

Pipes, Part 1: Introduction to pipes

SufeiZ edited this page on Dec 16, 2014 · 10 revisions

What is a pipe?

A POSIX pipe is almost like its real counterpart - you can stuff bytes down one end and they will appear at the other end in the same order. Unlike real pipes however, the flow is always in the same direction, one file descriptor is used for reading and the other for writing. The `pipe` system call is used to create a pipe.

```
int filedes[2];
pipe (filedes);
printf("read from %d, write to %d\n", filedes[0], filedes[1]);
```

These file descriptors can be used with `read` -

```
// To read...
char buffer[80];
int bytesread = read(filedes[0], buffer, sizeof(buffer));
```

And `write` -

```
write(filedes[1], "Go!", 4);
```

How can I use pipe to communicate with a child process?

A common method of using pipes is to create the pipe before forking.

```
int filedes[2];
pipe (filedes);
pid_t child = fork();
if (child > 0) { /* I must be the parent */
    char buffer[80];
    int bytesread = read(filedes[0], buffer, sizeof(buffer));
    // do something with the bytes read
}
```

The child can then send a message back to the parent:

```
if (child == 0) {
    write(filedes[1], "done", 4);
}
```

Edit

New Page

▼ Pages 51

[Home](#)

[#Example Markdown](#)

[#Informal Glossary](#)

[#Piazza: When And How to Ask For Help](#)

[C Programming, Part 1: Introduction](#)

[C Programming, Part 2: Text Input And Output](#)

[C Programming, Part 3: Common Gotchas](#)

[C Programming, Part 4: Debugging](#)

[Deadlock, Part 1: Resource Allocation Graph](#)

[Deadlock, Part 2: Deadlock Conditions](#)

[File System, Part 1: Introduction](#)

[File System, Part 2: Files are inodes \(everything else is just data...\)](#)


[File System, Part 3: Permissions](#)


[File System, Part 4: Working with directories](#)


[File System, Part 5: Virtual file systems](#)


Show 36 more pages...


Clone this wiki locally


 Clone in Desktop











Can I use pipes inside a single process?

Short answer: Yes, but I'm not sure why you would want to LOL!

Here's an example program that sends a message to itself:

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

int main() {
    int fh[2];
    pipe(fh);
    FILE *reader = fdopen(fh[0], "r");
    FILE *writer = fdopen(fh[1], "w");
    // Hurrah now I can use printf rather than using low-level read() write()
    printf("Writing...\n");
    fprintf(writer,"%d %d %d\n", 10, 20, 30);
    fflush(writer);

    printf("Reading...\n");
    int results[3];
    int ok = fscanf(reader,"%d %d %d", results, results + 1, results + 2);
    printf("%d values parsed: %d %d %d\n", ok, results[0], results[1], results[2])

    return 0;
}
```

The problem with using a pipe in this fashion is that writing to a pipe can block i.e. the pipe only has a limited buffering capacity. If the pipe is full the writing process will block! The maximum size of the buffer is system dependent; typical values from 4KB upto 128KB.

```
int main() {
    int fh[2];
    pipe(fh);
    int b = 0;
    #define MSG "....."
    while(1) {
        printf ("%d\n",b);
        write(fh[1], MSG, sizeof(MSG))
        b+=sizeof(MSG);
    }
    return 0;
}
```

See [Pipes, Part 2: Pipe programming secrets](#)

Legal and Licensing information: Unless otherwise specified, submitted content to the wiki must be original work (including text, java code, and media) and you provide this material under a [Creative Commons License](#). If you are not the copyright holder, please give proper attribution and credit to existing content and ensure that you have license to include the materials.

