NAME

IPC::Open3, open3 - open a process for reading, writing, and error handling

SYNOPSIS

```
$pid = open3(\*WTRFH, \*RDRFH, \*ERRFH,
  'some cmd and args', 'optarg', ...);

my($wtr, $rdr, $err);
$pid = open3($wtr, $rdr, $err,
  'some cmd and args', 'optarg', ...);
```

DESCRIPTION

Extremely similar to open2(), open3() spawns the given $cmd and connects RDRFH for reading, WTRFH for writing, and ERRFH for errors. If ERRFH is false, or the same file descriptor as RDRFH, then STDOUT and STDERR of the child are on the same filehandle. The WTRFH will have autoflush turned on.

If WTRFH begins with <&, then WTRFH will be closed in the parent, and the child will read from it directly. If RDRFH or ERRFH begins with >&, then the child will send output directly to that filehandle. In both cases, there will be a dup(2) instead of a pipe(2) made.

If either reader or writer is the null string, this will be replaced by an autogenerated filehandle. If so, you must pass a valid lvalue in the parameter slot so it can be overwritten in the caller, or an exception will be raised.

The filehandles may also be integers, in which case they are understood as file descriptors.

open3() returns the process ID of the child process. It doesn't return on failure: it just raises an exception matching `/^open3:/`. However, `exec` failures in the child are not detected. You'll have to trap SIGPIPE yourself.

Note if you specify `-` as the command, in an analogous fashion to `open(FOO, "-|")` the child process will just be the forked Perl process rather than an external command. This feature isn't yet supported on Win32 platforms.

open3() does not wait for and reap the child process after it exits. Except for short programs where it's acceptable to let the operating system take care of this, you need to do this yourself. This is normally as simple as calling `waitpid $pid, 0` when you're done with the process. Failing to do this can result in an accumulation of defunct or "zombie" processes. See *"waitpid" in perlfunc* for more information.

If you try to read from the child's stdout writer and their stderr writer, you'll have problems with blocking, which means you'll want to use select() or the IO::Select, which means you'd best use sysread() instead of readline() for normal stuff.

This is very dangerous, as you may block forever. It assumes it's going to talk to something like **bc**, both writing to it and reading from it. This is presumably safe because you "know" that commands like **bc** will read a line at a time and output a line at a time. Programs like **sort** that read their entire input stream first, however, are quite apt to cause deadlock.

The big problem with this approach is that if you don't have control over source code being run in the child process, you can't control what it does with pipe buffering. Thus you can't just open a pipe to `cat -v` and continually read and write a line from it.

WARNING

The order of arguments differs from that of open2().