Jason D. Bajade

**Pipes and Named Pipes**

Pipe is a feature provided by Linux and other Unices to allow separate processes to communicate without explicitly designing them to do so. Pipes provides solutions for inter-process communication. At its simplest, inter-process communication can be achieved by using the character | (pipe) in Linux.

A very simple example would be:

ls | grep x

“*When bash examines the command line, it finds the vertical bar character | that separates the two commands. Bash and other shells run both commands, connecting the output of the first to the input of the second. The ls program produces a list of files in the current directory, while the grep program reads the output of ls and prints only those lines containing the letter x.”*

(Introduction to Named Pipes by Vaught, A.)

The given above is an example of an *unnamed pip*e or a *regular pipe*. The pipe only exists in the kernel and cannot be accessed by the process that created it, which is the bash shell for this instance. The first process run by the program is the parent process that creates child processes that execute the program. A regular pipe works only with two related processes and vanishes immediately after the execution of the last process.

Another example of a pipe is the named pipe which is also called a FIFO which stands “First In, First Out”. The things that make it distinct from regular pipes is that, first, they “exist as a device special file in the file system; next, processes with different parent process can share data; and lastly, named pipe remains in the file system when all I/O is done sharing processes for future usage. With FIFO, the order of bytes going in follows the same order as it go out. Named pipe can be created in Linux by calling the mkfifo() function in C and must be explicitly opened and closed. Any process that knows the name of the pipe will be able use it to send or receive data if the access mode allows it to do so. open() function is used to access the named pipe which allows the execution of other I/O functions such as read(), write() and close().

The following code demonstrates a very basic example of named pipes:

**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;

}

Example: “How to send a simple string between two programs using pipes?” from StackOverflow.com

In this example, *writer.c* sends “Hi” and *reader.c* receives it and prints out the sent word which is “Hi”. This indeed shows that pipes and named pipes cater the problem of inter-process communication.

**Resources**

[1] (1996). *Named Pipes (FIFOs - First In First Out).* Retrieved from http://www.tldp.org/LDP/lpg/node15.html.

[2]StackOverflow.com*. How to send a simple string between two programs using pipes?* Retrieved from http://stackoverflow.com/questions/2784500/how-to-send-a-simple-string-between-two-programs-using-pipes

[3] Vaught, A. (1997). *Introduction to Named Pipes.* Retrieved from http://www.linuxjournal.com/article/2156?page=0,0

[4] Buffalo, V. *Using Names Pipes and Process Substitution in Bioinformatics*. Retrieved from http://www.vincebuffalo.com/blog/2013/08/08/using-names-pipes-and-process-substitution-in-bioinformatics.html

[5] IBM Knowledge Center. *Using named pipes*. Retrieved from https://www.ibm.com/support/knowledgecenter/