Implementation of Pipe under C in Linux

Tushar B. Kute,
http://tusharkute.com

Pipe

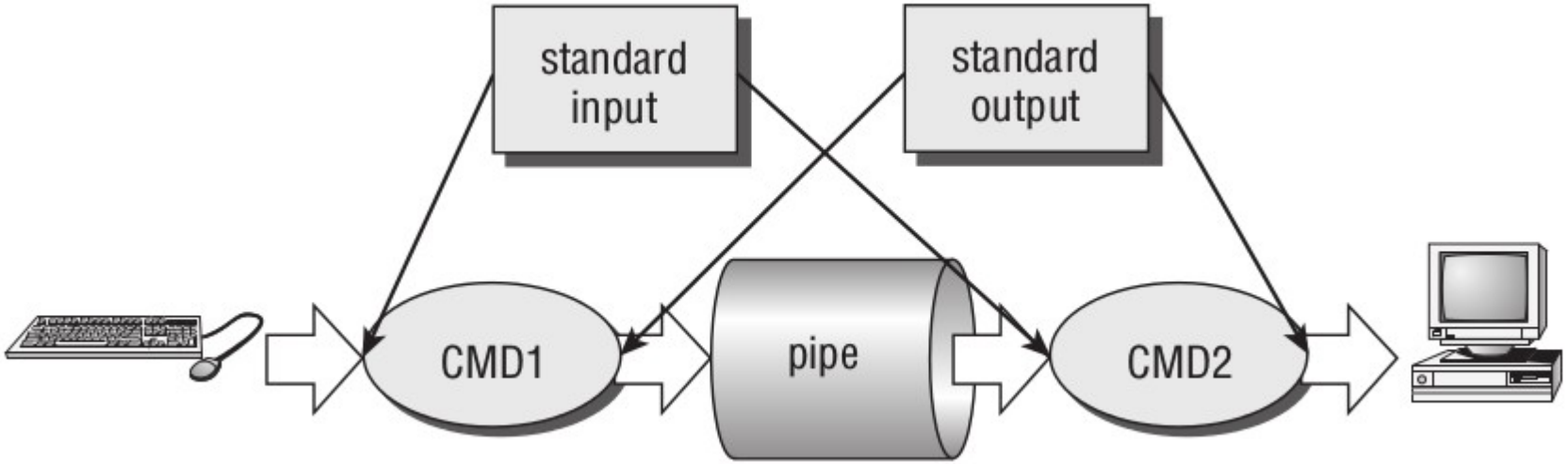
- We use the term pipe to mean connecting a data flow from one process to another.
- Generally you attach, or pipe, the output of one process to the input of another.
- Most Linux users will already be familiar with the idea of a pipeline, linking shell commands together so that the output of one process is fed straight to the input of another.
- For shell commands, this is done using the pipe character to join the commands, such as

```
cmd1 | cmd2
```

Pipes in Commands

- The output of first command is given as input to the second command.
- Examples:
  - `ls | wc`
  - `who | sort`
  - `cat file.txt | sort | wc`

How this works?



The pipe call

- The lower-level pipe function provides a means of passing data between two programs, without the overhead of invoking a shell to interpret the requested command. It also gives you more control over the reading and writing of data.
- The pipe function has the following prototype:

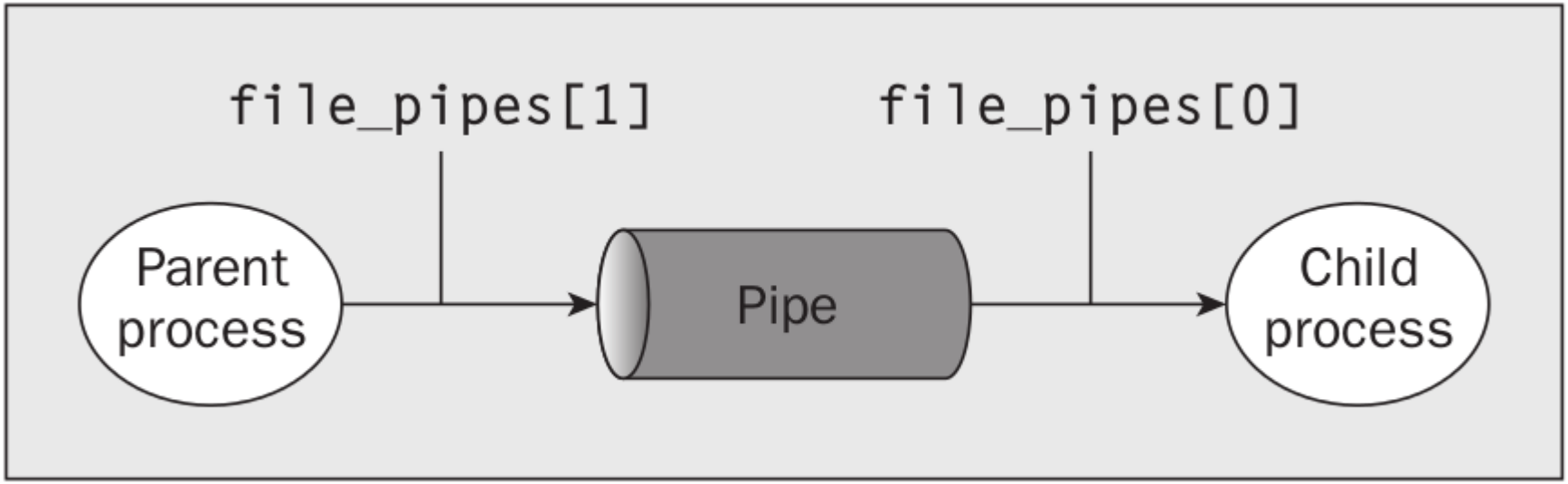
```
#include <unistd.h>

int pipe(int file_descriptor[2]);
```

- pipe is passed (a pointer to) an array of two integer file descriptors. It fills the array with two new file descriptors and returns a zero. On failure, it returns -1 and sets errno to indicate the reason for failure.

File descriptors

- The two file descriptors returned are connected in a special way.
- Any data written to file_descriptor[1] can be read back from file_descriptor[0] . The data is processed in a first in, first out basis, usually abbreviated to FIFO.
- This means that if you write the bytes 1 , 2 , 3 to file_descriptor[1] , reading from file_descriptor[0] will produce 1 , 2 , 3 . This is different from a stack, which operates on a last in, first out basis, usually abbreviated to LIFO.

The Pipe



The write system call

```
#include <unistd.h>
size_t write(int fildes, const void *buf,
size_t nbytes);
```

- It arranges for the first nbytes bytes from buf to be written to the file associated with the file descriptor fildes.
- It returns the number of bytes actually written. This may be less than nbytes if there has been an error in the file descriptor. If the function returns 0, it means no data was written; if it returns –1, there has been an error in the write call.

The read system call

```
#include <unistd.h>
size_t read(int fildes, void *buf, size_t
nbytes);
```

- It reads up to nbytes bytes of data from the file associated with the file descriptor fildes and places them in the data area buf.
- It returns the number of data bytes actually read, which may be less than the number requested. If a read call returns 0, it had nothing to read; it reached the end of the file. Again, an error on the call will cause it to return –1.

Example: pipe1.c

```
#include<stdio.h>
#include<string.h>
int main()
{
    int file_pipes[2], data_pro;
    const char data[] = "Hello Tushar";
    char buffer[20];
    if (pipe(file_pipes) == 0)
    {
        data_pro = write(file_pipes[1], data, strlen(data));
        printf("Wrote %d bytes\n", data_pro);
        data_pro = read(file_pipes[0], buffer, 20);
        printf("Read %d bytes: %s\n", data_pro, buffer);
    }
    return 0;
}
```

Output

```
tushar@tushar-laptop ~ $ gcc pipe1.c
tushar@tushar-laptop ~ $ ./a.out
Wrote 12 bytes
Read 12 bytes: Hello Tushar
tushar@tushar-laptop ~ $
```

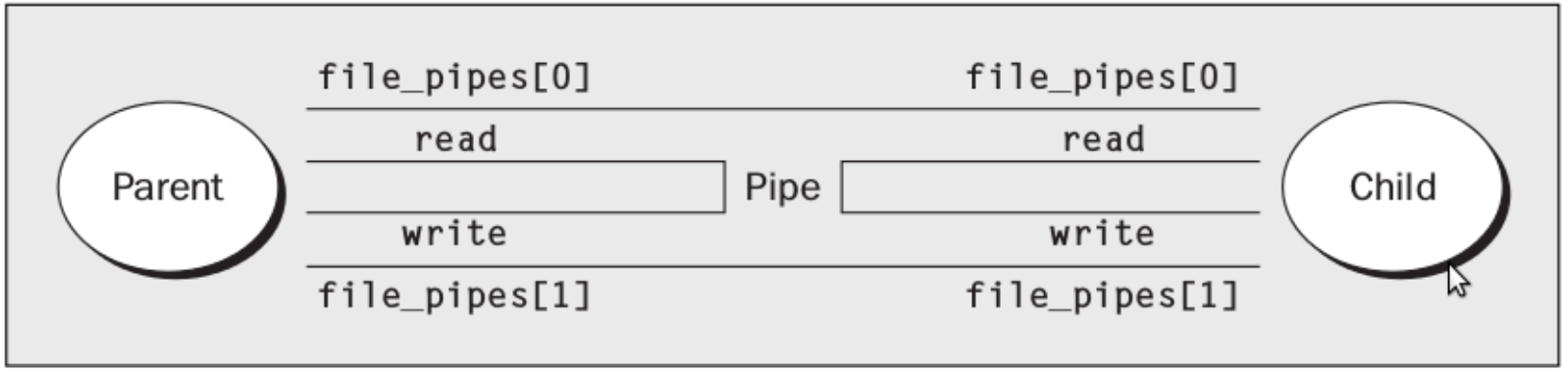
Example-2: pipe2.c (Add fork to pipe)

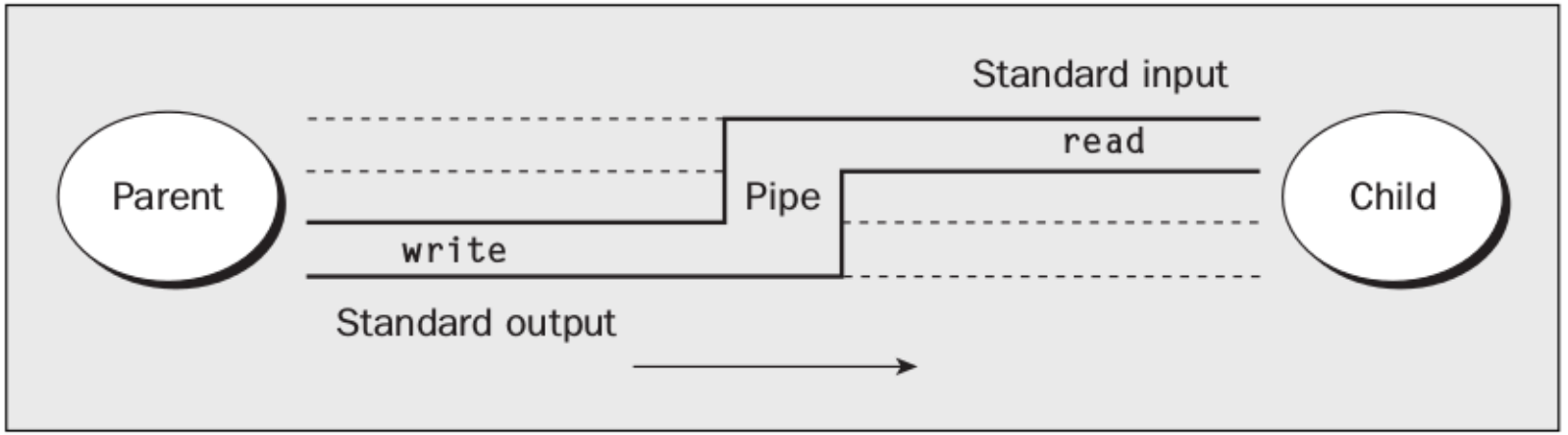
```
#include<stdio.h>
#include<string.h>
int main()
{
    int file_pipes[2], data_pro, pid;
    const char data[] = "Hello Tushar", buffer[20];
    pipe(file_pipes);
    pid = fork();
    if (pid == 0)
    {
        data_pro = write(file_pipes[1], data, strlen(data));
        printf("Wrote %d bytes\n", data_pro);
    }
    else
    {
        data_pro = read(file_pipes[0], buffer, 20);
        printf("Read %d bytes: %s\n", data_pro, buffer);
    }
    return 0;
}
```

Output

```
tushar@tushar-laptop ~ $ gcc pipe2.c
tushar@tushar-laptop ~ $ ./a.out
Read 12 bytes: Hello Tushar
Wrote 12 bytes
tushar@tushar-laptop ~ $
```

Uses of pipes

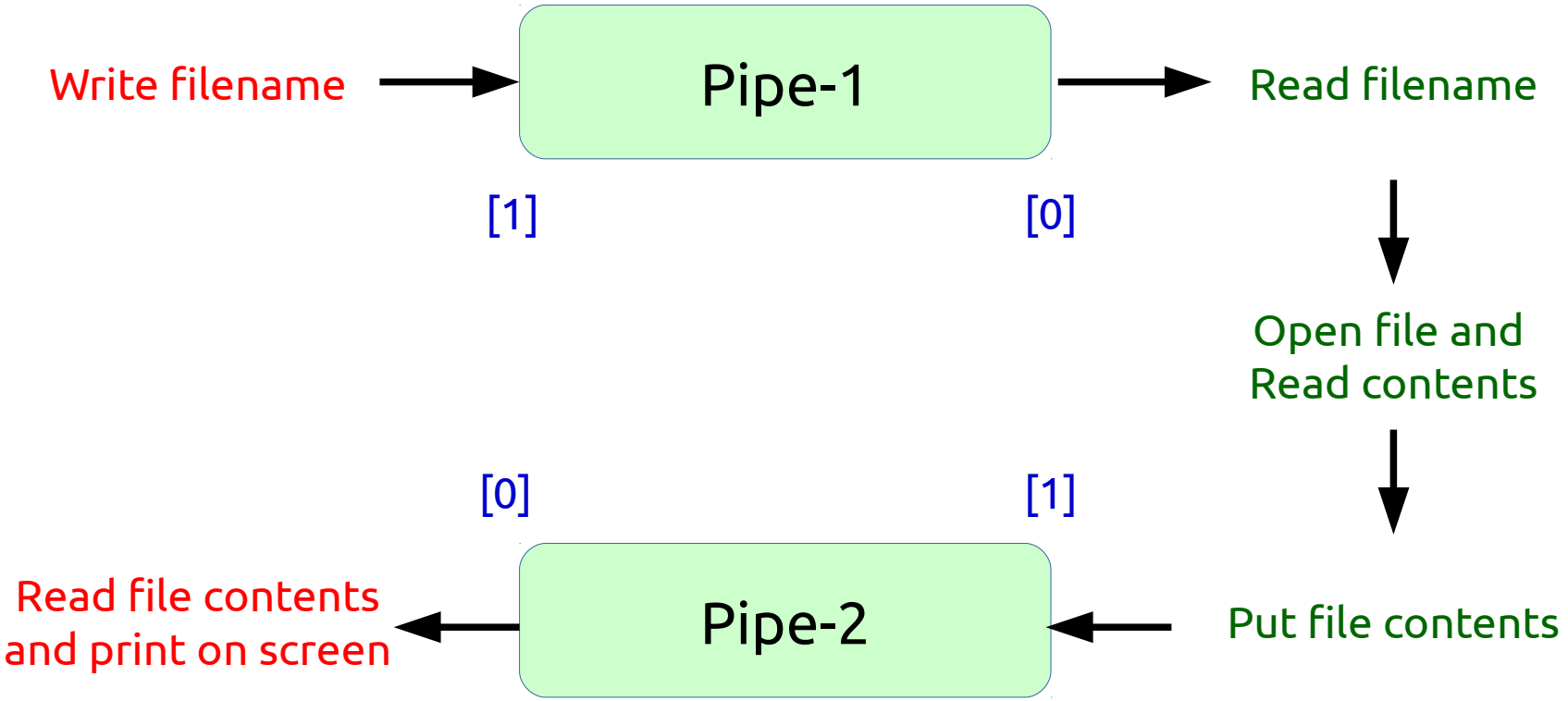




Problem Statement

- Implement using Pipe: Full duplex communication between parent and child processes. Parent process writes a pathname of a file (the contents of the file are desired) on one pipe to be read by child process and child process writes the contents of the file on second pipe to be read by parent process and displays on standard output.

How to do it?


Write filename
Pipe-1
Read filename
[1]
[0]
Open file and
Read contents
[0]
[1]
Read file contents
and print on screen
Pipe-2
Put file contents

hello.txt

```
Hi friends,

How are you...?

My name is Tushar B Kute.
```

Solution: pipe3.c

```
pipe(file_pipe2);                                   /* Second pipe created */
if (pipe(file_pipe1) == 0)                          /* first pipe created */
        fork_result = fork();                       /* Child process created */
if (fork_result == 0)  {
        write(file_pipe1[1], filename, strlen(filename));
        printf("CHILD PROCESS: Wrote filename...\n");
        read(file_pipe2[0], ch, 1024);
        printf("CHILD PROCESS: Its contents are...\n %s", ch);
}
else {
        read(file_pipe1[0], buffer, 10);
        printf("PARENT PROCESS: Read filename %s ...\n", buffer);
        fp = fopen(buffer,"r");
        while(!feof(fp)) {
                ch[count] = fgetc(fp);
                count++;
        }
        fclose(fp);
        write(file_pipe2[1], ch, strlen(ch));
        printf("PARENT PROCESS: The Contents are written ...\n");
```

Output

```
tushar@tushar-laptop ~ $ ./a.out
CHILD PROCESS: Wrote filename...
PARENT PROCESS: Read filename hello.txt ...
PARENT PROCESS: The Contents are written ...
CHILD PROCESS: Its contents are...
 Hi friends,
How are you...?
My name is Tushar B Kute.
tushar@tushar-laptop ~ $
```

Thank you

Web Resources
http://tusharkute.com

Blogs
http://digitallocha.blogspot.in
http://kyamputar.blogspot.in

tushar@tusharkute.com