18.5. popen2 — Subprocesses with accessible I/O streams¶

Deprecated since version 2.6: This module is obsolete. Use the `subprocess` module. Check especially the *Replacing Older Functions with the subprocess Module* section.

This module allows you to spawn processes and connect to their input/output/error pipes and obtain their return codes under Unix and Windows.

The `subprocess` module provides more powerful facilities for spawning new processes and retrieving their results. Using the `subprocess` module is preferable to using the `popen2` module.

The primary interface offered by this module is a trio of factory functions. For each of these, if *bufsize* is specified, it specifies the buffer size for the I/O pipes. *mode*, if provided, should be the string `'b'` or `'t'`; on Windows this is needed to determine whether the file objects should be opened in binary or text mode. The default value for *mode* is `'t'`.

On Unix, *cmd* may be a sequence, in which case arguments will be passed directly to the program without shell intervention (as with `os.spawnv()`). If *cmd* is a string it will be passed to the shell (as with `os.system()`).

The only way to retrieve the return codes for the child processes is by using the `poll()` or `wait()` methods on the `Popen3` and `Popen4` classes; these are only available on Unix. This information is not available when using the `popen2()`, `popen3()`, and `popen4()` functions, or the equivalent functions in the `os` module. (Note that the tuples returned by the `os` module's functions are in a different order from the ones returned by the `popen2` module.)

`popen2.popen2`(*cmd*[, *bufsize*[, *mode*]])¶
Executes *cmd* as a sub-process. Returns the file objects `(child_stdout, child_stdin)`.

`popen2.popen3`(*cmd*[, *bufsize*[, *mode*]])¶
Executes *cmd* as a sub-process. Returns the file objects `(child_stdout, child_stdin, child_stderr)`.

`popen2.popen4`(*cmd*[, *bufsize*[, *mode*]])¶

Executes *cmd* as a sub-process. Returns the file objects `(child_stdout_and_stderr, child_stdin)`.

New in version 2.0.

On Unix, a class defining the objects returned by the factory functions is also available. These are not used for the Windows implementation, and are not available on that platform.

class `popen2.Popen3`(*cmd*[, *capturestderr*[, *bufsize*]])¶

This class represents a child process. Normally, `Popen3` instances are created using the `popen2()` and `popen3()` factory functions described above.

If not using one of the helper functions to create `Popen3` objects, the parameter *cmd* is the shell command to execute in a sub-process. The *capturestderr* flag, if true, specifies that the object should capture standard error output of the child process. The default is false. If the *bufsize* parameter is specified, it specifies the size of the I/O buffers to/from the child process.

class `popen2.Popen4`(*cmd*[, *bufsize*])¶

Similar to `Popen3`, but always captures standard error into the same file object as standard output. These are typically created using `popen4()`.

New in version 2.0.

18.5.1. Popen3 and Popen4 Objects¶

Instances of the `Popen3` and `Popen4` classes have the following methods:

`Popen3.poll`()¶
Returns `-1` if child process hasn't completed yet, or its status code (see `wait()`) otherwise.

`Popen3.wait`()¶
Waits for and returns the status code of the child process. The status code encodes both the return code of the process and information about whether it exited using the `exit()` system call or died due to a signal. Functions to help interpret the status code are defined in the `os` module; see section *Process Management* for the `W*()` family of functions.

The following attributes are also available:

`Popen3.fromchild`¶

A file object that provides output from the child process. For `Popen4` instances, this will provide both the standard output and standard error streams.

`Popen3.tochild`¶

A file object that provides input to the child process.

`Popen3.childerr`¶

A file object that provides error output from the child process, if *capturestderr* was true for the constructor, otherwise `None`. This will always be `None` for `Popen4` instances.

`Popen3.pid`¶

The process ID of the child process.

18.5.2. Flow Control Issues¶

Any time you are working with any form of inter-process communication, control flow needs to be carefully thought out. This remains the case with the file objects provided by this module (or the `os` module equivalents).

When reading output from a child process that writes a lot of data to standard error while the parent is reading from the child's standard output, a deadlock can occur. A similar situation can occur with other combinations of reads and writes. The essential factors are that more than `_PC_PIPE_BUF` bytes are being written by one process in a blocking fashion, while the other process is reading from the first process, also in a blocking fashion.

There are several ways to deal with this situation.

The simplest application change, in many cases, will be to follow this model in the parent process:

```
import popen2

r, w, e = popen2.popen3('python slave.py')
e.readlines()
r.readlines()
r.close()
e.close()
w.close()
```

with code like this in the child:

```
import os
import sys

# note that each of these print statements
# writes a single long string

print >>sys.stderr, 400 * 'this is a test\n'
os.close(sys.stderr.fileno())
print >>sys.stdout, 400 * 'this is another test\n'
```

In particular, note that `sys.stderr` must be closed after writing all data, or `readlines()` won't return. Also note that `os.close()` must be used, as `sys.stderr.close()` won't close `stderr` (otherwise assigning to `sys.stderr` will silently close it, so no further errors can be printed).

Applications which need to support a more general approach should integrate I/O over pipes with their `select()` loops, or use separate threads to read each of the individual files provided by whichever `popen*()` function or `Popen*` class was used.

See also

Module `subprocess`

Module for spawning and managing subprocesses.

This Page

- Show Source

Navigation