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

Ask Question

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

11 Come on man... a little effort. Google "c pipes example code". The first result is exact:

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

7 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 = "/

    /* create the FIFO
    mkfifo(myfifo, 066
```

```
    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 = "/
    char buf[MAX_BUF];

    /* open, read, and
    fd = open(myfifo, 
    read(fd, buf, MAX_
    printf("Received: 
    close(fd);

    return 0;
}
```

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

swered May 7 '10 at 16:08

jschmier
12.8k 5 43 69

5 What is considered *related processes*? – Pithikos Aug 29 '14 at 18:00

5 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

4 This will not work if the reader kicks off first. A quick fix would be to

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 P
 (C)opyright 1994-1995,
 ********************
```

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

int main(void)
{
        int     fd[2],
        pid_t   childp
        char    string
        char    readbu

        pipe(fd);

        if((childpid =
        {
                perror
                exit(1
        }

        if(childpid ==
        {
                /* Chi
                close(

                /* Sen
                write(
                exit(0
        }
        else
        {
                /* Par
                close(

                /* Rea
                nbytes
                printf
        }

        return(0);
}
```

ited Jul 11 '14 at 16:18

Isa A
849 9 27

swered May 6 '10 at 21:16

Stephen
32.8k 6 47 60

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

And read:

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

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

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

ited Aug 14 '11 at 10:05

Shadow Wizard
56.4k 18 107 170

swered 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
```

swered May 6 '10 at 21:14

Johan
3,615 1 27 46

well then you should see this question stackoverflow.com/que

Here's a sample:

```
int main()
{
    char buff[1024] = 
    FILE* cvt;
    int status;
    /* Launch converte
*/
    cvt = popen("conve
    if (!cvt)
    {
        printf("couldn
        exit(1)
    }
    printf("enter Fahr
    fgets(buff, sizeof
    /* Send expression
    fprintf(cvt, "%s\n
    fflush(cvt);
    /* Close pipe to c
    status=pclose(cvt)
    /* Check the exit 
    if (!WIFEXITED(sta
        printf("error 
    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.

ited Apr 23 '13 at 16:22

Keith Pinson

4,469 5 39 84

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

swered May 6 '10 at 21:12

lfagundes
1,816 3 18 22

This answer might be helpful for a future Googler.

```
#include <stdio.h>
#include <unistd.h>

int main(){
     int p, f;
     int rw_setup[2];
     char message[20];
     p = pipe(rw_setup
     if(p < 0){
        printf("An err
        _exit(1);
     }
     f = fork();
```

```
        else{
            printf("Could
        }
        return 0;

    }
```

You can find an advanced two-way pipe call example here.

swered Oct 25 '17 at 2:43

Anjana
301 1 12