Implementation of FIFO in C under Linux

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

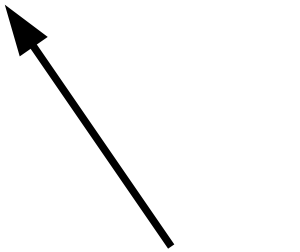
tusharkute.com

FIFO

- It is a named pipe, a method for passing information from one computer process to other processes using a pipe or message holding place that is given a specific name. Unlike a regular pipe, a named pipe can be used by processes that do not have to share a common process origin and the message sent to the named pipe can be read by any authorized process that knows the name of the named pipe.
- A named pipe is sometimes called a "FIFO" (first in, first out) because the first data written to the pipe is the first data that is read from it.

Create FIFO using in Commands

- **`mkfifo filename`**
- **`mknod filename p`**



Pipe

Example:

```
tushar@tushar-laptop ~/module $ mkfifo myfifo
tushar@tushar-laptop ~/module $ ls -l
total 28
-rw-r--r-- 1 tushar tushar  596 Jan  1 23:26 hello.c
-rw-r--r-- 1 tushar tushar 2624 Jan  1 23:35 hello.ko
-rw-r--r-- 1 tushar tushar  713 Jan  1 23:35 hello.mod.c
-rw-r--r-- 1 tushar tushar 1768 Jan  1 23:35 hello.mod.o
-rw-r--r-- 1 tushar tushar 1428 Jan  1 23:35 hello.o
-rw-r--r-- 1 tushar tushar  156 Jan  1 23:35 Makefile
-rw-r--r-- 1 tushar tushar   36 Jan  1 23:54 modules.order
-rw-r--r-- 1 tushar tushar    0 Jan  1 23:35 Module.symvers
prw-r--r-- 1 tushar tushar    0 Mar 11 21:54 myfifo
tushar@tushar-laptop ~/module $
```

Example:

```
tushar@tushar-laptop ~/module $ mknod myfifo p
tushar@tushar-laptop ~/module $ ls -l
total 28
-rw-r--r-- 1 tushar tushar  596 Jan  1 23:26 hello.c
-rw-r--r-- 1 tushar tushar 2624 Jan  1 23:35 hello.ko
-rw-r--r-- 1 tushar tushar  713 Jan  1 23:35 hello.mod.c
-rw-r--r-- 1 tushar tushar 1768 Jan  1 23:35 hello.mod.o
-rw-r--r-- 1 tushar tushar 1428 Jan  1 23:35 hello.o
-rw-r--r-- 1 tushar tushar  156 Jan  1 23:35 Makefile
-rw-r--r-- 1 tushar tushar   36 Jan  1 23:54 modules.order
-rw-r--r-- 1 tushar tushar    0 Jan  1 23:35 Module.symvers
prw-r--r-- 1 tushar tushar    0 Mar 11 21:57 myfifo
tushar@tushar-laptop ~/module $
```

Accessing a FIFO

- First, try reading the (empty) FIFO:

```
cat < my_fifo
```

- Now try writing to the FIFO. You will have to use a different terminal because the first command will now be hanging, waiting for some data to appear in the FIFO.

```
echo “Hello World” > my_fifo
```

Accessing a FIFO

```
tushar@tushar-laptop ~ $ cat myfifo &
[5] 4584
tushar@tushar-laptop ~ $ echo "Hello World..." > myfifo
Hello World...
tushar@tushar-laptop ~ $
```

```
tushar@tushar-laptop ~ $ echo "Hello World..." > myfifo &
[4] 4650
tushar@tushar-laptop ~ $ cat myfifo
Hello World...
```

Create a FIFO in C

```
#include <sys/types.h>
#include <sys/stat.h>
int mkfifo(const char *filename, mode_t
mode);
int mknod(const char *filename, mode_t mode |
S_IFIFO, (dev_t) 0);
```

- Like the mknod and mkfifo command, you can use the mknod function for making many special types of files. Using a dev_t value of 0 and ORing the file access mode with S_IFIFO is the only portable use of this function that creates a named pipe.

Create a FIFO in C

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/stat.h>
int main()
{
    int res = mkfifo("myfifo", 0766);
    if (res == 0)
        printf("FIFO created...\n");
    return(0);
}
```

Output

```
tushar@tushar-laptop ~/module $ gcc fifo1.c -o fifo
tushar@tushar-laptop ~/module $ ./fifo
FIFO created...
tushar@tushar-laptop ~/module $ ls -l
total 40
-rwxr-xr-x 1 tushar tushar 7330 Mar 11 22:16 fifo
-rw-r--r-- 1 tushar tushar  170 Mar 11 22:16 fifo1.c
-rw-r--r-- 1 tushar tushar  596 Jan  1 23:26 hello.c
-rw-r--r-- 1 tushar tushar 2624 Jan  1 23:35 hello.ko
-rw-r--r-- 1 tushar tushar  713 Jan  1 23:35 hello.mod.c
-rw-r--r-- 1 tushar tushar 1768 Jan  1 23:35 hello.mod.o
-rw-r--r-- 1 tushar tushar 1428 Jan  1 23:35 hello.o
-rw-r--r-- 1 tushar tushar  156 Jan  1 23:35 Makefile
-rw-r--r-- 1 tushar tushar   36 Jan  1 23:54 modules.order
-rw-r--r-- 1 tushar tushar    0 Jan  1 23:35 Module.symvers
prwxr--r-- 1 tushar tushar    0 Mar 11 22:16 myfifo
tushar@tushar-laptop ~/module $
```

Opening a FIFO

- Unlike unnamed Pipe, FIFO needs to be opened for reading and writing so the file descriptors can be used alongwith it.

```
fd = open(const char *path, O_RDONLY);
fd = open(const char *path, O_WRONLY);
```

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

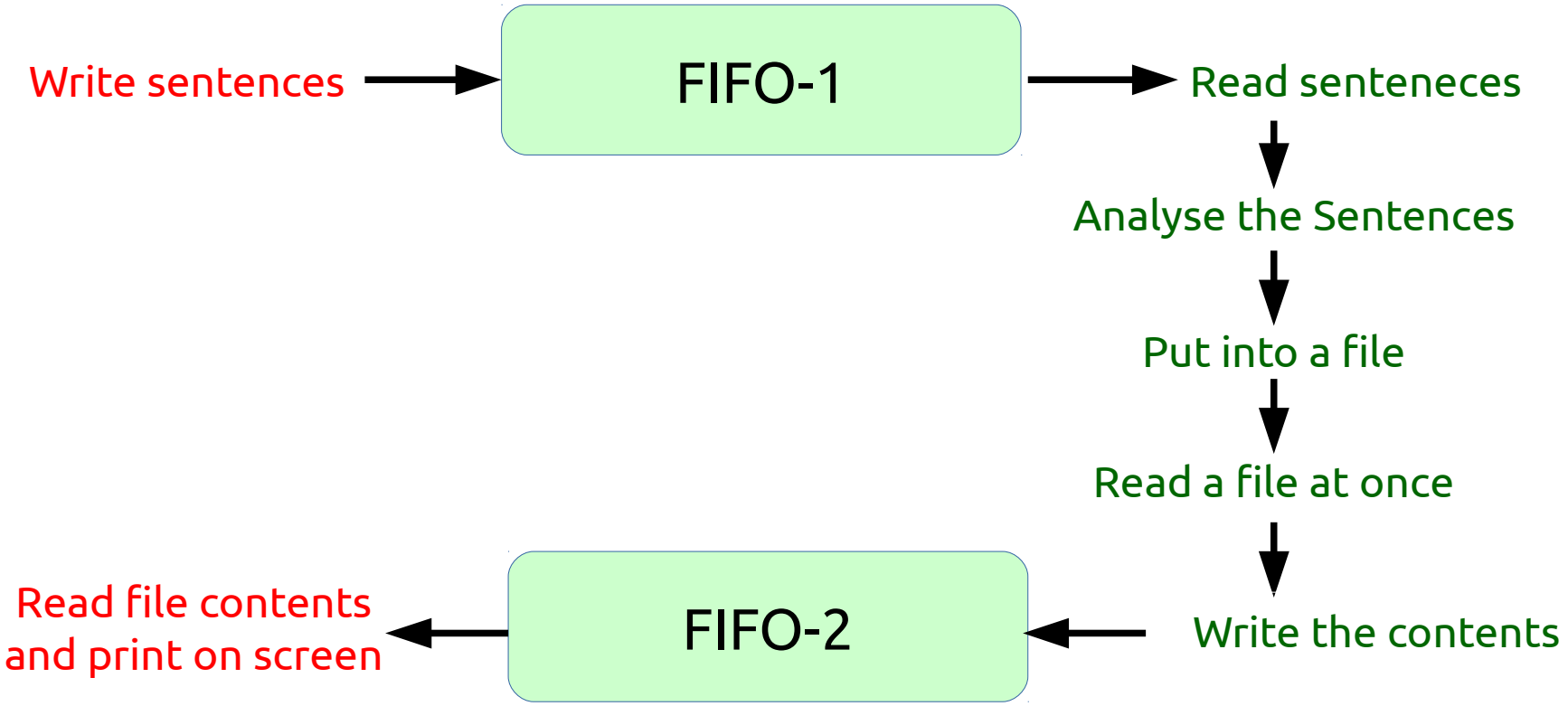
```
#include <unistd.h>
size_t read(int fildes, void *buf, size_t
nbytes);
```

- 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.

Problem Statement

- Implement full duplex communication between two independent processes using FIFO. First process accepts sentences and writes on one pipe to be read by second process and second process counts number of characters, number of words and number of lines in accepted sentences, writes this output in a text file and writes the contents of the file on second pipe to be read by first process and displays on standard output.

How to do it?



writer.c

```
    int fd,fd1;
char * myfifo1 = "myfifo1";
char * myfifo2 = "myfifo2";
char buf[512];
mkfifo(myfifo1, 0666);
mkfifo(myfifo2, 0777);
fd = open(myfifo1, O_WRONLY);
write(fd, "Hello friends.. \nWelcome..\nI am Tushar B Kute", 55);

printf("Data wrote in FIFO1 by writer\n");
close(fd);
fd1 = open(myfifo2, O_RDONLY);
read(fd1, buf, sizeof(buf));
printf("Data received by FIFO2 by writer\n");
printf("%s",buf);
close(fd1);
```

reader.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