Subprocess to FFI

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

Christine Spang

Inbox is a startup

that I co-founded

in San Francisco

funded by top VCs

building a new email platform.

We ❤ Python

python™

EXTERNAL BINARY VS. C LIBRARY

CPython 2.7 on Linux

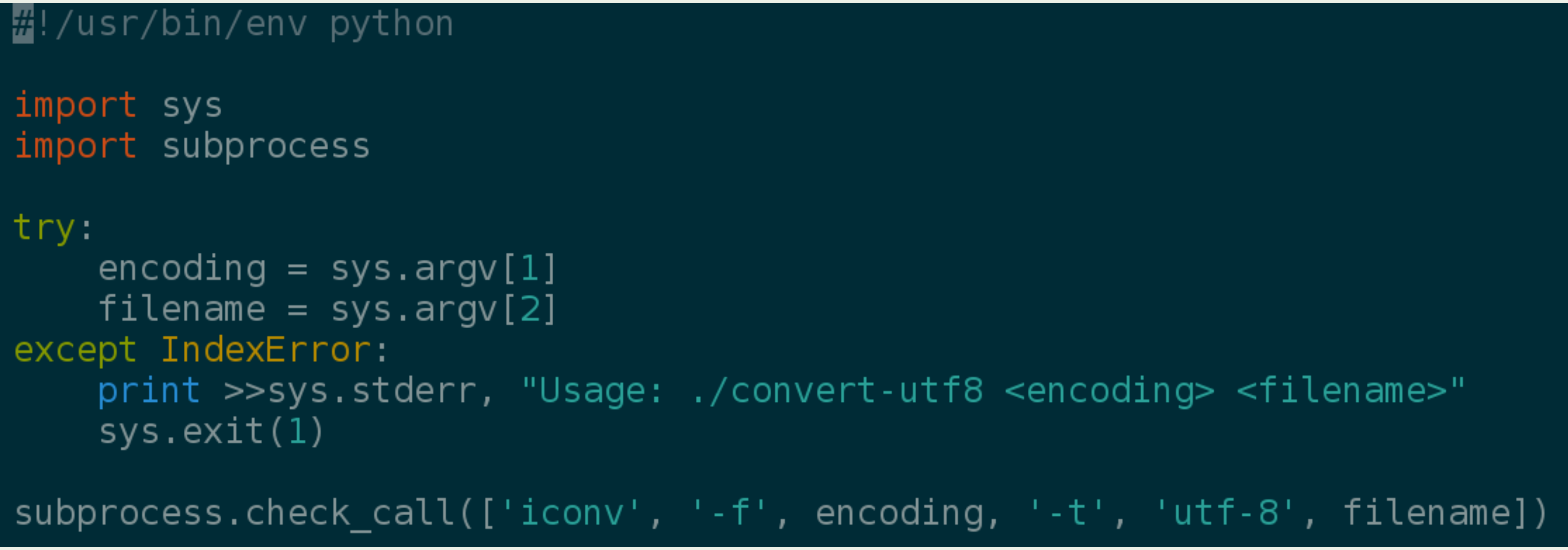
Why you shouldn’t $hell out!

And sometimes why you should...

iconv

$$\begin{tikzcd}\texttt{convert-utf8} \arrow[r] & \texttt{iconv}\end{tikzcd}$$

SUBPROCESS

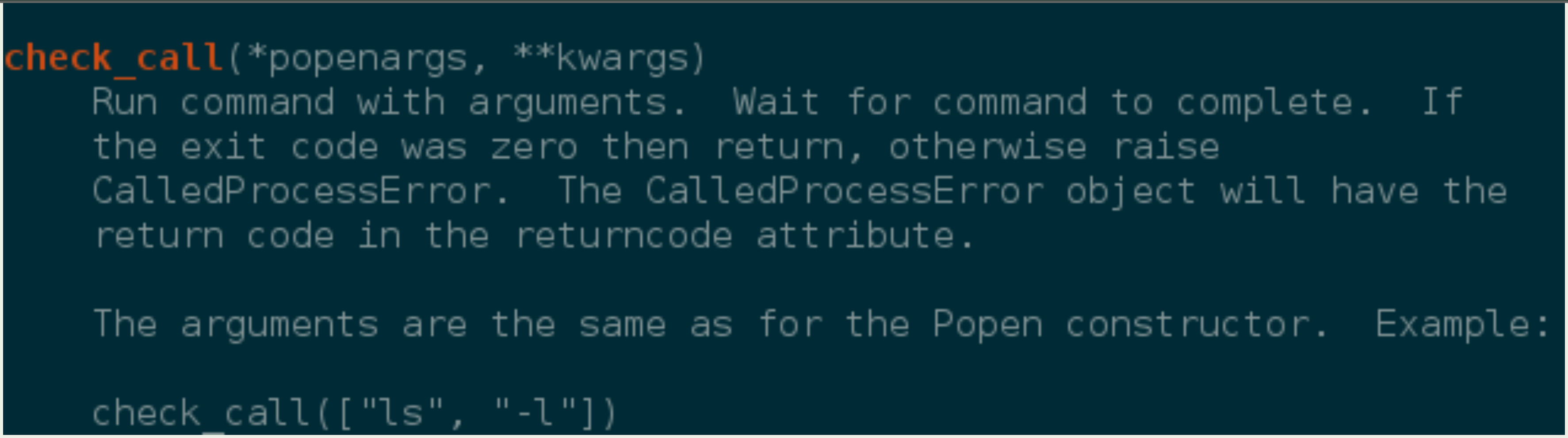


```python
#!/usr/bin/env python

import sys
import subprocess

try:
    encoding = sys.argv[1]
    filename = sys.argv[2]
except IndexError:
    print >>sys.stderr, "Usage: ./convert-utf8 <encoding> <filename>"
    sys.exit(1)

subprocess.check_call(['iconv', '-f', encoding, '-t', 'utf-8', filename])
```

(USUALLY PART OF LARGER SYSTEM.)

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

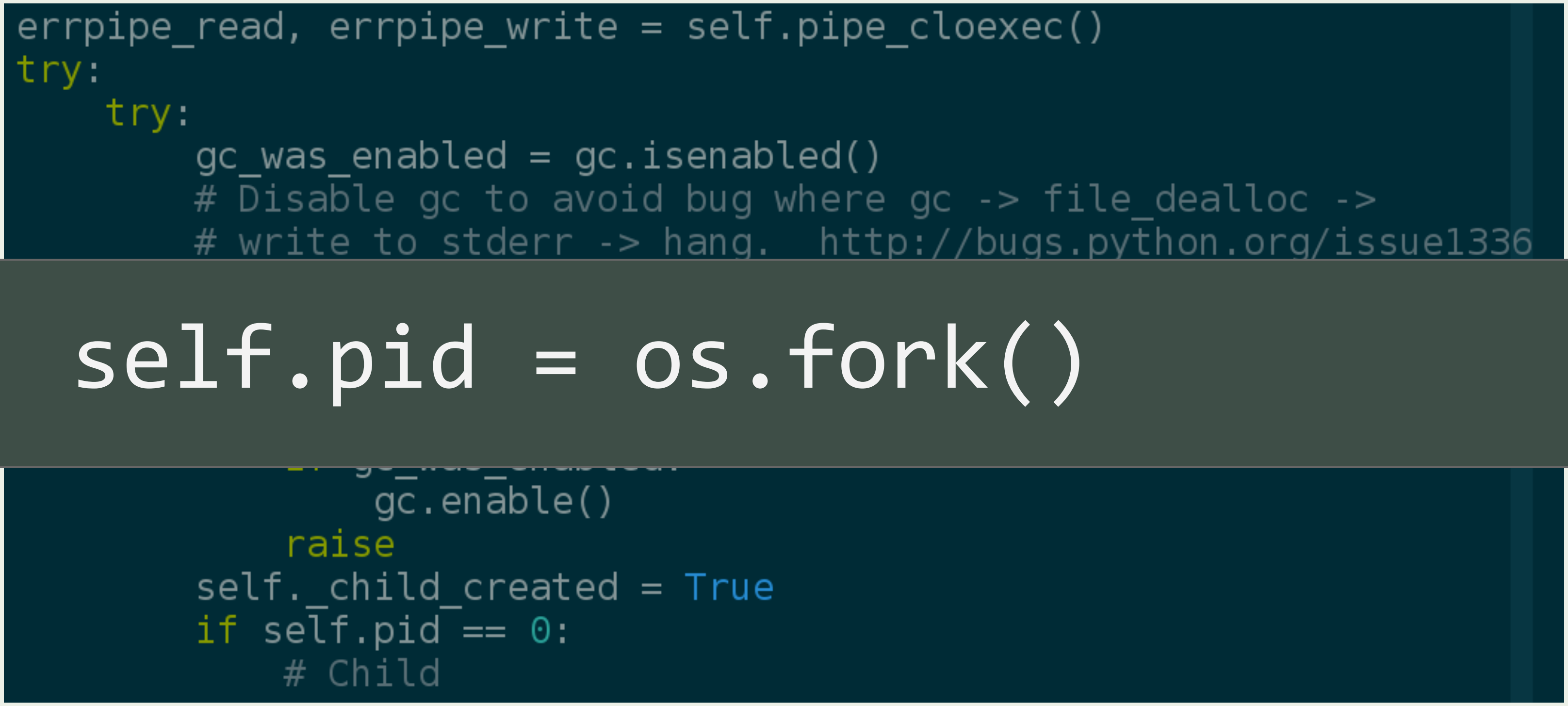
```
subprocess.check_call(
['iconv', '-f', encoding, '-t',
'utf-8', filename])
```



```
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()
```

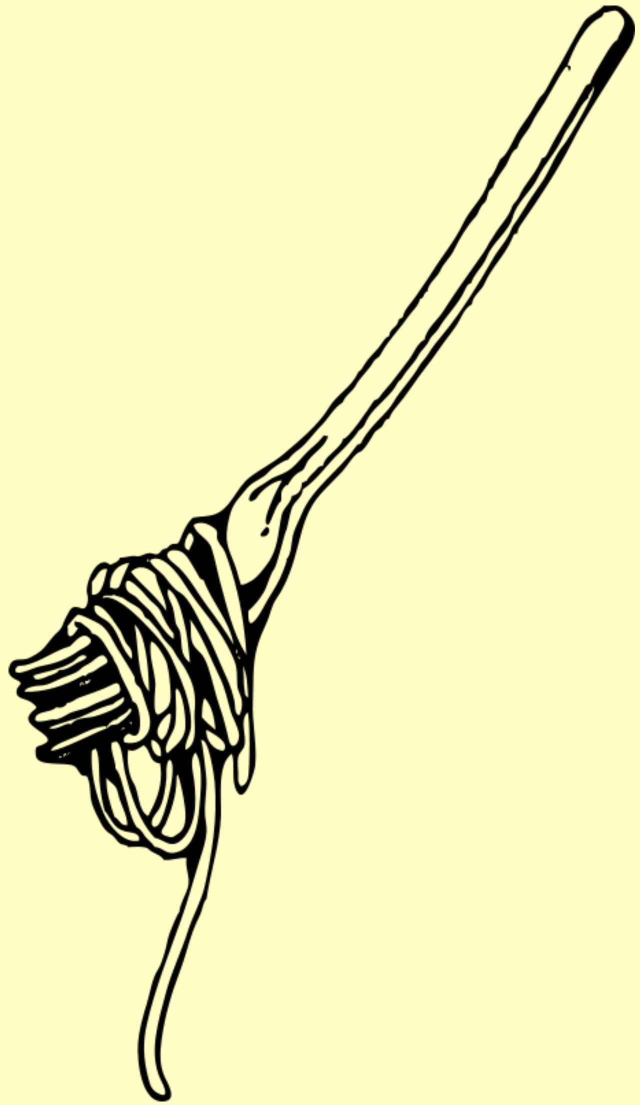
```
            if gc_was_enabled:
                gc.enable()
            raise
        self._child_created = True
        if self.pid == 0:
            # Child
```

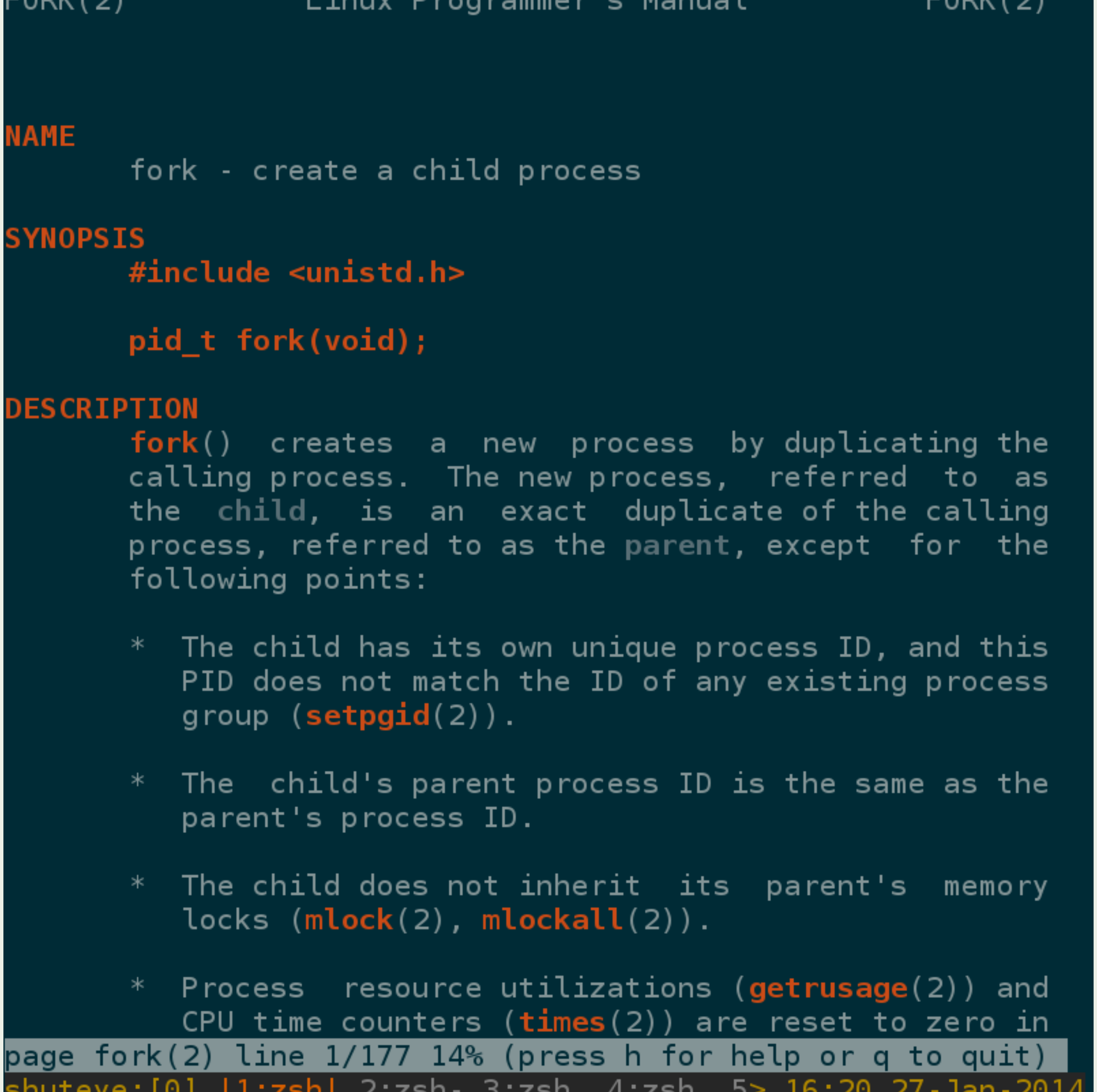
photo credit: http://flic.kr/p/hBMUP5

a system call (syscall)

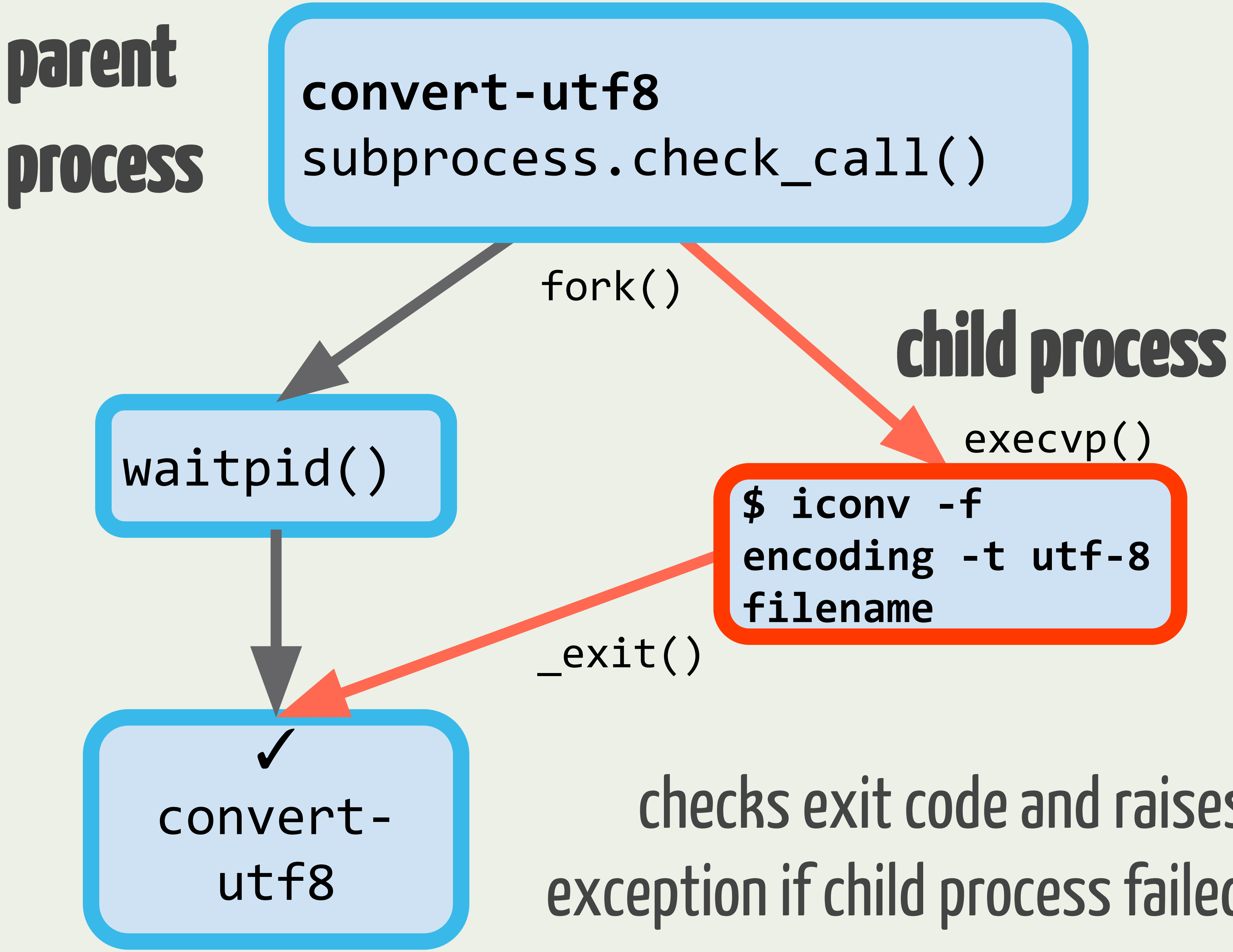
the API between a userspace application (like `convert-utf8`) and the operating system's kernel

fork()





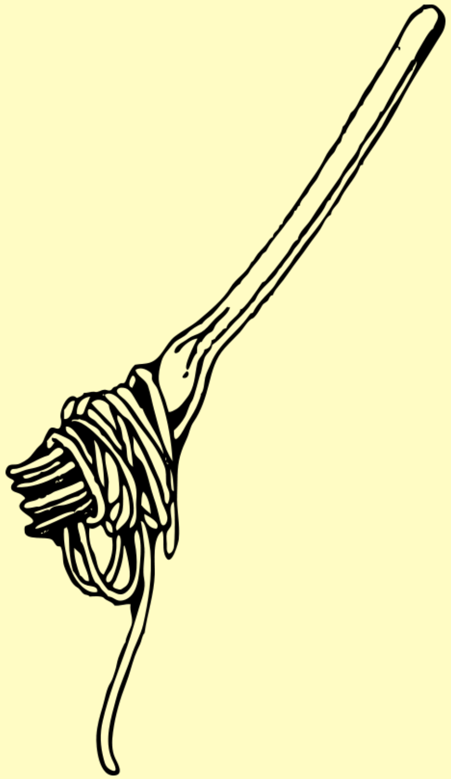
```
FORK(2)              Linux Programmer's Manual              FORK(2)



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
page fork(2) line 1/177 14% (press h for help or q to quit)
shuteye:[0] [1:zsh] 2:zsh- 3:zsh  4:zsh  5> 16:20 27-Jan-2014
```



```
$ iconv -f encoding -t utf-8 filename
```

checks exit code and raises exception if child process failed

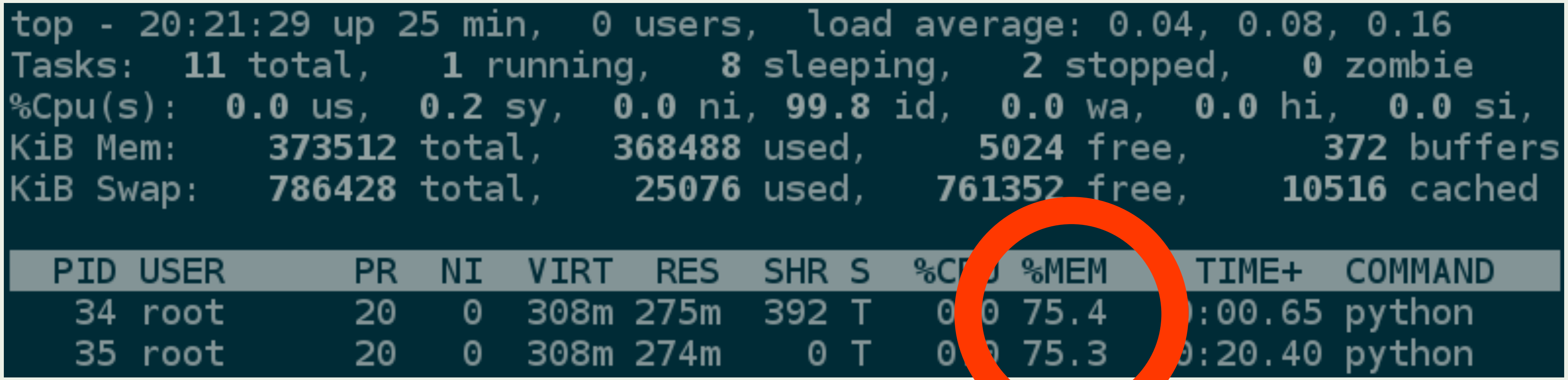
fork()



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



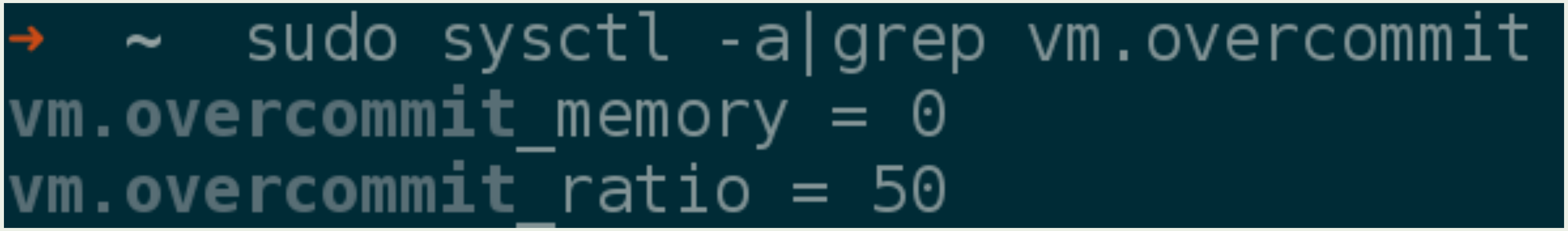
```
top - 20:21:29 up 25 min,  0 users,  load average: 0.04, 0.08, 0.16
Tasks:  11 total,   1 running,   8 sleeping,   2 stopped,   0 zombie
%Cpu(s):  0.0 us,  0.2 sy,  0.0 ni, 99.8 id,  0.0 wa,  0.0 hi,  0.0 si,
KiB Mem:    373512 total,   368488 used,     5024 free,      372 buffers
KiB Swap:   786428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

OVERCOMMIT

SOMETIMES MORE COMPLICATED

why shell out?

using subprocess module

simple and easy

flexible enough

throws native Python exceptions

photo credit: http://flic.kr/p/6UTVj7

the dangers...

of forking your process

significant overhead (fork, file I/O vs memory)

limited API

parsing stdout/stderr

flushing, buffering, deadlocks issues w/pipes

DO IT ANYWAY

(USUALLY)

Another option:

Wrapping C libraries from Python

FFI: foreign function interface

a way to call ***functions*** and use ***data structures*** provided by one language in another language.

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 (wraps libffi), no C compiler needed, but tedious and clunky

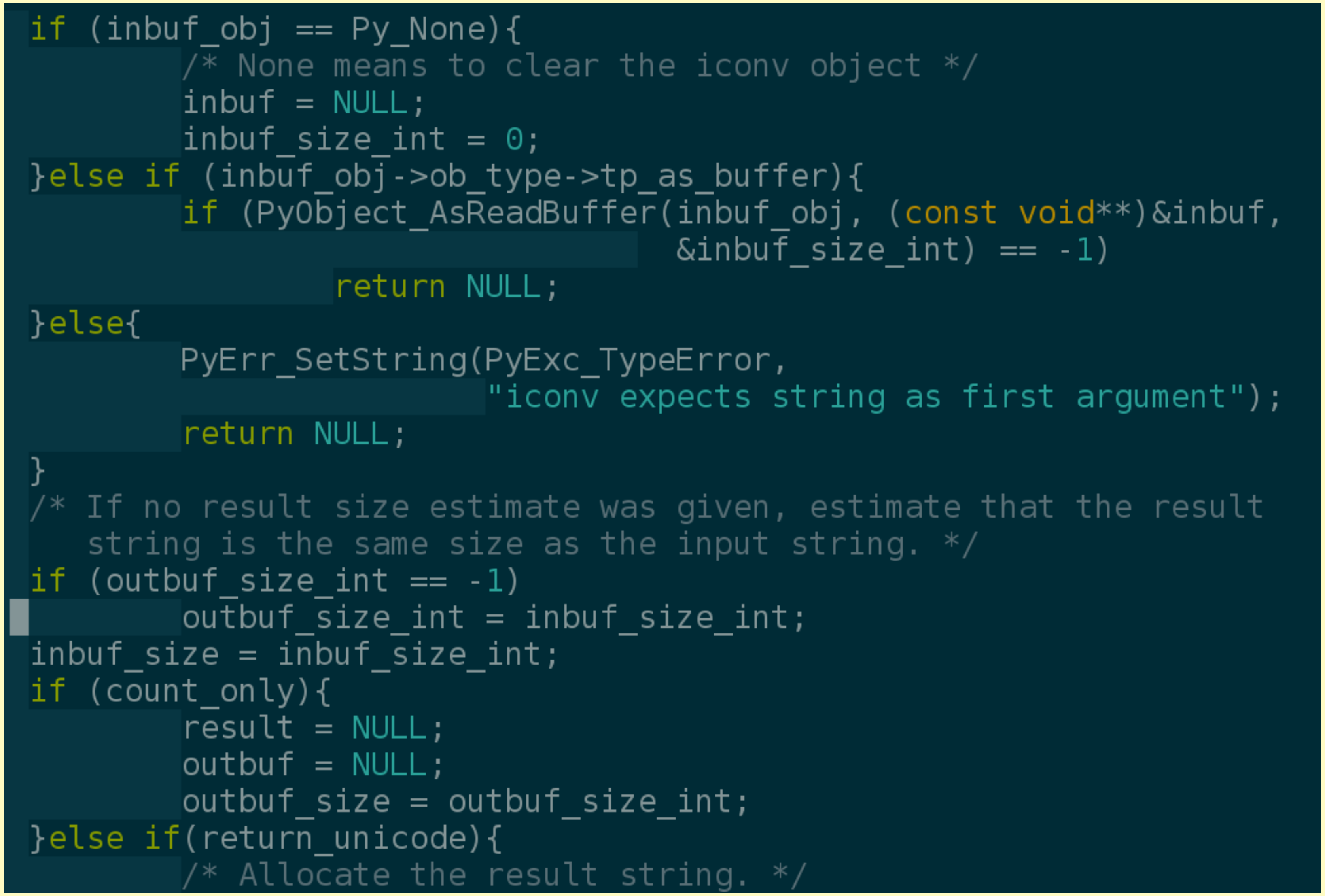
The usual suspects

C extension	write lots of C with Python's C API
ctypes	standard library (wraps libffi), no C compiler needed, but tedious and clunky
Cython	Python/C hybrid language, more for optimizing speed than wrapping

The usual suspects

C extension	write lots of C with Python's C API
ctypes	standard library (wraps libffi), no C compiler needed, but tedious and clunky
Cython	Python/C hybrid language, more for optimizing speed than wrapping
CFFI	written to address ctypes shortcomings, ABI or API (needs compiler) interface

Wrapping libiconv: C extension



```
if (inbuf_obj == Py_None){
        /* None means to clear the iconv object */
        inbuf = NULL;
        inbuf_size_int = 0;
}else if (inbuf_obj->ob_type->tp_as_buffer){
        if (PyObject_AsReadBuffer(inbuf_obj, (const void**)&inbuf,
                                  &inbuf_size_int) == -1)
                return NULL;
}else{
        PyErr_SetString(PyExc_TypeError,
                "iconv expects string as first argument");
        return NULL;
}
/* If no result size estimate was given, estimate that the result
   string is the same size as the input string. */
if (outbuf_size_int == -1)
        outbuf_size_int = inbuf_size_int;
inbuf_size = inbuf_size_int;
if (count_only){
        result = NULL;
        outbuf = NULL;
        outbuf_size = outbuf_size_int;
}else if(return_unicode){
        /* Allocate the result string. */
```

Wrapping libiconv: CFFI

```
def iconv(self, cd, msg_bytes, errors='strict'):
    # can't do &inbuf in cffi, need to explicitly create, fill pointer
    inbuf = ffi.new("char **")
    inbuf_text = ffi.new("char[]", msg_bytes)
    # *inbuf in cffi (works in C too but atypical)
    inbuf[0] = inbuf_text
    # give the output buffer some extra bytes compared to the input buffer
    # in case the input charset is more efficient for this string than
    # utf-8
    outbuf_size = len(msg_bytes) * 2
    outbuf = ffi.new("char **")
    outbuf_text = ffi.new("char []", outbuf_size)
    outbuf[0] = outbuf_text
    inbytesleft = ffi.new("size_t *")
    inbytesleft[0] = ffi.sizeof(inbuf_text)
    outbytesleft = ffi.new("size_t *")
    outbytesleft[0] = outbuf_size

    nconv = ffi.cast('int',
            C.iconv(cd, inbuf, inbytesleft, outbuf, outbytesleft))

    self._check_errors(int(nconv))

    data_size = outbuf_size - outbytesleft[0]
```

Write less C.

(you are not a superhuman)

Python C extension: 252 lines of C

CFFI wrapper: 120 lines of Python/C
(~40 lines actually interface with C)

What did we learn?

$HELLING OUT IS

EXPENSIVE

IN BOTH MEMORY AND COMPUTATION

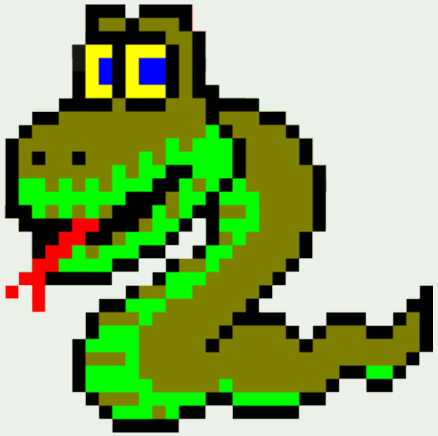
TO MAKE IT FASTER &

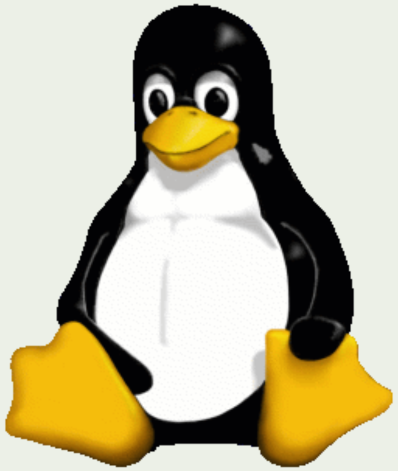
HAVE MORE CONTROL...

WRAP YOUR LIBRARIES

WITH CFFI (usually)

EVEN WHEN USING A HIGH-LEVEL LANGUAGE...





KNOW YOUR OS

Say hi !

spang@inboxapp.com

follow @spang

all examples on GitHub

Like this? Come work at Inbox! :)