11-Native-Code-Binding

November 21, 2019

In [10]:
```
from multiprocessing import Process, Lock, RLock, Queue, Value, Array
import random
import time
from socket import *

def echoservice(sock,i):
    ''' echo uppercase string back in a loop'''
    try:
        while True:
            ns, peer = sock.accept()
            req = ns.recv(1000)
            while req and req != '':
                # remove trailing newline and blanks
                req = req.rstrip()
                print(i,'serving',req.decode())
                ns.send(req.decode().upper().encode())
                req = ns.recv(1000)
            print(peer, ' closing')
    finally:
        pass


def client(n, port):
    # send n random request
    # the connection is kept alive until client closes it.
    mess = ['hello', 'bye', 'why', 'yes', 'no', 'maybe', 'are you sure', 'why not?']
    c = socket(AF_INET, SOCK_STREAM)
    c.connect(('127.0.0.1', port))
    for i in range(n):
        time.sleep(random.random()*3)
        c.send(random.choice(mess).encode())
        reply = c.recv(1024)
        print(c.getsockname(), reply)
    c.close()


s = socket(AF_INET, SOCK_STREAM)
```

```
        s.bind(('',20446))
        s.listen(1)    # 1 is queue size for "not yet accept()'ed connections"

        # create 3 workers
        workers = [Process(target = echoservice, args=(s,i)) for i in range(3)]
        for w in workers: w.start()

        # create 10 clients
        clients = [Process(target = client, args=(5, 20446)) for i in range(30)]
        # start clients
        for cl in clients: cl.start()

        # just for python book not to keep port open. Kill all processes and close the socket
        time.sleep(20)
        for w in workers: w.terminate()
        s.close()
```

```
2 serving are you sure
('127.0.0.1', 44706) b'ARE YOU SURE'
0 serving hello
1 serving why
('127.0.0.1', 44702) b'HELLO'
('127.0.0.1', 44704) b'WHY'
2 serving no
('127.0.0.1', 44706) b'NO'
1 serving hello
('127.0.0.1', 44704) b'HELLO'
1 serving why
('127.0.0.1', 44704) b'WHY'
0 serving maybe
('127.0.0.1', 44702) b'MAYBE'
1 serving yes
('127.0.0.1', 44704) b'YES'
2 serving no
('127.0.0.1', 44706) b'NO'
0 serving no
('127.0.0.1', 44702) b'NO'
2 serving yes
('127.0.0.1', 44706) b'YES'
1 serving bye
('127.0.0.1', 44704) b'BYE'
('127.0.0.1', 44704)  closing
1 serving bye
('127.0.0.1', 44708) b'BYE'
2 serving are you sure
('127.0.0.1', 44706) b'ARE YOU SURE'
('127.0.0.1', 44706)  closing
2 serving are you sure
```

```
('127.0.0.1', 44710) b'ARE YOU SURE'
2 serving why not?
('127.0.0.1', 44710) b'WHY NOT?'
0 serving no
('127.0.0.1', 44702) b'NO'
1 serving are you sure
('127.0.0.1', 44708) b'ARE YOU SURE'
1 serving are you sure
('127.0.0.1', 44708) b'ARE YOU SURE'
2 serving yes
('127.0.0.1', 44710) b'YES'
0 serving why
('127.0.0.1', 44702) b'WHY'
('127.0.0.1', 44702)  closing
0 serving why not?
('127.0.0.1', 44722) b'WHY NOT?'
2 serving yes
('127.0.0.1', 44710) b'YES'
1 serving are you sure
('127.0.0.1', 44708) b'ARE YOU SURE'
2 serving yes
('127.0.0.1', 44710) b'YES'
('127.0.0.1', 44710)  closing
2 serving hello
('127.0.0.1', 44734) b'HELLO'
1 serving hello
('127.0.0.1', 44708) b'HELLO'
('127.0.0.1', 44708)  closing
2 serving yes
('127.0.0.1', 44734) b'YES'
0 serving are you sure
('127.0.0.1', 44722) b'ARE YOU SURE'
1 serving are you sure
('127.0.0.1', 44736) b'ARE YOU SURE'
0 serving maybe
('127.0.0.1', 44722) b'MAYBE'
0 serving bye
('127.0.0.1', 44722) b'BYE'
1 serving are you sure
('127.0.0.1', 44736) b'ARE YOU SURE'
2 serving why not?
('127.0.0.1', 44734) b'WHY NOT?'
0 serving yes
('127.0.0.1', 44722) b'YES'
('127.0.0.1', 44722)  closing
0 serving yes
('127.0.0.1', 44758) b'YES'
2 serving bye
```

```
('127.0.0.1', 44734) b'BYE'
1 serving why
('127.0.0.1', 44736) b'WHY'
1 serving maybe
('127.0.0.1', 44736) b'MAYBE'
2 serving why
('127.0.0.1', 44734) b'WHY'
('127.0.0.1', 44734)  closing
2 serving bye
('127.0.0.1', 44724) b'BYE'
1 serving are you sure
('127.0.0.1', 44736) b'ARE YOU SURE'
('127.0.0.1', 44736)  closing
1 serving no
('127.0.0.1', 44752) b'NO'
0 serving hello
('127.0.0.1', 44758) b'HELLO'
2 serving why not?
('127.0.0.1', 44724) b'WHY NOT?'
1 serving hello
('127.0.0.1', 44752) b'HELLO'
1 serving hello
('127.0.0.1', 44752) b'HELLO'
0 serving hello
2 serving hello
('127.0.0.1', 44758) b'HELLO'
('127.0.0.1', 44724) b'HELLO'
1 serving why not?
('127.0.0.1', 44752) b'WHY NOT?'
('127.0.0.1', 44752) b''
('127.0.0.1', 44724) b''
('127.0.0.1', 44758) b''


Process Process-92:
Traceback (most recent call last):
  File "/usr/lib/python3.5/multiprocessing/process.py", line 249, in _bootstrap
    self.run()
  File "/usr/lib/python3.5/multiprocessing/process.py", line 93, in run
    self._target(*self._args, **self._kwargs)
  File "<ipython-input-10-0662c8189334>", line 31, in client
    c.send(random.choice(mess).encode())
BrokenPipeError: [Errno 32] Broken pipe
Process Process-74:
Traceback (most recent call last):
  File "/usr/lib/python3.5/multiprocessing/process.py", line 249, in _bootstrap
    self.run()
  File "/usr/lib/python3.5/multiprocessing/process.py", line 93, in run
```

```
    self._target(*self._args, **self._kwargs)
  File "<ipython-input-10-0662c8189334>", line 31, in client
    c.send(random.choice(mess).encode())
BrokenPipeError: [Errno 32] Broken pipe
Process Process-94:
Process Process-90:
Process Process-84:
Process Process-86:
Process Process-72:
Process Process-85:
Process Process-75:
Process Process-71:
Process Process-87:
Traceback (most recent call last):
Process Process-76:
Traceback (most recent call last):
Traceback (most recent call last):
Traceback (most recent call last):
Traceback (most recent call last):
  File "/usr/lib/python3.5/multiprocessing/process.py", line 249, in _bootstrap
    self.run()
Traceback (most recent call last):
Traceback (most recent call last):
Process Process-93:
Traceback (most recent call last):
Traceback (most recent call last):
  File "/usr/lib/python3.5/multiprocessing/process.py", line 249, in _bootstrap
    self.run()
  File "/usr/lib/python3.5/multiprocessing/process.py", line 249, in _bootstrap
    self.run()
  File "/usr/lib/python3.5/multiprocessing/process.py", line 93, in run
    self._target(*self._args, **self._kwargs)
  File "/usr/lib/python3.5/multiprocessing/process.py", line 249, in _bootstrap
    self.run()
  File "/usr/lib/python3.5/multiprocessing/process.py", line 249, in _bootstrap
    self.run()
Traceback (most recent call last):
  File "/usr/lib/python3.5/multiprocessing/process.py", line 249, in _bootstrap
    self.run()
Traceback (most recent call last):
  File "/usr/lib/python3.5/multiprocessing/process.py", line 249, in _bootstrap
    self.run()
  File "/usr/lib/python3.5/multiprocessing/process.py", line 93, in run
    self._target(*self._args, **self._kwargs)
  File "/usr/lib/python3.5/multiprocessing/process.py", line 249, in _bootstrap
    self.run()
  File "/usr/lib/python3.5/multiprocessing/process.py", line 249, in _bootstrap
    self.run()
```

```
  File "/usr/lib/python3.5/multiprocessing/process.py", line 93, in run
    self._target(*self._args, **self._kwargs)
  File "/usr/lib/python3.5/multiprocessing/process.py", line 249, in _bootstrap
    self.run()
  File "/usr/lib/python3.5/multiprocessing/process.py", line 93, in run
    self._target(*self._args, **self._kwargs)
  File "/usr/lib/python3.5/multiprocessing/process.py", line 93, in run
    self._target(*self._args, **self._kwargs)
  File "<ipython-input-10-0662c8189334>", line 32, in client
    reply = c.recv(1024)
  File "/usr/lib/python3.5/multiprocessing/process.py", line 93, in run
    self._target(*self._args, **self._kwargs)
  File "<ipython-input-10-0662c8189334>", line 32, in client
    reply = c.recv(1024)
  File "/usr/lib/python3.5/multiprocessing/process.py", line 93, in run
    self._target(*self._args, **self._kwargs)
  File "/usr/lib/python3.5/multiprocessing/process.py", line 249, in _bootstrap
    self.run()
  File "/usr/lib/python3.5/multiprocessing/process.py", line 93, in run
    self._target(*self._args, **self._kwargs)
  File "<ipython-input-10-0662c8189334>", line 32, in client
    reply = c.recv(1024)
  File "/usr/lib/python3.5/multiprocessing/process.py", line 93, in run
    self._target(*self._args, **self._kwargs)
  File "<ipython-input-10-0662c8189334>", line 32, in client
    reply = c.recv(1024)
  File "/usr/lib/python3.5/multiprocessing/process.py", line 93, in run
    self._target(*self._args, **self._kwargs)
ConnectionResetError: [Errno 104] Connection reset by peer
  File "<ipython-input-10-0662c8189334>", line 32, in client
    reply = c.recv(1024)
  File "/usr/lib/python3.5/multiprocessing/process.py", line 93, in run
    self._target(*self._args, **self._kwargs)
ConnectionResetError: [Errno 104] Connection reset by peer
  File "<ipython-input-10-0662c8189334>", line 32, in client
    reply = c.recv(1024)
ConnectionResetError: [Errno 104] Connection reset by peer
  File "<ipython-input-10-0662c8189334>", line 32, in client
    reply = c.recv(1024)
  File "<ipython-input-10-0662c8189334>", line 32, in client
    reply = c.recv(1024)
ConnectionResetError: [Errno 104] Connection reset by peer
ConnectionResetError: [Errno 104] Connection reset by peer
ConnectionResetError: [Errno 104] Connection reset by peer
ConnectionResetError: [Errno 104] Connection reset by peer
  File "<ipython-input-10-0662c8189334>", line 32, in client
    reply = c.recv(1024)
  File "<ipython-input-10-0662c8189334>", line 32, in client
```

```
    reply = c.recv(1024)
ConnectionResetError: [Errno 104] Connection reset by peer
ConnectionResetError: [Errno 104] Connection reset by peer
  File "<ipython-input-10-0662c8189334>", line 32, in client
    reply = c.recv(1024)
ConnectionResetError: [Errno 104] Connection reset by peer
ConnectionResetError: [Errno 104] Connection reset by peer
Process Process-80:
Process Process-78:
Process Process-91:
Process Process-79:
Traceback (most recent call last):
Process Process-73:
Traceback (most recent call last):
Process Process-81:
Traceback (most recent call last):
Traceback (most recent call last):
  File "/usr/lib/python3.5/multiprocessing/process.py", line 249, in _bootstrap
    self.run()
Traceback (most recent call last):
  File "/usr/lib/python3.5/multiprocessing/process.py", line 249, in _bootstrap
    self.run()
  File "/usr/lib/python3.5/multiprocessing/process.py", line 249, in _bootstrap
    self.run()
  File "/usr/lib/python3.5/multiprocessing/process.py", line 249, in _bootstrap
    self.run()
  File "/usr/lib/python3.5/multiprocessing/process.py", line 93, in run
    self._target(*self._args, **self._kwargs)
  File "/usr/lib/python3.5/multiprocessing/process.py", line 93, in run
    self._target(*self._args, **self._kwargs)
  File "/usr/lib/python3.5/multiprocessing/process.py", line 93, in run
    self._target(*self._args, **self._kwargs)
  File "/usr/lib/python3.5/multiprocessing/process.py", line 249, in _bootstrap
    self.run()
Traceback (most recent call last):
  File "<ipython-input-10-0662c8189334>", line 32, in client
    reply = c.recv(1024)
  File "<ipython-input-10-0662c8189334>", line 32, in client
    reply = c.recv(1024)
  File "/usr/lib/python3.5/multiprocessing/process.py", line 249, in _bootstrap
    self.run()
ConnectionResetError: [Errno 104] Connection reset by peer
  File "/usr/lib/python3.5/multiprocessing/process.py", line 93, in run
    self._target(*self._args, **self._kwargs)
  File "/usr/lib/python3.5/multiprocessing/process.py", line 93, in run
    self._target(*self._args, **self._kwargs)
  File "<ipython-input-10-0662c8189334>", line 32, in client
    reply = c.recv(1024)
```

```
  File "<ipython-input-10-0662c8189334>", line 32, in client
    reply = c.recv(1024)
ConnectionResetError: [Errno 104] Connection reset by peer
ConnectionResetError: [Errno 104] Connection reset by peer
  File "<ipython-input-10-0662c8189334>", line 32, in client
    reply = c.recv(1024)
ConnectionResetError: [Errno 104] Connection reset by peer
  File "/usr/lib/python3.5/multiprocessing/process.py", line 93, in run
    self._target(*self._args, **self._kwargs)
ConnectionResetError: [Errno 104] Connection reset by peer
  File "<ipython-input-10-0662c8189334>", line 32, in client
    reply = c.recv(1024)
ConnectionResetError: [Errno 104] Connection reset by peer
```

```
In [13]: ''' Thread pool example. Using slightly complex mechanism to transfer socket
             to threads in the pool (through a shared variable and condition variables).
             This way, load balancing and better pool control is possible. (not implemented here
              but possible)'''
         from socket import *
         import time
         import random
         from threading import Thread, Condition, Lock

         terminate = False

         def echoservice(sockets, myid, mycond, done):
             ''' echo uppercase string back in a loop'''
             while not terminate:
                 with mycond:
                     while not terminate and  type(sockets[myid]) != socket:
                         mycond.wait()
                     if terminate:
                         break
                 print(myid, "serving", sockets[myid].getpeername())
                 req = sockets[myid].recv(1000)
                 while req and req != '':
                     # remove trailing newline and blanks
                     req = req.rstrip()
                     sockets[myid].send(req.decode().upper().encode())
                     req = sockets[myid].recv(1000)
                 print(sockets[myid].getpeername(), ' closing')
                 sockets[myid].close()
                 with mycond:
                     sockets[myid] = 'FREE'
                     done.notify()
             if type(sockets[myid]) == socket:
                 sockets[myid].close()
```

```
def client(n, port):
    # send n random request
    # the connection is kept alive until client closes it.
    mess = ['hello', 'bye', 'why', 'yes', 'no', 'maybe', 'are you sure', 'why not?']
    c = socket(AF_INET, SOCK_STREAM)
    c.connect(('127.0.0.1', port))
    for i in range(n):
        time.sleep(random.random()*3)
        c.send(random.choice(mess).encode())
        reply = c.recv(1024)
        print(c.getsockname(), reply)
    c.close()

def checkfree(s):
    try:
        r = s.index('FREE')
        return r
    except:
        return None


N = 3

sockets = ['FREE'] * N
slock = Lock()
conditions = [ Condition(slock) for i in range(N)]
clientready = Condition(slock)

# create 3 workers
workers = [Thread(target = echoservice, args=(sockets, i, conditions[i], clientready))
for w in workers: w.start()

s = socket(AF_INET, SOCK_STREAM)
s.bind(('',20447))
s.listen(5)    # 1 is queue size for "not yet accept()'ed connections"

# create 5 clients
clients = [Process(target = client, args=(5, 20447)) for i in range(5)]
# start clients
for cl in clients: cl.start()

s.settimeout(20)     # special to python notebook. if no incoming connection for 20 sec
# otherwise python notebook kernel stays alive occupying the port
try:
    while True:
```

```
                ns, peer = s.accept()
                with slock:
                    r = checkfree(sockets)
                    while r == None:
                        clientready.wait()
                        r = checkfree(sockets)
                    sockets[r] = ns
                    conditions[r].notify()
        except Exception as e:
            print("exitting", str(e), ". Will wait for ongoing connections to close")

        terminate = True

        for i in range(N):    # let all workers terminate
            with slock:
                conditions[i].notify()


        for w in workers: w.join()

        s.close()
        # just for python book not to keep port open. Kill all processes and close the socket
        for cl in clients:
            cl.terminate()
            cl.join()
        print("complete")
```

```
0 serving ('127.0.0.1', 41124)
1 serving ('127.0.0.1', 41126)
2 serving ('127.0.0.1', 41128)
('127.0.0.1', 41128) b'WHY'
('127.0.0.1', 41124) b'MAYBE'
('127.0.0.1', 41128) b'HELLO'
('127.0.0.1', 41126) b'BYE'
('127.0.0.1', 41124) b'MAYBE'
('127.0.0.1', 41124) b'NO'
('127.0.0.1', 41128) b'BYE'
('127.0.0.1', 41126) b'NO'
('127.0.0.1', 41126) b'YES'
('127.0.0.1', 41126) b'WHY NOT?'
('127.0.0.1', 41128) b'WHY'
('127.0.0.1', 41126) b'MAYBE'
('127.0.0.1', 41130) b'WHY NOT?'
('127.0.0.1', 41126)  closing
1 serving ('127.0.0.1', 41130)
('127.0.0.1', 41124) b'NO'
('127.0.0.1', 41130) b'ARE YOU SURE'
('127.0.0.1', 41124) b'BYE'
```

```
('127.0.0.1', 41132) b'WHY'
('127.0.0.1', 41128) b'NO'
('127.0.0.1', 41124)  closing
0 serving ('127.0.0.1', 41132)
('127.0.0.1', 41128)  closing
('127.0.0.1', 41132) b'MAYBE'
('127.0.0.1', 41132) b'HELLO'
('127.0.0.1', 41130) b'WHY'
('127.0.0.1', 41132) b'NO'
('127.0.0.1', 41132) b'BYE'
('127.0.0.1', 41132)  closing
('127.0.0.1', 41130) b'BYE'
('127.0.0.1', 41130) b'WHY NOT?'
('127.0.0.1', 41130)  closing
exitting timed out . Will wait for ongoing connections to close
complete
```

1 Binary Interfacing

Motivations of binary interfacing:

1. Use of binary libraries within the script language
2. From a C, C++ code, use a python interpreter and use flexibility of script languages.

- AutoCAD vs List
- Gimp vs Scheme and Python
- Blender vs Python
- Excell vs Visual Basic
- ... Script language provides a powerful interface to Human to interact with a binary program, debuging, automating, macros etc.

1.1 ctypes

It loads dynamic libraries built in C/C++ in python run-time and let all C/C++ types encapsulated in classes so that data can be interchanged. It calls C/C++ functions within python.

class wrappers around C values: * `c_int, c_double, c_float, c_char` * `c_int * n` for an integer array of size n. `int a[10]; -> (c_int * 10)` * `c_void_p` and `c_char_p` for `void *` and `char * *` use `POINTER(type)` for pointer to type * `byref(value)` gives a pointer to value `POINTER(typeof(value))`

For structures. Inherit `ctypes.Structure` for creating a special wrapper:

```
struct Complex {
    double x;
    double y;
}
```

```
class Complex(ctypes.Structure):
    _fields_ = [("x", c_double), ("y", c_double)]
```

You need to set `func.restype` to set result type of Functions since python does not have any idea about the function prototypes. Also you can use `argtypes = (type1, type2, ...)` to set arguments type

```
lib = ctypes.CDLL("binaryfile.so")
lib.func.restype = c_double

darray = c_double * 10
dval = darray( * [ 1, 2, 4, 5, 6, 7, 8, 9, 10, 3])  # * unwraps the list in set of arguments
lib.func( c_double(arg1), c_int(arg2), dval)
# or
lib.func.argtypes = c_double, c_int, darray
lib.func(arg1, arg2, dval )
```

```
In [14]: from ctypes import *

         # use system math library
         mlib = CDLL('libm.so.6')

         #this will return an integer since result type unknown
         print(mlib.sin(c_double(3.1415926536)))

         # try again by setting return type
         mlib.sin.restype = c_double
         print(mlib.sin(c_double(3.1415926536)))
```

```
1074340347
-1.0206823934513925e-11
```

Following is a comprehensive example containing different parameter passing mechanisms in *ctest. c* compile as > `gcc -shared -o libctest.so ctest.c`
to create `libctest.so`

```
In [15]: # ctest.c contains following functions in C
         # compile as 'gcc -shared -o libctest.so ctest.c' to create libctest.so
         # struct Complex add(struct Complex , struct Complex );
         # void swap(struct Complex *, struct Complex *);
         # double sum(double f[], int n);
         # void titlecase(char *);
         # int tokenize(char sep, char *str, char t[][20]);
         # void mult(double a[][100], double b[][100], double c[][100], int n, int r , int n);

         # get the python library. searched in system path, specify full or
         # relative path if not in system library
         lib = CDLL('./libctest.so')

         class Complex(Structure):
```

```
    _fields_ = [("x", c_double), ("y", c_double)]

# struct Complex add(struct Complex , struct Complex );
lib.add.restype = Complex
r = lib.add(Complex(3.1, 4.2), Complex(1.9, 2.8))
print('1-add\n',r, r.x, r.y)

a = Complex(3, 4)
b = Complex(2, 7)

# void swap(struct Complex *, struct Complex *);
# send pointers to values to pass by pointer
lib.swap(byref(a), byref(b))
print('2-swap\n',a.x, a.y, b.x, b.y)

# double sum(double f[], int n);
# Array type. You can pass pointers as in C. BE CAREFULL ABOUT STORAGE
# YOU CAN GET A SEGFAULT in PYTHON!!!
lib.sum.restype = c_double
lib.sum.argtypes = POINTER(c_double), c_int
myarr = (c_double * 10)( * [1, 2, 1, 6, 4, 2, 7, 3, 1, 5])
print('3-sum\n',lib.sum(myarr, 10))

# void titlecase(char *);
# c_char_p can be use to 0 terminated C strings. Make sure the storage is sufficient
# use value field to get the content
lib.titlecase.argtypes = (c_char_p,)
name = c_char_p(b'the advantages of script languages over OTHER languages')
lib.titlecase(name)
print('4-titlecase\n', name.value)

# int tokenize(char sep, char *str, char t[][20]);
# this is slightly tricky t needs to store resulting tokens, so need sufficient storage
lib.tokenize.restype = c_int
tokenstype = (c_char * 20) * 20
lib.tokenize.argtypes = c_char, c_char_p, tokenstype

# create an array for the tokenize result array of 20 strings
res = tokenstype ( )
for i in range(20):
    res[i] = create_string_buffer(20)
n = lib.tokenize(b' ', b'the advantages of script languages over OTHER languages', res
tokens = [tok.value for tok in res[:n]]
print('5-tokenize\n',n,tokens)

# void mult(double a[][100], double b[][100], double c[][100], int n, int r , int n);
a = [[ 1, 2, 3, 4, 5], [11, 12, 13, 14, 15]]
b = [[1, 2], [2, 3],[4, 5], [6, 7],[8,9]]
```

```
        arrtype = (c_double * 100) * 100

        # initialize matrix a
        matA = arrtype()
        i = 0
        for row in a:
            matA[i] = (c_double * 100)( * row)
            i += 1

        # initialize matrix B
        matB = arrtype()
        i = 0
        for row in b:
            matB[i] = (c_double * 100)( * row)
            i += 1
        matC = arrtype()

        # multiply
        lib.mult.argtypes = arrtype, arrtype, arrtype, c_int, c_int, c_int
        lib.mult(matA, matB, matC, 2, 5, 2)

        # result
        print('6-mult\n',[[matC[i][j] for i in range(2)] for j in range(2)])
```

```
1-add
 <__main__.Complex object at 0x7fba500b1d08> 5.0 7.0
2-swap
 2.0 7.0 3.0 4.0
3-sum
 32.0
4-titlecase
 b'The Advantages Of Script Languages Over OTHER Languages'
5-tokenize
 8 [b'the', b'advantages', b'of', b'script', b'languages', b'over', b'OTHER', b'languages']
6-mult
 [[81.0, 291.0], [96.0, 356.0]]
```