

Subprocess to FFI in Python

Memory, Performance, and Why You Shouldn't Shell Out!



Christine Spang

MIT class of 2010



Ksplice®



Inbox is a startup

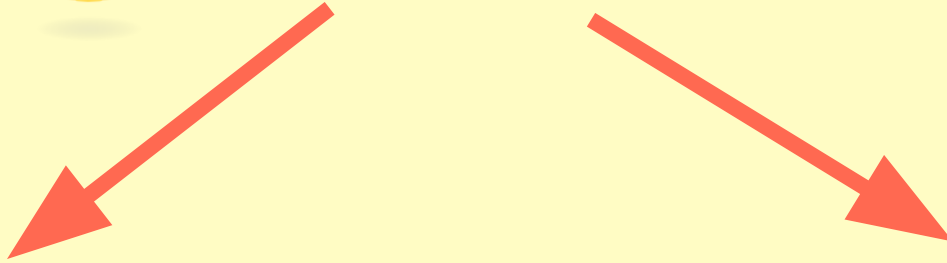
that I co-founded

in San Francisco

funded by top VCs

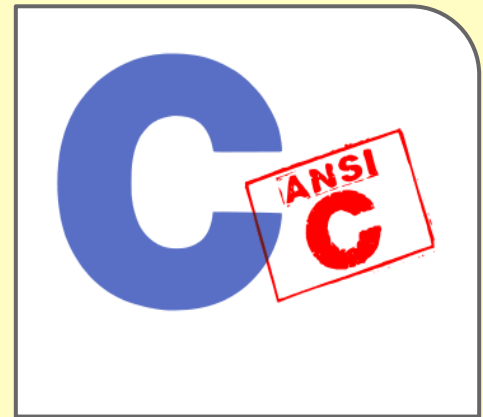
building a new email platform.

We ♥ Python



SHELL SCRIPT

VS.



C LIBRARY

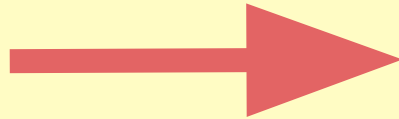
Python 2.7 on Linux

Why you shouldn't \$hell out!

And sometimes why you should...



7z



7zdecom.py

7z

SUBPROCESS

```
#!/usr/bin/env python

import subprocess
import argparse

parser = argparse.ArgumentParser(help="Extract a 7z archive")
parser.add_argument('filename')

args = parser.parse_args()

subprocess.check_call(['7z', 'x', args.filename])
```

(USUALLY PART OF LARGER SYSTEM.)



photo credit: <http://flic.kr/p/hBMUP5>

```
subprocess.check_call(  
    ['7z', 'x', args.filename])
```

```
check_call(*popenargs, **kwargs)
```

Run command with arguments. Wait for command to complete. If the exit code was zero then return, otherwise raise CalledProcessError. The CalledProcessError object will have the return code in the returncode attribute.

The arguments are the same as for the Popen constructor. Example:

```
check_call(["ls", "-l"])
```

Let's go source diving...

```
>>> from inspect import getsourcefile
>>> import subprocess
>>> getsourcefile(subprocess)
'/usr/lib/python2.7/subprocess.py'
```

```
errpipe_read, errpipe_write = self.pipe_cloexec()
try:
    try:
        gc_was_enabled = gc.isenabled()
        # Disable gc to avoid bug where gc -> file_dealloc ->
        # write to stderr -> hang. http://bugs.python.org/issue1336
```

```
self.pid = os.fork()
```

```
        gc.enable()
    raise
self._child_created = True
if self.pid == 0:
    # Child
```



photo credit: <http://flic.kr/p/hBMUP5>

a system call (syscall)

the API between a userspace application (like `7zdecom.py`) and the operating system's kernel

**parent
process**

```
7zdecom.py  
subprocess.check_call()
```

`fork()`

child process

`execvp()`

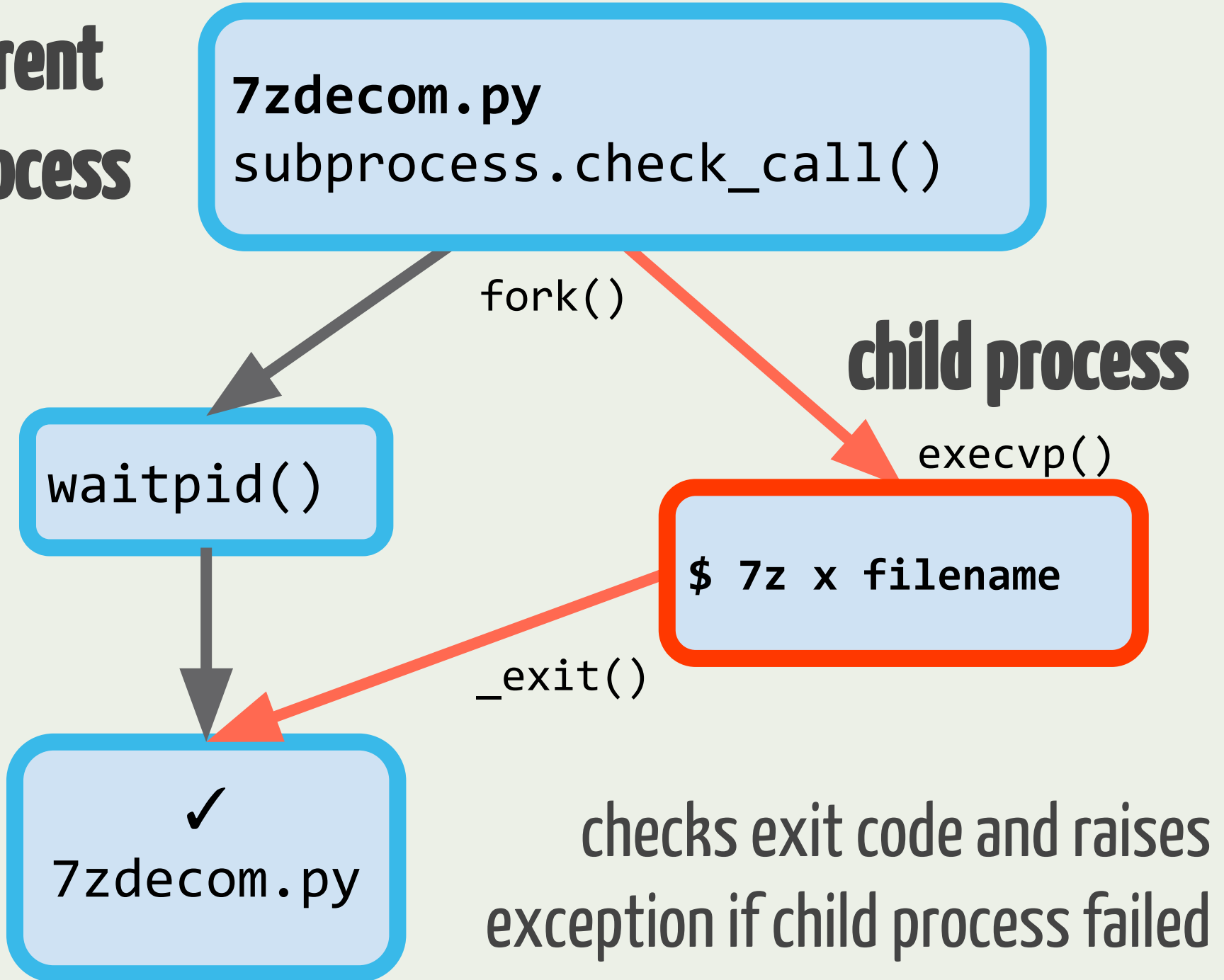
```
waitpid()
```

```
$ 7z x filename
```

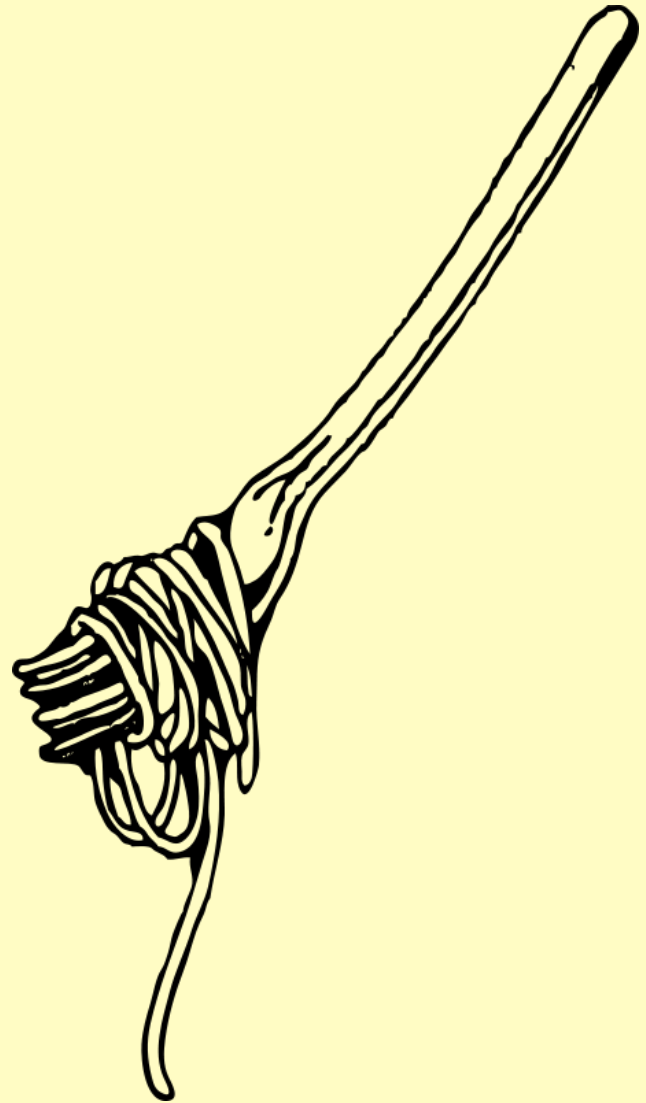
`_exit()`

✓
`7zdecom.py`

checks exit code and raises
exception if child process failed



fork()



NAME

fork - create a child process

SYNOPSIS

```
#include <unistd.h>
```

```
pid_t fork(void);
```

DESCRIPTION

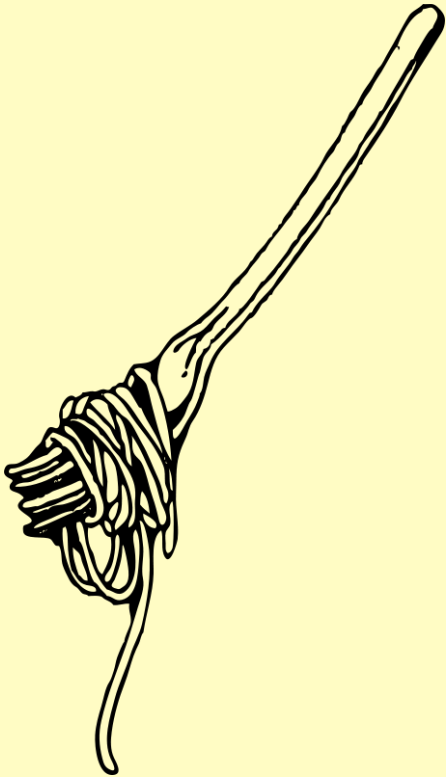
fork() creates a new process by duplicating the calling process. The new process, referred to as the child, is an exact duplicate of the calling process, referred to as the parent, except for the following points:

- * The child has its own unique process ID, and this PID does not match the ID of any existing process group (**setpgid(2)**).
- * The child's parent process ID is the same as the parent's process ID.
- * The child does not inherit its parent's memory locks (**mlock(2)**, **mlockall(2)**).
- * Process resource utilizations (**getrusage(2)**) and CPU time counters (**times(2)**) are reset to zero in

fork()

creates the child process by making a **copy** of the parent process.

The child process inherits the parent's memory pages: the program data is **shared** between the two processes, and the data, heap, and stack are given to the child **copy-on-write**.



fork() -> TWO PROCESSES

```
top - 20:21:29 up 25 min,  0 users,  load average: 0.04, 0.08, 0.16
Tasks:  11 total,   1 running,   8 sleeping,   2 stopped,   0 zombie
%Cpu(s):  0.0 us,  0.2 sy,  0.0 ni, 99.8 id,  0.0 wa,  0.0 hi,  0.0 si,
KiB Mem:  373512 total,  368488 used,   5024 free,   372 buffers
KiB Swap: 786428 total,  25076 used,  761352 free,  10516 cached
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
34	root	20	0	308m	275m	392	T	0.0	75.4	0:00.65	python
35	root	20	0	308m	274m	0	T	0.0	75.3	0:20.40	python

USING MORE TOTAL MEMORY THAN THE
ENTIRE SYSTEM HAS ALLOCATED

COPY - ON - WRITE



photo credit: <http://flic.kr/p/7eypMU>

OVERCOMMIT

```
→ ~ sudo sysctl -a | grep vm.overcommit  
vm.overcommit_memory = 0  
vm.overcommit_ratio = 50
```

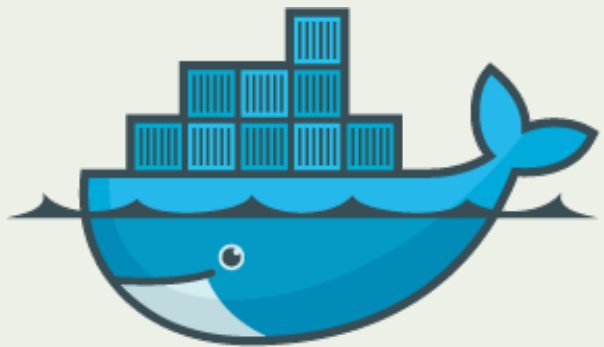
When `overcommit_memory` flag is 0, the kernel attempts to estimate the amount of free memory left when userspace requests more memory.

— docs from [Kernel.org](https://kernel.org)

OVERCOMMIT

SOMETIMES MORE COMPLICATED

OOM Kill



docker

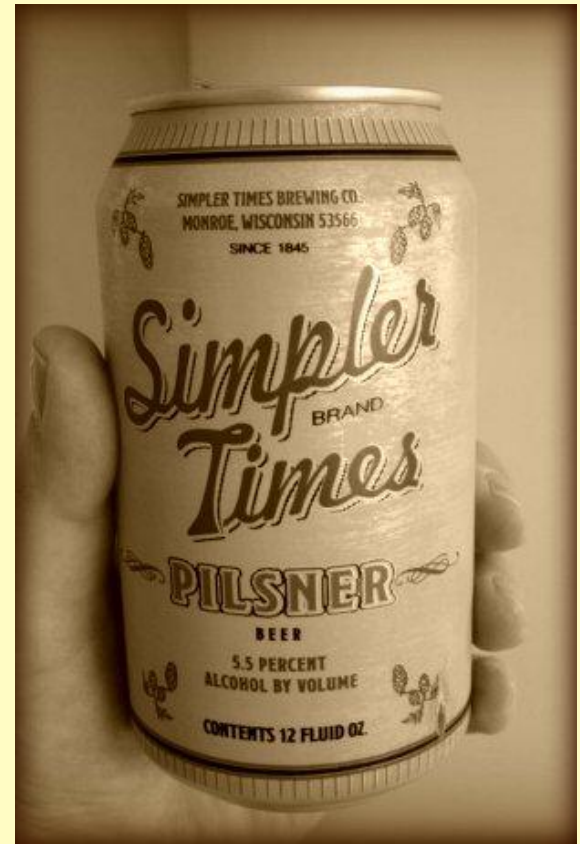
why shell out?

using subprocess module

simple and easy

flexible enough

throws native Python exceptions



the dangers...

of forking your process

significant overhead

limited API

parsing stdout/stderr

DO IT ANYWAY

(USUALLY)

Another option:

Wrapping C libraries from Python

The usual suspects

The usual suspects

C extension

write lots of C with Python's C API

The usual suspects

C extension

write lots of C with Python's C API

ctypes

standard library, no C compiler needed,
but tedious and clunky

The usual suspects

C extension

write lots of C with Python's C API

ctypes

standard library, no C compiler needed,
but tedious and clunky

Cython

Python/C hybrid language

The usual suspects

C extension

write lots of C with Python's C API

ctypes

standard library, no C compiler needed, but tedious and clunky

Cython

Python/C hybrid language

CFFI

written by PyPy authors, clean, ABI or API (needs compiler) interface

What did we learn?

\$HELLING OUT IS

EXPENSIVE

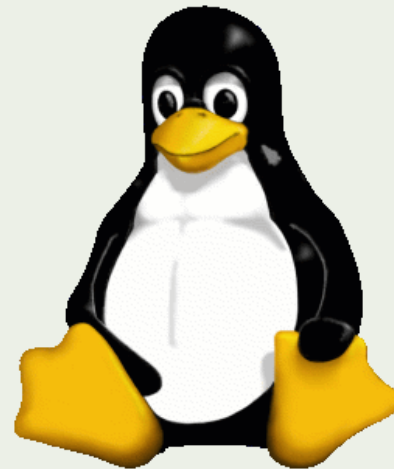
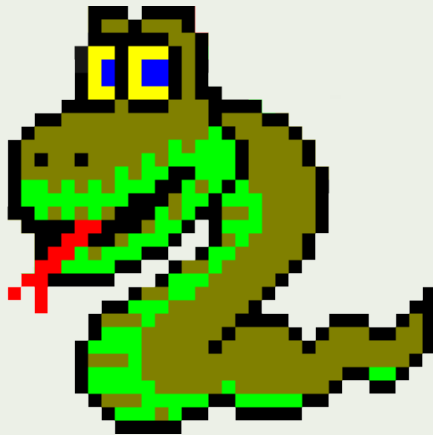
IN BOTH MEMORY AND COMPUTATION

**TO MAKE IT FASTER &
HAVE MORE CONTROL...**

WRAP YOUR LIBRARIES

WITH CFFI

EVEN WHEN USING A HIGH-LEVEL LANGUAGE...



KNOW YOUR OS

Say hi!

spang@mit.edu

follow @spang

all examples on GitHub

Like this? Come work at Inbox! :)