Capturing stderr and stdout from a subprocess in GNU Guile

Dr. Arne Babenhauserheide

<2020-07-24 Fr>

```
(call-command-with-output-error-to-string "echo 1; echo 2 >&2")
$1 = "1\n"
$2 = "2\n"
```

A common need when calling programs is to capture both the regular output (stdout) and the error output (stderr). While Guile provides lots of facilities to call other processes, this use-case had been unfulfilled for me until now.

I'm sure that there are other tools that already solved the problem, but I did not find them. So now you can use my tool :-) License: LGPLv2 or later.

After seeing the question on IRC, and with some guidance from RhodiumToad, I implemented a procedure (function) that fills the gap:

```
(import (ice-9 rdelim) (ice-9 popen) (rnrs io ports))

(define (call-command-with-output-error-to-string cmd)
  (let* ((err-cons (pipe))
 (port (with-error-to-port (cdr err-cons)
 (λ() (open-input-pipe cmd))))
 (_ (setvbuf (car err-cons) 'block
     (* 1024 1024 16)))
 (result (read-delimited "" port)))
    (close-port (cdr err-cons))
    (values
     result
     (read-delimited "" (car err-cons)))))
```

```
(call-command-with-output-error-to-string "echo 1; echo 2 >&2")
```

Caveats: This deadlocks when the process throws more error output than the buffers in the pipe allow. The buffer is set explicitly to 16 MiB, you might want to vary it depending on your usecase.

I call this beautiful code the flat iron on its head.

For pointers to a cleaner solution, see non-blocking IO to alternate between reading from stdout and stderr.