NAME

pipe, pipe2 − create pipe

SYNOPSIS

#include <unistd.h>

int pipe(int *pipefd***[2]);**

#define _GNU_SOURCE
#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_NONBLOCK
Set the **O_NONBLOCK** file status flag on the two new open file descriptions. Using this flag saves extra calls to **fcntl**(2) to achieve the same result.

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.

RETURN VALUE

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

ERRORS

EFAULT
pipefd is not valid.

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

EMFILE
Too many file descriptors are in use by the process.

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

VERSIONS

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

CONFORMING TO

pipe(): POSIX.1-2001.

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 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.

#include <sys/wait.h>

```
#include <assert.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;

   assert(argc == 2);

   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), **write**(2), **popen**(3), **pipe**(7)

COLOPHON

This page is part of release 3.22 of the Linux *man-pages* project. A description of the project, and information about reporting bugs, can be found at http://www.kernel.org/doc/man-pages/.