the Linux *man-pages* project. A description of the project, information about reporting bugs, and the latest version of this page, can be found at https://www.kernel.org/doc/man−pages/.