Notes on the `xpipe` Package

Jan T. Kim, `jtk@cmp.uea.ac.uk`

August 11, 2009

Abstract

Unnamed pipes allow exchange of data between processes. Differently from communication via (temporary) files or named pipes, an unnamed pipe cannot be interfered with by other processes. The `pipe` function provided by the R base package provides pipes for writing xor reading, but it is not possible to obtain file connections to both the input and the output handles of a pipe.

The objective of the `xpipe` (for "extended pipe" or "pipe through an external process") package is to provide a more complete access to unnamed pipes. This is an initial version, much needs to be done. Comments and suggestions are very welcome.

1 Introduction: The Problem and the Solution

Suppose you have a program `foo` that reads its input from the standard input and writes its output to the standard output, and implements some algorithm or facility that is not provided by `R` or any of its packages. Let us further assume that in your `R` program, you have a variable `foo.data`, holding the data to be processed by `foo`, and you want to assign the result to `foo.result`.

You could write the contents of `foo.data` into some temporary input file `foo.in`, call `foo` with the standard output directed to a temporary output file `foo.out`, and finally read the result from that output file:

```
write(foo.data, file = "foo.in"));
system("foo < foo.in > foo.out");
foo.result <- readLines("foo.out");
```

The trouble with this is that you cannot prevent that something happens to the temporary files before (or while) they are processed in the intended manner. If two `R` processes are running this sequence of calls in the same directory, both will end up with garbled results, and as the number of times you run this sequence increases, the probability that such a clash occurs asymptotically approaches one.

Of course, there are clever measures you can take to prevent this from happening, such as file locking and facilities for generating unique file names, but all these are really workarounds that incur unnecessary overhead. The solution that avoids such problems once and for all is to use an unnamed pipe, known just to the processes involved, and therefore safe from any interference by unrelated other processes. `R` provides a `pipe` function (in the `base` package) for starting a subprocess, but that only allows to *either* writing to the subprocess' standard input *or* reading form the subprocess' standard output — *not* both.

Part of the reason for this is that one `R` process cannot do both, as blocking may occur and cause the flow of execution to stall. The blocking problem is traditionally solved by forking off a child process which feeds input into the (external) subprocess and terminates when finished doing so. The main `R` process then reads the output provided by the subprocess. The `xpipe` function implements this technique.

2 Examples

This example is rather contrived and does not have any practical use, but it depends only on `bc` which should be available on any reasonable Unix variant.

Use `bc` to achieve better precision than `R`'s numeric type:

```
> options(width = 20)
> x <- "11111111111111111111"
> x <- as.numeric(x)
> y1 <- x + x
> y2 <- x + x + 1
> dy <- y2 - y1
> sprintf("imprecise result: dy = %g", dy)


[1] "imprecise result: dy = 0"


> library(xpipe)
> x <- "11111111111111111111"
> y1 <- xpipe("bc", sprintf("%s + %s", x, x))
> y2 <- xpipe("bc", sprintf("%s + %s + 1", x, x))
> dy <- xpipe("bc", sprintf("%s - %s", y2, y1))
> sprintf("precise result: dy = %s", dy)


[1] "precise result: dy = 1"
```

A more compact and efficient variant (all done in one `bc` process, using `bc` variables):

```
> library(xpipe)
> x <- "11111111111111111111"
> bclines <- sprintf("y1 = %s + %s", x, x)
> bclines[2] <- sprintf("y2 = %s + %s + 1", x, x)
> bclines[3] <- "dy = y2 - y1"
> bclines[4] <- "dy"
> dy <- xpipe("bc", bclines)
> sprintf("precise result: dy = %s", dy)


[1] "precise result: dy = 1"
```

3 Future Plans

- The `xpipe` function could be made more flexible and useful by
  - add a verbose argument, for debugging and explanatory / exploratory purposes.
  - capability of returning output obtained from the subprocess in numerical rather than in character type. A `data.frame` would be the next logical target.
  - allowing the input data to be provided by a function rather than by a `character` vector. This would greatly improve efficiency in cases where large amounts of input that can easily be generated are involved (pointless but paradigmatic examples include all natural numbers $1 \geq n \geq 10^8$, this many random numbers etc.etc.).
- Arrange for error flagging if there is no pipe / fork support provided by the host platform.
- As an improved mechanism for detecting `xpipe` support, provide a `xpipe.capabilities` function, modelled after `capabilities` of the `base` package.
- In the future, I'd like to see full pipe and fork support move into the mainstream of `R`, much as `fifo` is there today. Availability of pipes and fork could be reflected by `capabilities`, so `R` programmers would have a reliable and convenient way of telling whether that's available.

4 Current Problems

The most natural way to interface to an `xpipe` would seem to be providing two connections, one for writing and one for reading. Unfortunately, the current code structure does not make this possible for a package; it would be necessary to modify the `R` source code (`src/main/connection.c`). This source file contains a static array `Connections` in which all connections are stored, and there does not seem to be an interface for registering connections set up by a package like `xpipe`.

Unfortunately, this makes it rather difficult to read the output of the external process through `R`'s standard data import functions, such as `read.table`. As a cumbersome but generic solution, it is possible to open an anonymous file, write the lines obtained from `xpipe` into that using `write`, and read the data back from there using `read.table`, `scan`, or any other function.

Appendix

This appendix runs tests (it will do in the future) and thus reflects the amount of functionality provided by `xpipe` on the host on which `Sweave` was run.

A From R to R

This depends only on R, thus minimising dependencies ;-)

```
> library(xpipe)
> set.seed(1)
> num.testvalues <- 1000
> x <- as.character(runif(num.testvalues))
> y <- xpipe("R --vanilla --slave", sprintf("set.seed(1); write(as.character(runif(%d)), file = \"\");",
+     as.integer(num.testvalues)))
> which(x != y)

integer(0)

> y[x != y]

character(0)
```