

Join the Stack Overflow Community

Stack Overflow is a community of 7.1 million programmers, just like you, helping each other.
Join them; it only takes a minute:

[Sign up](#)

How to send a simple string between two programs using pipes?

I tried searching on the net, but there are hardly any resources. A small example would suffice.

EDIT I mean, two different C programs communicating with each other. One program should send "Hi" and the other should receive it. Something like that.

[c](#) [unix](#) [pipe](#)

edited May 6 '10 at 21:13

asked May 6 '10 at 21:06
[user244333](#)

- 1 Presumably you don't mean something like `ls | grep ".o"` ? Perhaps a bit more explanation of what you do mean would help... – [Jerry Coffin](#) May 6 '10 at 21:09
- 9 Come on man... a little effort. Google "c pipes example code". The first result is exact: tldp.org/LDP/lpg/node11.html – [Stephen](#) May 6 '10 at 21:12
- 4 I want communication between two completely different programs. I was not able to find a resource for that. – [user244333](#) May 6 '10 at 21:16
- 1 If you are not forking a process, then you need to look at "named pipes". – [Judge Maygarden](#) May 7 '10 at 13:49

6 Answers

A regular pipe can only connect two related processes. It is created by a process and will vanish when the last process closes it.

A **named pipe**, also called a FIFO for its behavior, can be used to connect two unrelated processes and exists independently of the processes; meaning it can exist even if no one is using it. A FIFO is created using the `mkfifo()` library function.

Example

writer.c

```
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>

int main()
{
    int fd;
    char * myfifo = "/tmp/myfifo";

    /* create the FIFO (named pipe) */
    mkfifo(myfifo, 0666);

    /* write "Hi" to the FIFO */
    fd = open(myfifo, O_WRONLY);
    write(fd, "Hi", sizeof("Hi"));
    close(fd);

    /* remove the FIFO */
    unlink(myfifo);
}
```

```
    return 0;
}
```

reader.c

```
#include <fcntl.h>
#include <stdio.h>
#include <sys/stat.h>
#include <unistd.h>

#define MAX_BUF 1024

int main()
{
    int fd;
    char * myfifo = "/tmp/myfifo";
    char buf[MAX_BUF];

    /* open, read, and display the message from the FIFO */
    fd = open(myfifo, O_RDONLY);
    read(fd, buf, MAX_BUF);
    printf("Received: %s\n", buf);
    close(fd);

    return 0;
}
```

Note: Error checking was omitted from the above code for simplicity.

answered May 7 '10 at 16:08



[jschmier](#)

11.5k 4 36 64

3 What is considered *related processes*? – [Pithikos](#) Aug 29 '14 at 18:00

3 Probably processes which are related via one or more parent/child relations (e.g. includes siblings). The common ancestor would have created the two ends of the pipe. Unrelated processes lack that common ancestor. – [MSalters](#) Nov 5 '14 at 15:58

2 This will not work if the reader kicks off first. A quick fix would be to put the `open()` of the reader inside a loop. However +1 because you provide a two programs example. – [gsamaras](#) Nov 15 '14 at 12:12

I take it this example needs some tweaking to work on windows? `unistd.h` being POSIX and all... – [David Karlsson](#) Mar 4 '15 at 12:01

Yes, it will need tweaking for Windows. The [Wikipedia article on named pipes](#) discusses some of the Unix/Windows differences and a quick [Google search](#) can help with the Windows implementation. – [jschmier](#) Mar 4 '15 at 16:46

From [Creating Pipes in C](#), this shows you how to fork a program to use a pipe. If you don't want to fork(), you can use [named pipes](#).

In addition, you can get the effect of `prog1 | prog2` by sending output of `prog1` to stdout and reading from `stdin` in `prog2`. You can also read stdin by opening a file named `/dev/stdin` (but not sure of the portability of that).

```
/******
Excerpt from "Linux Programmer's Guide - Chapter 6"
(C)copyright 1994-1995, Scott Burkett
*****
MODULE: pipe.c
*****/

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

int main(void)
{
    int    fd[2], nbytes;
    pid_t  childpid;
    char   string[] = "Hello, world!\n";
    char   readbuffer[80];

    pipe(fd);

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

    if(childpid == 0)
    {
        /* Child process closes up input side of pipe */
        close(fd[0]);

        /* Send "string" through the output side of pipe */
        write(fd[1], string, (strlen(string)+1));
        exit(0);
    }
    else
    {

```

```

/* Parent process closes up output side of pipe */
close(fd[1]);

/* Read in a string from the pipe */
nbytes = read(fd[0], readbuffer, sizeof(readbuffer));
printf("Received string: %s", readbuffer);
}

return(0);
}

```

edited Jul 11 '14 at 16:18



Isa A
753 8 24

answered May 6 '10 at 21:16



Stephen
27.1k 6 42 59

1 Hey Stephen, anyway I can use this code for two different functions? meaning writing to the pipe is done in one function and reading the pipe in another function?? a working code like this would be appreciated. – Mohsin Sep 8 '16 at 15:40

```
dup2( STDIN_FILENO, newfd )
```

And read:

```

char reading[ 1025 ];
int fdin = 0, r_control;
if( dup2( STDIN_FILENO, fdin ) < 0 ){
    perror( "dup2( )" );
    exit( errno );
}
memset( reading, '\0', 1025 );
while( ( r_control = read( fdin, reading, 1024 ) ) > 0 ){
    printf( "<%s>", reading );
    memset( reading, '\0', 1025 );
}
if( r_control < 0 )
    perror( "read( )" );
close( fdin );

```

But, I think that `fcntl` can be a better solution

```
echo "salut" | code
```

edited Aug 14 '11 at 10:05



Shadow Wizard
50.5k 15 93 141

answered Aug 14 '11 at 9:56



mlouk
81 1 1

What one program writes to stdout can be read by another via stdin. So simply, using c, write prog1 to print something using `printf()` and prog2 to read something using `scanf()`. Then just run

```
./prog1 | ./prog2
```

answered May 6 '10 at 21:14



Johan
3,328 24 45

Here's a sample:

```

int main()
{
    char buff[1024] = {0};
    FILE* cvt;
    int status;
    /* Launch converter and open a pipe through which the parent will write to it */
    cvt = popen("converter", "w");
    if (!cvt)
    {
        printf("couldn't open a pipe; quitting\n");
        exit(1)
    }
    printf("enter Fahrenheit degrees: ");
    fgets(buff, sizeof(buff), stdin); /*read user's input */
    /* Send expression to converter for evaluation */
    fprintf(cvt, "%s\n", buff);
    fflush(cvt);
    /* Close pipe to converter and wait for it to exit */
    status=pclose(cvt);
    /* Check the exit status of pclose() */
    if (!WIFEXITED(status))
        printf("error on closing the pipe\n");
    return 0;
}


```



The important steps in this program are:

1. The `popen()` call which establishes the association between a child process and a pipe in the parent.

- 2. The `fprintf()` call that uses the pipe as an ordinary file to write to the child process's stdin or read from its stdout.
- 3. The `pclose()` call that closes the pipe and causes the child process to terminate.


edited Apr 23 '13 at 16:22


 **Keith Pinson**

3,850 5 33 71

answered May 6 '10 at 21:13

 **Preet Sangha**



49.1k 14 97 160

I think this example misses the point of the question, although I grant that the "converter" program is a different program. The first comment addresses communication between completely independent programs that do not have a sibling/parent/second-cousin relationship. — [cmm](#) Mar 9 '15 at 16:19

first, have program 1 write the string to stdout (as if you'd like it to appear in screen). then the second program should read a string from stdin, as if a user was typing from a keyboard. then you run:

```
program_1 | program_2
```

answered May 6 '10 at 21:12

 **Ifagundes**



1,619 2 16 22