Programming Python by Mark Lutz

Chapter 5: Parallel System Tools

Forking Processes

PP4E/System/Processes/fork1.py

```
"forks child processes until you type 'q'"

import os

def child():
    print('Hello from child',  os.getpid())
    os._exit(0)  # else goes back to parent loop

def parent():
    while True:
        newpid = os.fork()
        if newpid == 0:
            child()
        else:
            print('Hello from parent', os.getpid(), newpid)
        if input() == 'q': break

parent()
```

PP4E/System/Processes/fork1.py

```
$ python3 fork1.py
Hello from parent 657 658
Hello from child 658

Hello from parent 657 659
Hello from child 659

Hello from parent 657 660
Hello from child 660

Hello from parent 657 661
Hello from child 661

Hello from parent 657 662
Hello from child 662

Hello from parent 657 663
Hello from child 663
q
$
```

PP4E/System/Processes/fork-count.py

```
"""
fork basics: start 5 copies of this program running in parallel with
the original; each copy counts up to 5 on the same stdout stream--forks
copy process memory, including file descriptors; fork doesn't currently
work on Windows without Cygwin: use os.spawnv or multiprocessing on
Windows instead; spawnv is roughly like a fork+exec combination;
"""

import os, time

def counter(count):                                         # run in new process
    for i in range(count):
        time.sleep(1)                                       # simulate real work
        print('[%s] => %s' % (os.getpid(), i))

for i in range(5):
    pid = os.fork()
    if pid != 0:
        print('Process %d spawned' % pid)                   # in parent: continue
    else:
        counter(5)                                          # else in child/new
process
        os._exit(0)                                         # run function and exit

print('Main process exiting.')                              # parent need not wait
```

PP4E/System/Processes/fork-count.py

```
$ python3 fork-count.py
Process 837 spawned
Process 838 spawned
Process 839 spawned
Process 840 spawned
Process 841 spawned
Main process exiting.
$ [837] => 0
[838] => 0
[839] => 0
[840] => 0
[841] => 0
...
[837] => 4
[838] => 4
[839] => 4
[840] => 4
[841] => 4
```

PP4E/System/Processes/fork-exec.py

```
"starts programs until you type 'q'"

import os

parm = 0
while True:
  parm += 1
  pid = os.fork()
  if pid == 0:                                             # copy process
    os.execlp('python3', 'python3', 'child.py', str(parm)) # overlay program
    assert False, 'error starting program'                 # shouldn't return
  else:
    print('Child is', pid)
    if input() == 'q': break
```

PP4E/System/Processes/child.py

```
import os, sys

print('Hello from child', os.getpid(), sys.argv[1])
```

PP4E/System/Processes/fork-exec.py

```
$ python3 fork-exec.py
Child is 897
('Hello from child', 897, '1')
q

$ python3 fork-exec.py
Child is 929
Hello from child 929 1

Child is 930
Hello from child 930 2

Child is 931
Hello from child 931 3

Child is 932
Hello from child 932 4

Child is 933
Hello from child 933 5
q
```

PP4E/System/Processes/pipe1.py

```
import os, time

def child(pipeout):
  zzz = 0
  while True:
    time.sleep(zzz)                            # make parent wait
    msg = ('Spam %03d' % zzz).encode()         # pipes are binary bytes
    os.write(pipeout, msg)                     # send to parent
    zzz = (zzz+1) % 5                          # goto 0 after 4

def parent():
  pipein, pipeout = os.pipe()                  # make 2-ended pipe
  if os.fork() == 0:                           # copy this process
    child(pipeout)                             # in copy, run child
  else:                                        # in parent, listen to pipe
    while True:
      line = os.read(pipein, 32)               # blocks until data sent
      print('Parent %d got [%s] at %s' % (os.getpid(), line, time.time()))

parent()
```

PP4E/System/Processes/pipe1.py

```
$ python3 pipe1.py
Parent 4391 got [b'Spam 000'] at 1332781761.500339
Parent 4391 got [b'Spam 001'] at 1332781762.501251
Parent 4391 got [b'Spam 002'] at 1332781764.502363
Parent 4391 got [b'Spam 003'] at 1332781767.505495
Parent 4391 got [b'Spam 004Spam 000'] at 1332781771.508707
Parent 4391 got [b'Spam 001'] at 1332781772.509974
Parent 4391 got [b'Spam 002'] at 1332781774.512041
Parent 4391 got [b'Spam 003'] at 1332781777.512628
Parent 4391 got [b'Spam 004Spam 000'] at 1332781781.516459
Parent 4391 got [b'Spam 001'] at 1332781782.516878
Parent 4391 got [b'Spam 002'] at 1332781784.51905
...
```

PP4E/System/Processes/pipe2.py

```
# same as pipe1.py, but wrap pipe input in stdio file object
# to read by line, and close unused pipe fds in both processes

import os, time

def child(pipeout):
  zzz = 0
  while True:
    time.sleep(zzz)                             # make parent wait
    msg = ('Spam %03d\n' % zzz).encode()        # pipes are binary in 3.X
    os.write(pipeout, msg)                      # send to parent
    zzz = (zzz+1) % 5                           # roll to 0 at 5

def parent():
  pipein, pipeout = os.pipe()                   # make 2-ended pipe
  if os.fork() == 0:                            # in child, write to pipe
    os.close(pipein)                            # close input side here
    child(pipeout)
  else:                                         # in parent, listen to pipe
    os.close(pipeout)                           # close output side here
    pipein = os.fdopen(pipein)                  # make text mode input file object
    while True:
      line = pipein.readline()[:-1]             # blocks until data sent
      print('Parent %d got [%s] at %s' % (os.getpid(), line, time.time()))

parent()
```

PP4E/System/Processes/pipe2.py

```
$ python3 pipe2.py
Parent 4414 got [Spam 000] at 1332782205.789812
Parent 4414 got [Spam 001] at 1332782206.791198
Parent 4414 got [Spam 002] at 1332782208.792644
Parent 4414 got [Spam 003] at 1332782211.795856
Parent 4414 got [Spam 004] at 1332782215.796212
Parent 4414 got [Spam 000] at 1332782215.796337
Parent 4414 got [Spam 001] at 1332782216.797505
Parent 4414 got [Spam 002] at 1332782218.798857
Parent 4414 got [Spam 003] at 1332782221.800243
Parent 4414 got [Spam 004] at 1332782225.804157
Parent 4414 got [Spam 000] at 1332782225.804266
Parent 4414 got [Spam 001] at 1332782226.805408
Parent 4414 got [Spam 002] at 1332782228.807614
...
```

PP4E/System/Processes/pipes.py

```
"""
spawn a child process/program, connect my stdin/stdout to child process's
stdout/stdin--my reads and writes map to output and input streams of the
spawned program; much like tying together streams with subprocess module;
"""

import os, sys

def spawn(prog, *args):                       # pass progname, cmdline args
  stdinFd  = sys.stdin.fileno()               # get descriptors for streams
  stdoutFd = sys.stdout.fileno()              # normally stdin=0, stdout=1

  parentStdin, childStdout  = os.pipe()       # make two IPC pipe channels
  childStdin,  parentStdout = os.pipe()       # pipe returns (inputfd, outoutfd)
  pid = os.fork()                             # make a copy of this process
  if pid:
    os.close(childStdout)                     # in parent process after fork:
    os.close(childStdin)                      # close child ends in parent
    os.dup2(parentStdin,  stdinFd)            # my sys.stdin copy  = pipe1[0]
    os.dup2(parentStdout, stdoutFd)           # my sys.stdout copy = pipe2[1]
  else:
    os.close(parentStdin)                     # in child process after fork:
    os.close(parentStdout)                    # close parent ends in child
    os.dup2(childStdin,  stdinFd)             # my sys.stdin copy  = pipe2[0]
    os.dup2(childStdout, stdoutFd)            # my sys.stdout copy = pipe1[1]
    args = (prog,) + args
    os.execvp(prog, args)                     # new program in this process
    assert False, 'execvp failed!'            # os.exec call never returns here
```

PP4E/System/Processes/pipes.py

```
if __name__ == '__main__':
    mypid = os.getpid()
    spawn('python3', 'pipes-testchild.py', 'spam')     # fork child program

    print('Hello 1 from parent', mypid)                # to child's stdin
    sys.stdout.flush()                                 # subvert stdio buffering
    reply = input()                                    # from child's stdout
    sys.stderr.write('Parent got: "%s"\n' % reply)     # stderr not tied to pipe!

    print('Hello 2 from parent', mypid)
    sys.stdout.flush()
    reply = sys.stdin.readline()
    sys.stderr.write('Parent got: "%s"\n' % reply[:-1])
```

PP4E/System/Processes/pipes-testchild.py

```
import os, time, sys

mypid     = os.getpid()
parentpid = os.getppid()
sys.stderr.write('Child %d of %d got arg: "%s"\n' %
                                (mypid, parentpid, sys.argv[1]))
for i in range(2):
    time.sleep(3)               # make parent process wait by sleeping here
    recv = input()              # stdin tied to pipe: comes from parent's stdout
    time.sleep(3)
    send = 'Child %d got: [%s]' % (mypid, recv)
    print(send)                 # stdout tied to pipe: goes to parent's stdin
    sys.stdout.flush()          # make sure it's sent now or else process blocks
```

PP4E/System/Processes/pipes.py

```
$ python3 pipes.py
Child 4481 of 4480 got arg: "spam"
Parent got: "Child 4481 got: [Hello 1 from parent 4480]"
Parent got: "Child 4481 got: [Hello 2 from parent 4480]"
$
```