# NAMED PIPES. IMPLEMENTATION IN JAVA

**Ionița Vladimir**

Universitatea Tehnică a Moldovei

***Abstract:*** *There are times when it's extremely useful to be able to pass some data between different programs running on the same system. For example, you might have multiple programs forming part of the same package, and they need to share some important information or work together to process something. There are several ways to do it. One of the fundamental features operating systems can offer for solving this is pipes. The fact that pipes semantics differ substantially over different operating system creates some troubles for cross platform languages as Java.*

***Keywords:*** *pipes, named pipes, unix, windows, java.*

## 1. Introduction

In computer programming, a pipe is a method used to pass information from one program process to another. Unlike other types of inter-process communication, a pipe only offers one-way communication by passing a parameter or output from one process to another. The information that is passed through a pipe is held by the system until it can be read by the receiving process. [1]

The major disadvantage of pipes is that they can be used only by one process (there are readers and writers within the same process) or the processes which share the same file descriptor table (normally the processes and the child processes or threads created by them). [2]

A named pipe is a one-way or duplex pipe that provides communication between the pipe server and some pipe clients. [3] It is also known as a FIFO (first in, first out) because the first data written to the pipe is the first data that is read from it.

A named pipe is an extension over the traditional pipe. Unlike a regular pipe, it can be used by processes that do not have to share a common process origin. The message sent to the named pipe can be read by any authorized process that knows the name of the named pipe. [4] This allows tools quite narrow in their function to be combined in complex ways. [5]

The "name" of a named pipe is actually a file name, within the file system (Unix) or outside (Windows). A traditional pipe is "unnamed" because it exists anonymously and persists only for as long as the process is running. A named pipe is system-persistent and exists beyond the life of the process and must be deleted once it is no longer being used.

Every instance of a named pipe shares the same name but each instance has its own buffers and handles. These instances also provide a separate medium for communication between the client and server, allowing the use of the same named pipe for multiple pipe clients. [3]

Despite the fact named pipes on both Unix and Windows systems are used for the same purpose - inter-process communication (IPC), the semantics differ substantially.

## 2. Named Pipes in Unix

Instead of a conventional, unnamed, shell pipeline, a named pipeline makes use of the filesystem. It is explicitly created using *mkfifo()* or *mknod()*, and two separate processes can access the pipe by name - one process can open it as a reader, and the other as a writer.

One way to create a named pipe is using *mkfifo* command. The syntax of this command is presented in the figure 1.

```
mkfifo [OPTIONS] NAME
```

Figure 1 *mkfifo* command syntax

To simply test the command, you can run the code from the figure 2 in a Shell.

```
$ mkfifo fifo
```

Figure 2 Example of creating a named pipe using *mkfifo* command

The other way to create a pipe is using mknod() command. mknod is used to create a block or character special files. The syntax of this command is presented in figure 3.

```
$ mknod [OPTION] NAME TYPE
```

Figure 3 *mknod* command syntax

In order to create a named pipe using *mknod* command, you can run the code from the figure 4 in a Shell. The `p` corresponds to the file type `pipe`.

```
$ mknod fifo1 p
```

Figure 4 Example of creating a named pipe using *mknod* command

To write (see figure 5) or read (see figure 6) from a named pipe we can use the *cat* command. You can experiment with *cat* command using two terminals. Open the first terminal and write:

```
$ cat > fifo
Experiment with cat command
Second Line
```

Figure 5 Writing to a named pipe.

Now you can open the second terminal, go to the directory containing the named pipe `fifo` and write:

```
$ cat fifo
```

Figure 6 Reading from a named pipe.

You will notice the above two lines displayed there. If you keep writing to the first terminal, you will notice that every time you press enter, the corresponding line appears in the second terminal.

Finally, when you need to delete a pipe, use the command from the figure 7.

```
$ rm fifo
```

Figure 7 Deleting a named pipe.

A named pipe can be used to transfer information from one application to another without the use of an intermediate temporary file. For example you can pipe the output of `gzip` command into a named pipe, then load the uncompressed data into a MySQL table. Without this named pipe one would need to write out the entire uncompressed version of the `gz` file before loading it into MySQL. Writing the temporary file is both time consuming and results in more I/O and less free space on the hard drive.

## 3. Named Pipes in Windows

In Windows, the design of named pipes is based towards client-server communication, and they work much like sockets, other than the usual read and write operations. Windows named pipes also support an explicit "passive" mode for server applications.

A named pipe can be accessed much like a file. Win32 SDK functions CreateFile, ReadFile, WriteFile and CloseHandle open, read from, write to, and close a pipe, respectively.

A big disadvantage is that there is no command line interface like in Unix.

Named pipes cannot be mounted within a normal filesystem, unlike in Unix. Also unlike their Unix counterparts, named pipes are volatile (removed after the last reference to them is closed). Every pipe is placed in the root directory of the named pipe filesystem (NPFS), mounted under the special path \\.\pipe\ (that is, a pipe named "foo" would have a full path name of \\.\pipe\foo). Anonymous pipes used in pipelining are actually named pipes with a random name.

Here's a quick overview of the steps required to create and use a simple named pipe to send data from a server program to a client program. [6]

Server program:

1. Call CreateNamedPipe(..) to create an instance of a named pipe.
2. Call ConnectNamedPipe(..) to wait for the client program to connect.
3. Call WriteFile(..) to send data down the pipe.
4. Call CloseHandle(..) to disconnect and close the pipe instance.

Client program:

1. Call CreateFile(..) to connect to the pipe.
2. Call ReadFile(..) to get data from the pipe.
3. Output data to the screen.
4. Call CloseHandle(..) to disconnect from the pipe.

This is a very simple example though, and there are lots more you can do with pipes on Win32.

You can name Win32 pipes almost anything you like, as long as they start with the prefix "\\.\pipe\". In practice it becomes "\\\\.\\pipe\\", since you have two escape backslashes in C++ strings. Everything after that in the name is up to you, so long as you don't use backslashes though, and don't exceed 256 characters in total. [6]

There are two main modes of read/write operation available on pipes: byte stream, and message. The difference is fairly small, but can be very significant depending on your application. [6]

Message mode simply makes a distinction between each set of data sent down the pipe. If a program sends 50 bytes, then 100 bytes, then 40 bytes, the receiving program will receive it in these separate blocks (and will therefore need to read the pipe at least 3 times to receive everything). [6]

On the other hand, byte stream mode lets all the sent data flow continuously. In our example of 50, 100, then 40 bytes, the client could happily receive everything in a single 190-byte chunk. Which mode you choose depends on what your programs need to do. [6]

Summary of named pipes on Microsoft Windows:

1. Intermachine and Intramachine IPC
2. Half-duplex or full-duplex
3. Byte-oriented or Message-oriented
4. Reliable
5. Blocking or Nonblocking read and write (choosable)
6. Standard device I/O handles (FileRead, FileWrite)
7. Namespace used to create handles
8. Inefficient WAN traffic (explicit data transfer request, unlike e.g. TCP/IP sliding window, etc.)
9. Peekable reads (read without removing from pipe's input buffer)

## 4. Java Implementation

Named pipes become a real problem if you want to use them cross platform. Their implementation in Unix and Windows differ substantially.

Firstly, in Windows there is no command line tool for creating named pipes.

Secondly, in Windows named pipes cannot be created as files in a writeable filesystem, they live in a special filesystem and can be created only by using the Win32 API.

If you want to implement Named Pipes in Java, because of the second problem, you have to resort to native code. Of course there is a solution, JNI/JNA seems the industry standard for that.

Now the question becomes a little philosophic. If we want to use Named Pipes, should we use Java? Java is meant to be cross platform, to make things portable. Resorting to native code isn't quite portable.

There a two solutions: either switch to sockets for IPC, probably the best long term solution since it's much more portable, either implement the Named Pipes by yourself, using files for inter-process communication. Of course pipes are more elegant and more efficient than sockets or files, but since we are speaking about Java, there is always a loss in performance.

## BIBLIOGRAPHY

1] C. Janssen, "Pipe," techopedia, [Online]. Available: http://www.techopedia.com/definition/3410/pipe. [Accessed 22 November 2012].

2] "FIFO - NamedPipes: mkfifo, mknod," Programming in Linux, 14 February 2008. [Online]. Available: http://linuxprograms.wordpress.com/2008/02/14/fifo-named-pipes-mkfifo-mknod/. [Accessed 22 November 2012].

3] C. Janssen, "Named Pipe," techopedia, [Online]. Available: http://www.techopedia.com/definition/16416/named-pipe. [Accessed 22 November 2012].

4] M. Rouse, "named pipe," SearchCIO-Midmarket, September 2005. [Online]. Available: http://searchcio-midmarket.techtarget.com/definition/named-pipe. [Accessed 21 November 2012].

5] A. Vaught, "Introduction to Named Pipes," 1 September 1997. [Online]. Available: http://www.linuxjournal.com/article/2156. [Accessed 21 November 2012].

6] P. R. Bloomfield, "Introduction to Win32 Named Pipes (C++)," Avid Insight, 31 March 2012. [Online]. Available: http://avid-insight.co.uk/joomla/component/k2/item/589-introduction-to-win32-named-pipes-cpp. [Accessed 22 November 2012].