PyCon China 2020

PyConChina 2020
PYTHON 中国开发者大会 2020

Python For Good

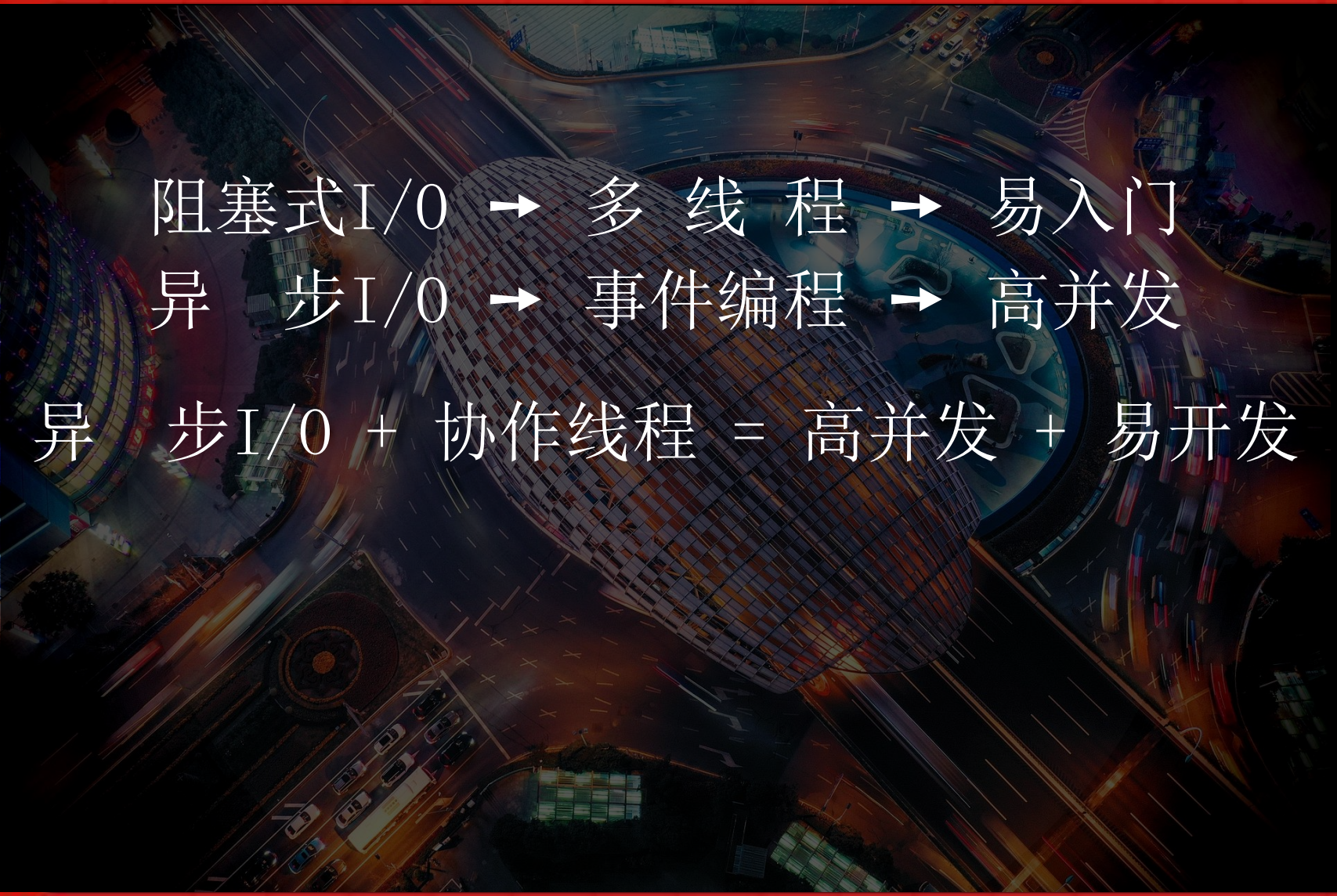
标准线程的协程替换实现

沈崴

Python1～3・重度使用者
湖州迅普信息技术有限公司

前言：什么是协程

前言：协程能带来什么



阻塞式I/O → 多 线 程 → 易入门

异　步I/O → 事件编程 → 高并发

异　步I/O + 协作线程 = 高并发 + 易开发

标准线程的协程替换实现

主题介绍

使用 Python 将 POSIX Thread 重新实现为基于协程的版本。使现有的非异步程序，无论使用何种语言开发，都能在无需修改和重新编译的情况下直接变成协程架构下的异步程序。

主题介绍

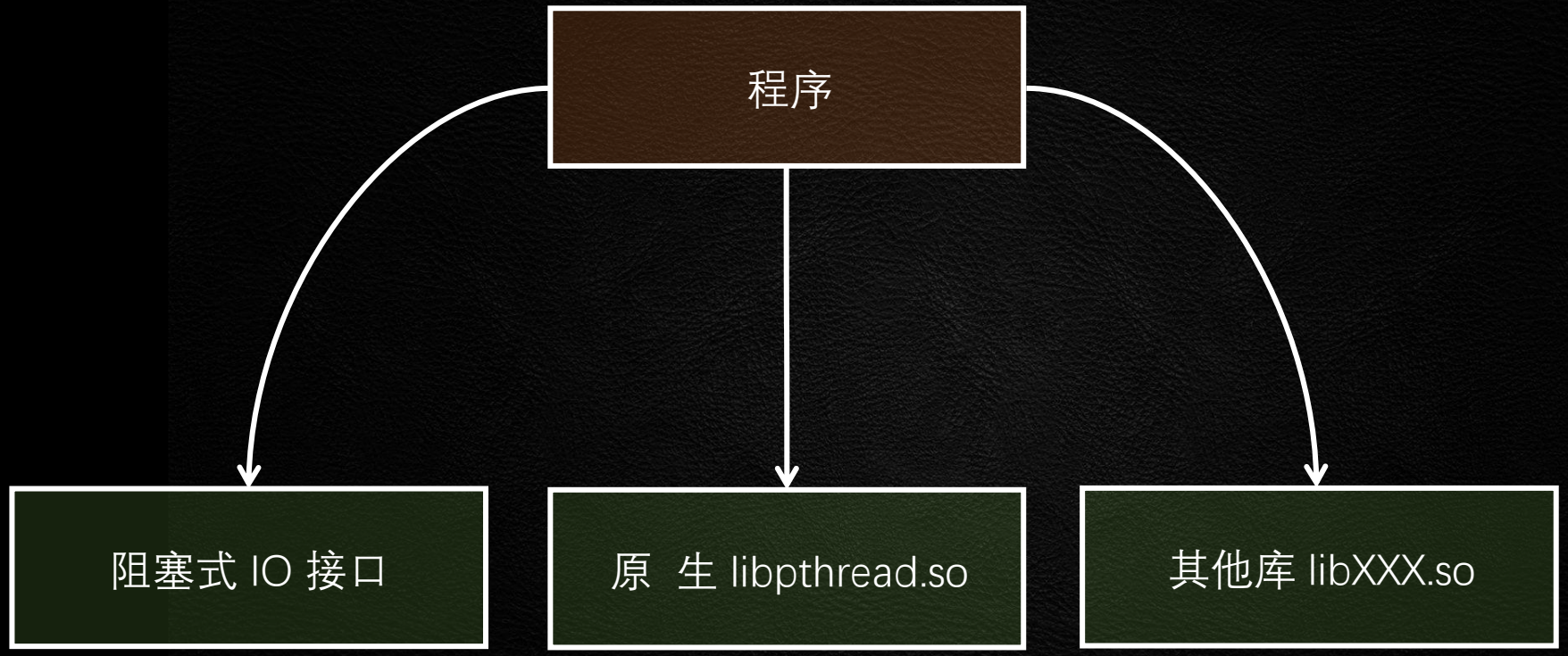
在不改动程序的情况下，把任意程序变成异步程序。

原理

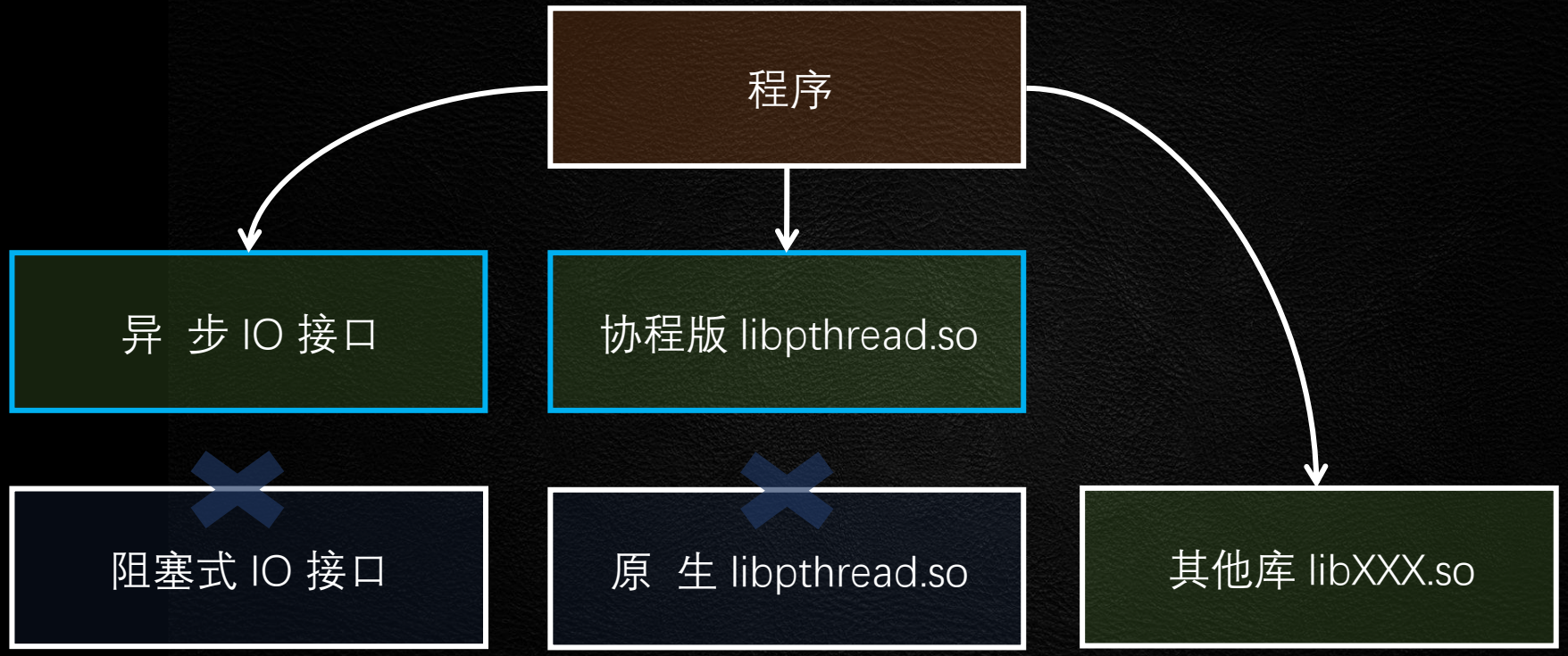
原理

把 libpthread 用协程重新实现一遍（同时把阻塞式接口用异步重新实现一遍），并在运行期注入程序，覆盖原生库。

原理



原理


程序
异 步 IO 接口
协程版 libpthread.so
阻塞式 IO 接口
原 生 libpthread.so
其他库 libXXX.so

案例

Proxychains

https://github.com/haad/proxychains/

https://github.com/rofl0r/proxychains-ng/

https://github.com/wilhelmshen/passless/

Proxychains

```
env LD_PRELOAD=libproxychains.so
```

```
wilhelmshen@william-laptop:/opt/slowdown/bin$ ./proxychains chacha20
:******@:8080/H/P/ wget https://cn.pycon.org/
--2020-11-19 16:56:38--  https://cn.pycon.org/
正在解析主机 cn.pycon.org (cn.pycon.org)... 40.83.100.99
正在连接 cn.pycon.org (cn.pycon.org)|40.83.100.99|:443... 已连接。
已发出 HTTP 请求，正在等待回应... 200 OK
长度： 1427 (1.4K) [text/html]
正在保存至： “index.html”

index.html       100%[==========>]   1.39K  --.-KB/s  用时 0.04s

2020-11-19 16:56:39 (32.7 KB/s) - 已保存 “index.html” [1427/1427])

wilhelmshen@william-laptop:/opt/slowdown/bin$
```

```
wilhelmshen@william-laptop:~/Desktop/passless/bin$ ./slowdown -vv
2020-11-19 16:56:28 INFO slowdown/0.0.1.dev2
2020-11-19 16:56:28 INFO Serving HTTP on 0.0.0.0 port 8080 ...
2020-11-19 16:56:39 INFO 127.0.0.1:42380 - pass://DEFAULT:ALL_HOSTS
:ITWORKS - 40.83.100.99:443
2020-11-19 16:56:39 --- GET '/P/0a1eae547c5d4fb8.mp4' 127.0.0.1 'Mo
zilla/5.0 (X11; Ubuntu; Linux x86_64; rv:52.0) Gecko/20100101 Firef
ox/52.0'
```

技术方案

Gevent

https://www.gevent.org/

Cython

https://www.cython.org/

```
cdef int htoi(char *s):
    cdef int value, c

    c = (<unsigned char *>s)[0]
    if isupper(c):
        c = tolower(c)
    if c >= 48 and c <= 57:
        value = (c - 48) * 16
    else:
        value = (c - 97 + 10) * 16

    c = (<unsigned char *>s)[1]
    if isupper(c):
        c = tolower(c)
    if c >= 48 and c <= 57:
        value += c - 48
    else:
        value += c - 97 + 10

    return value
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
```

demo.pyx 1,1 全部

```
static int htoi(char *s) {
  int value;
  int c;
  int result;
  int t1;
  int t2;
  c = (((unsigned char *)s)[0]);
  t1 = (isupper(c) != 0);
  if (t1) {
    c = tolower(c);
  }
  t2 = ((c >= 48) != 0);
  if (t2) {} else {
    t1 = t2;
    goto label2_bool_binop_done;
  }
  t2 = ((c <= 57) != 0);
  t1 = t2;
  label2_bool_binop_done:;
  if (t1) {
    value = ((c - 48) * 16);
    goto label1;
  }
  value = (((c - 97) + 10) * 16);
  label1:;
  c = (((unsigned char *)s)[1]);
  t1 = (isupper(c) != 0);
  if (t1) {
    c = tolower(c);
  }
  t2 = ((c >= 48) != 0);
  if (t2) {} else {
    t1 = t2;
    goto label4_bool_binop_done;
  }
  t2 = ((c <= 57) != 0);
  t1 = t2;
  label4_bool_binop_done:;
  if (t1) {
    value = (value + (c - 48));
    goto label3;
  }
  value = (value + ((c - 97) + 10));
  label3:;
  result = value;
  goto label0;
  label0:;
  return result;
}
~
```

demo.c 1,1 顶端

思路验证

实验性覆盖 pthread_create() 接口

```
#!/usr/bin/make

all: mythread.so libmythread.so

mythread.so: mythread.pyx
	cython3 -I. -Includes mythread.pyx
	gcc -I/usr/include/python3.7m -fPIC -c mythread.c -o mythre
ad.o
	gcc -shared -lpython3.7m mythread.o -o mythread.so

libmythread.so: libmythread.c
	gcc -I/usr/include/python3.7m -fPIC -c libmythread.c -o lib
mythread.o
	gcc -shared -Wl,-init,libmythread__init   libmythread.o -o
libmythread.so
~
```

Makefile [只读] 13,0-1 全部

```
#include <Python.h>
#include "mythread.h"

void libmythread_init()
{
    PyImport_AppendInittab("mythread", PyInit_mythread);
    Py_Initialize();
    PyImport_Module("mythread");
}
~
~
~
~
~
~
~
~
```

libmythread.c [只读] 1,1 全部

```
cimport libc.errno
cimport libc.stdio

cdef extern from '<pthread.h>':

    ctypedef struct pthread_t:
        pass

    ctypedef struct pthread_attr_t:
        pass

cdef public int pthread_create(pthread_t *newthread,
                               const pthread_attr_t *attr,
                               void *(*start_routine) (void *),
                               void *arg):

    libc.stdio.printf(b'\n*****************\n')
    libc.stdio.printf(b'PyCon China 2020!\n')
    libc.stdio.printf(b'*****************\n\n')

    return libc.errno.EINVAL
~
~
~
~
~
~
~
~
~
~
~
~
~
```

mythread.pyx 1,1 全部

```
wilhelmshen@william-laptop:~/Desktop/mythread$ make
cython3 -I. -Includes mythread.pyx
gcc -I/usr/include/python3.7m -fPIC -c mythread.c -o mythread.o
gcc -shared -lpython3.7m mythread.o -o mythread.so
gcc -I/usr/include/python3.7m -fPIC -w -c libmythread.c -o libmythread.o
gcc -shared -Wl,-init,libmythread_init libmythread.o -o libmythread.so
wilhelmshen@william-laptop:~/Desktop/mythread$ export LD_PRELOAD=./libmythread.so:./mythread.so
wilhelmshen@william-laptop:~/Desktop/mythread$ python3.7
Python 3.7.3 (default, Jul 25 2020, 13:03:44)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import _thread
>>> _thread.start_new_thread(lambda: None, ())

**********
PyCon China 2020!
**********

Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
RuntimeError: can't start new thread
>>>
```

Makefile

每个程序员一生中只需要一个 Makefile

```
#!/usr/bin/make

CYTHON := cython3
PYTHON := python3

define PY_CONFIG_VAR_CC
import distutils.sysconfig; \
print (distutils.sysconfig.get_config_var('CC'))
endef
CC := $(shell $(PYTHON) -c "$(PY_CONFIG_VAR_CC)")

define PY_CONFIG_VAR_EXT_SUFFIX
import distutils.sysconfig; \
print (distutils.sysconfig.get_config_var('EXT_SUFFIX') or '.so')
endef
EXT_SUFFIX := $(shell $(PYTHON) -c "$(PY_CONFIG_VAR_EXT_SUFFIX)")

PYX_SOURCES := $(wildcard *.pyx)
SOURCES := $(PYX_SOURCES:.pyx=.c)
MODULES := $(PYX_SOURCES:.pyx=$(EXT_SUFFIX))

all: $(MODULES)

%$(EXT_SUFFIX): %.c %.setup
	$(PYTHON) $(patsubst %.c,%,$<).setup build_ext --inplace

%.c: %.pyx
	$(CYTHON) -IIncludes $<

clean:
	rm -rf build $(MODULES) $(SOURCES) __pycache__
	find . -name '*.pyc' -exec rm {} \;
	find . -name '__pycache__' -exec rm -d {} \;

.PRECIOUS: $(SOURCES)

.PHONY: all clean
~
~
```

Makefile 1,1 全部

```
def hello():
    print ('PyCon China 2020!')
~
~
~
~
~
~
~
~
~
~
~
~
~
~
```

demo.pyx [只读] 1,1 全部

```
import os.path
import sys
from distutils.core import setup
from distutils.extension import Extension
name = os.path.splitext(os.path.basename(__file__))[0]
setup(
    ext_modules=
    [
        Extension(
            name=name,
            sources=
            [
                os.path.join(
                    os.path.abspath(sys.path[0]),
                    f'{name}.c'
                )
            ]
        )
    ]
)
~
```

demo.setup 21,0-1 全部

```
wilhelmshen@william-laptop:~/Desktop/demo$ ls
demo.pyx  demo.setup  Makefile
wilhelmshen@william-laptop:~/Desktop/demo$ make
cython3 -IIncludes demo.pyx
python3 demo.setup build_ext --inplace
running build_ext
building 'demo' extension
creating build
creating build/temp.linux-x86_64-3.7
creating build/temp.linux-x86_64-3.7/home
creating build/temp.linux-x86_64-3.7/home/wilhelmshen
creating build/temp.linux-x86_64-3.7/home/wilhelmshen/Desktop
creating build/temp.linux-x86_64-3.7/home/wilhelmshen/Desktop/demo
x86_64-linux-gnu-gcc -pthread -DNDEBUG -g -fwrapv -O2 -Wall -g -fstack-protector-strong -Wformat -Werror=format-
security -Wdate-time -D_FORTIFY_SOURCE=2 -fPIC -I/usr/include/python3.7m -c /home/wilhelmshen/Desktop/demo/demo.
c -o build/temp.linux-x86_64-3.7/home/wilhelmshen/Desktop/demo/demo.o
x86_64-linux-gnu-gcc -pthread -shared -Wl,-O1 -Wl,-Bsymbolic-functions -Wl,-z,relro -Wl,-z,relro -g -fstack-prot
ector-strong -Wformat -Werror=format-security -Wdate-time -D_FORTIFY_SOURCE=2 build/temp.linux-x86_64-3.7/home/w
ilhelmshen/Desktop/demo/demo.o -o /home/wilhelmshen/Desktop/demo/demo.cpython-37m-x86_64-linux-gnu.so
wilhelmshen@william-laptop:~/Desktop/demo$ ls
build  demo.c  demo.cpython-37m-x86_64-linux-gnu.so  demo.pyx  demo.setup  Makefile
wilhelmshen@william-laptop:~/Desktop/demo$ python3.7
Python 3.7.3 (default, Jul 25 2020, 13:03:44)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import demo
>>> demo.hello()
PyCon China 2020!
>>>
```

完整版

```
cdef class Attr:

    cdef int  detachstate
    cdef size_t guardsize
    cdef sched_param schedparam
    cdef int schedpolicy
    cdef int inheritsched
    cdef int scope
    cdef void  *stackaddr
    cdef size_t stacksize

    cdef int init(self, const pthread_attr_t *attr):
        if NULL == attr:
            return self.set_default()
        else:
            return self.set(attr)

    cdef int set_default(self):
        cdef pthread_attr_t attr
        cdef int fail = pthread_attr_init(&attr)
        if fail:
            return fail
        return self.set(&attr)

    cdef int set(self, const pthread_attr_t *attr):
        cdef int fail
        fail = pthread_attr_getdetachstate(attr, &self.detachstate)
        if fail:
            return fail
        fail = pthread_attr_getguardsize(attr, &self.guardsize)
        if fail:
            return fail
        fail = pthread_attr_getschedparam(attr, &self.schedparam)
        if fail:
            return fail
        fail = pthread_attr_getschedpolicy(attr, &self.schedpolicy)
        if fail:
            return fail
        fail = pthread_attr_getinheritsched(attr, &self.inheritsched)
        if fail:
            return fail
        fail = pthread_attr_getscope(attr, &self.scope)
        if fail:
            return fail
        fail = \
            pthread_attr_getstack(
                attr,
                &self.stackaddr,
                &self.stacksize
            )
        if fail:
            return fail
```



```
cdef class Thread:

    cdef bytes name
    cdef Attr  attr
    cdef int cancelstate
    cdef int canceltype
    cdef void *(*start_routine) (void *) nogil
    cdef void *arg
    cdef void *retval

    cdef int init \
        (
            self,
            bytes name,
            Attr  attr,
            void  *(*start_routine) (void *) nogil,
            void  *arg
        ):
        self.name = name
        self.attr = attr
        self.cancelstate = PTHREAD_CANCEL_DISABLE
        self.canceltype  = PTHREAD_CANCEL_DEFERRED
        self.start_routine = start_routine
        self.arg = arg
        self.retval = NULL
        return 0

def run():
    g = <Greenlet?>gevent.getcurrent()
    if g._start_event is None:
        g._start_event = _cancelled_start_event
    g._start_event.stop()
    g._start_event.close()
    g._start_event = _start_completed_event
    t = <Thread?>g.args[0]
    cdef void *start_routine = t.start_routine
    cdef void *arg = t.arg
    del t
    del g
    cdef void *retval = (<void *(*) (void *) nogil>start_routine)(arg)
    g   = <Greenlet?>gevent.getcurrent()
    t   = <Thread?>g.args[0]
    key = \
            <unsigned long>     \
          <libc.stdint.uintptr_t> \
        <cpython.object.PyObject*>g
    t.retval = retval
    g._exc_info = (None, None, None)
    g.value = None
    if g._links and not g._notifier:
        g._notifier = (<greenlet>g) \
                    .    parent     \
```



```
                    .    parent     \
                    .     loop      \
                    . run_callback(g._notify_links)
    del g.run
    g.args = ()
    g.kwargs.clear()
    del greenlets[key]

# Create a new thread, starting with execution of START-ROUTINE
# getting passed ARG.  Creation attributed come from ATTR.  The new
# handle is stored in *NEWTHREAD.
cdef public int pthread_create \
    (
        pthread_t *newthread,
        const pthread_attr_t *attr,
        void *(*start_routine) (void *) nogil,
        void *arg
    ):
    pAttr = Attr()
    cdef int fail = pAttr.init(attr)
    if fail:
        return fail
    t = Thread()
    g = Greenlet()
    t.init(g.name.encode(), pAttr, start_routine, arg)
    g.run  =  run
    g.args = (t, )
    g.start()
    key = \
              <unsigned long>     \
          <libc.stdint.uintptr_t> \
        <cpython.object.PyObject*>g
    greenlets[key] = g
    newthread[0] = \
                  <pthread_t>       \
          <libc.stdint.uintptr_t> \
        <cpython.object.PyObject*>g
    return 0

# Terminate calling thread.
#
# The registered cleanup handlers are called via exception handling
# so we cannot mark this function with __THROW.
cdef public void pthread_exit(void *retval):
    g = gevent.getcurrent()
    cdef pthread_t th = \
                  <pthread_t>       \
          <libc.stdint.uintptr_t> \
        <cpython.object.PyObject*>g
    del g
    __cancel(th, retval)
```

C10K 计划

https://github.com/wilhelmshen/c10k/

libpthread

pthread_create pthread_exit pthread_join pthread_tryjoin_np pthread_timedjoin_np pthread_detach
pthread_t pthread_self pthread_equal pthread_getattr_np pthread_setschedparam pthread_getschedparam
pthread_setschedprio pthread_getname_np pthread_setname_np pthread_getconcurrency
pthread_setconcurrency pthread_yield pthread_setaffinity_np pthread_getaffinity_np pthread_once
pthread_setcancelstate pthread_setcanceltype pthread_cancel pthread_testcancel pthread_kill

pthread_mutex_init pthread_mutex_destroy pthread_mutex_trylock pthread_mutex_lock
pthread_mutex_timedlock pthread_mutex_unlock pthread_mutex_getprioceiling pthread_mutex_setprioceiling
pthread_mutex_consistent pthread_mutex_consistent_np

pthread_rwlock_init pthread_rwlock_destroy pthread_rwlock_rdlock pthread_rwlock_tryrdlock
pthread_rwlock_timedrdlock pthread_rwlock_wrlock pthread_rwlock_trywrlock pthread_rwlock_timedwrlock
pthread_rwlock_unlock

pthread_cond_init pthread_cond_destroy pthread_cond_signal pthread_cond_broadcast pthread_cond_wait
pthread_cond_timedwait

pthread_spin_init pthread_spin_destroy pthread_spin_lock pthread_spin_trylock pthread_spin_unlock

pthread_barrier_init pthread_barrier_destroy pthread_barrier_wait

pthread_key_create pthread_key_delete pthread_getspecific pthread_setspecific pthread_atfork

技术内幕

Greenlet 对象

gevent/src/gevent/_gevent_cgreenlet.pxd

```
typedef struct _greenlet {
    PyObject_HEAD
    char* stack_start;
    char* stack_stop;
    char* stack_copy;
    intptr_t stack_saved;
    struct _greenlet* stack_prev;
    struct _greenlet* parent;
    PyObject* run_info;
    struct _frame* top_frame;
    int recursion_depth;
    PyObject* weakreflist;
#if PY_VERSION_HEX >= 0x030700A3
    _PyErr_StackItem* exc_info;
    _PyErr_StackItem exc_state;
#else
    PyObject* exc_type;
    PyObject* exc_value;
    PyObject* exc_traceback;
#endif
    PyObject* dict;
#if PY_VERSION_HEX >= 0x030700A3
    PyObject* context;
#endif
} PyGreenlet;
~
~
~
```

greenlet.h 1,1 全部

```
cdef class Greenlet(greenlet):
    cdef readonly object value
    cdef readonly tuple args
    cdef readonly dict kwargs
    cdef readonly object spawning_greenlet
    cdef readonly _Frame spawning_stack
    cdef public dict spawn_tree_locals

    cdef list _links
    cdef tuple _exc_info
    cdef object _notifier
    cdef object _start_event
    cdef str _formatted_info
    cdef object _ident
~
~
~
~
~
~
~
~
~
~
~
~
```

_gevent_cgreenlet.pxd 15,0-1 全部

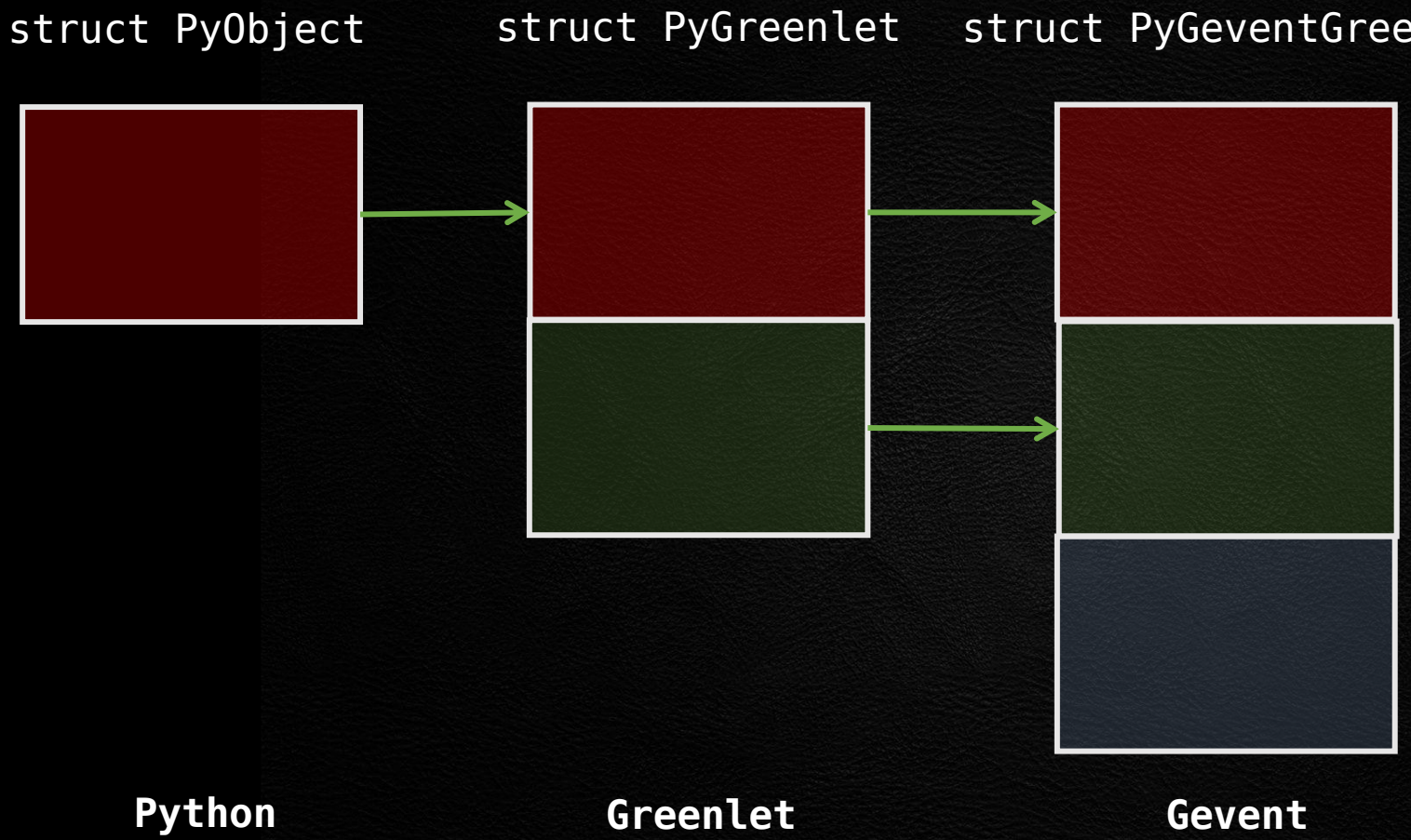
```
typedef struct _greenlet {
    PyObject_HEAD
    char* stack_start;
    char* stack_stop;
    char* stack_copy;
    intptr_t stack_saved;
    struct _greenlet* stack_prev;
    struct _greenlet* parent;
    PyObject* run_info;
    struct _frame* top_frame;
    int recursion_depth;
    PyObject* weakreflist;
#if PY_VERSION_HEX >= 0x030700A3
    _PyErr_StackItem* exc_info;
    _PyErr_StackItem exc_state;
#else
    PyObject* exc_type;
    PyObject* exc_value;
    PyObject* exc_traceback;
#endif
    PyObject* dict;
#if PY_VERSION_HEX >= 0x030700A3
    PyObject* context;
#endif
} PyGreenlet;
~
~
~
```

greenlet.h 1,1 全部

```
typedef struct _gevent_greenlet {
        PyGreenlet __pyx_base;
        void     * __pyx_vtab;
        PyObject *value;
        PyObject *args;
        PyObject *kwargs;
        PyObject *spawning_greenlet;
        PyObject *spawning_stack;
        PyObject *spawn_tree_locals;
        PyObject *_links;
        PyObject *_exc_info;
        PyObject *_notifier;
        PyObject *_start_event;
        PyObject *_formatted_info;
        PyObject *_ident;
} PyGeventGreenlet;

~
~
~
~
~
~
~
~
~
~
```

_gevent_cgreenlet.h 17,0-1 全部

Greenlet对象


struct PyObject
struct PyGreenlet
struct PyGeventGreenlet
Python
Greenlet
Gevent

内存就是要整整齐齐

C语言
面向对象编程

继承：访问父类


```
curr = gevent.getcurrent()
cdef PyGeventGreenlet *g = <PyGeventGreenlet*>curr
run = <object>g.__pyx_base.run_info
```

```
typedef struct _greenlet {
    PyObject_HEAD
    char* stack_start;
    char* stack_stop;
    char* stack_copy;
    intptr_t stack_saved;
    struct _greenlet* stack_prev;
    struct _greenlet* parent;
    PyObject* run_info;
    struct _frame* top_frame;
    int recursion_depth;
    PyObject* weakreflist;
#if PY_VERSION_HEX >= 0x030700A3
    _PyErr_StackItem* exc_info;
    _PyErr_StackItem exc_state;
#else
    PyObject* exc_type;
    PyObject* exc_value;
    PyObject* exc_traceback;
#endif
    PyObject* dict;
#if PY_VERSION_HEX >= 0x030700A3
    PyObject* context;
#endif
} PyGreenlet;
~
~
~
```

greenlet.h 1,1 全部

```
typedef struct _gevent_greenlet {
        PyGreenlet __pyx_base;
        void     * __pyx_vtab;
        PyObject *value;
        PyObject *args;
        PyObject *kwargs;
        PyObject *spawning_greenlet;
        PyObject *spawning_stack;
        PyObject *spawn_tree_locals;
        PyObject * _links;
        PyObject * _exc_info;
        PyObject * _notifier;
        PyObject * _start_event;
        PyObject * _formatted_info;
        PyObject * _ident;
} PyGeventGreenlet;
~
~
~
~
~
~
~
~
~
~
~
```

_gevent_cgreenlet.h 17,0-1 全部

类型转换


```
curr = gevent.getcurrent()
cdef PyObject *obj = <PyObject*>curr
cdef PyGreenlet *g1 = <PyGreenlet*>obj
cdef PyGeventGreenlet *g2 = <PyGeventGreenlet*>g1
```

重新开放私有属性

```
cdef class Greenlet(greenlet):
    cdef readonly object value
    cdef readonly tuple args
    cdef readonly dict kwargs
    cdef readonly object spawning_greenlet
    cdef readonly _Frame spawning_stack
    cdef public dict spawn_tree_locals

    cdef list _links
    cdef tuple _exc_info
    cdef object _notifier
    cdef object _start_event
    cdef str _formatted_info
    cdef object _ident
```

```
ctypedef struct PyGeventGreenlet:

    PyGreenlet __pyx_base
    void     * __pyx_vtab
    cpython.object.PyObject *value
    cpython.object.PyObject *args
    cpython.object.PyObject *kwargs
    cpython.object.PyObject *spawning_greenlet
    cpython.object.PyObject *spawning_stack
    cpython.object.PyObject *spawn_tree_locals
    cpython.object.PyObject *_links
    cpython.object.PyObject *_exc_info
    cpython.object.PyObject *_notifier
    cpython.object.PyObject *_start_event
    cpython.object.PyObject *_formatted_info
    cpython.object.PyObject *_ident

ctypedef class gevent._greenlet.Greenlet [object PyGeventGreenlet]:

    cdef object value
    cdef object args
    cdef object kwargs
    cdef object spawning_greenlet
    cdef object spawning_stack
    cdef object spawn_tree_locals
    cdef object _links
    cdef object _exc_info
    cdef object _notifier
    cdef object _start_event
    cdef object _formatted_info
    cdef object _ident
```

gevent_cgreenlet.pxd 1,1 全部 | gevent_cgreenlet.pyx 32,0-1 全部

重新开放私有属性


```
g = <Greenlet?>gevent.getcurrent()
print (g._start_event)
```

pthread_exit()

greenlet 堆栈管理细节

实现思路


```
cdef public void pthread_exit():
    gevent.getcurrent().parent.switch()
```

greenlet：所有 switch() 调用必须返回

VS

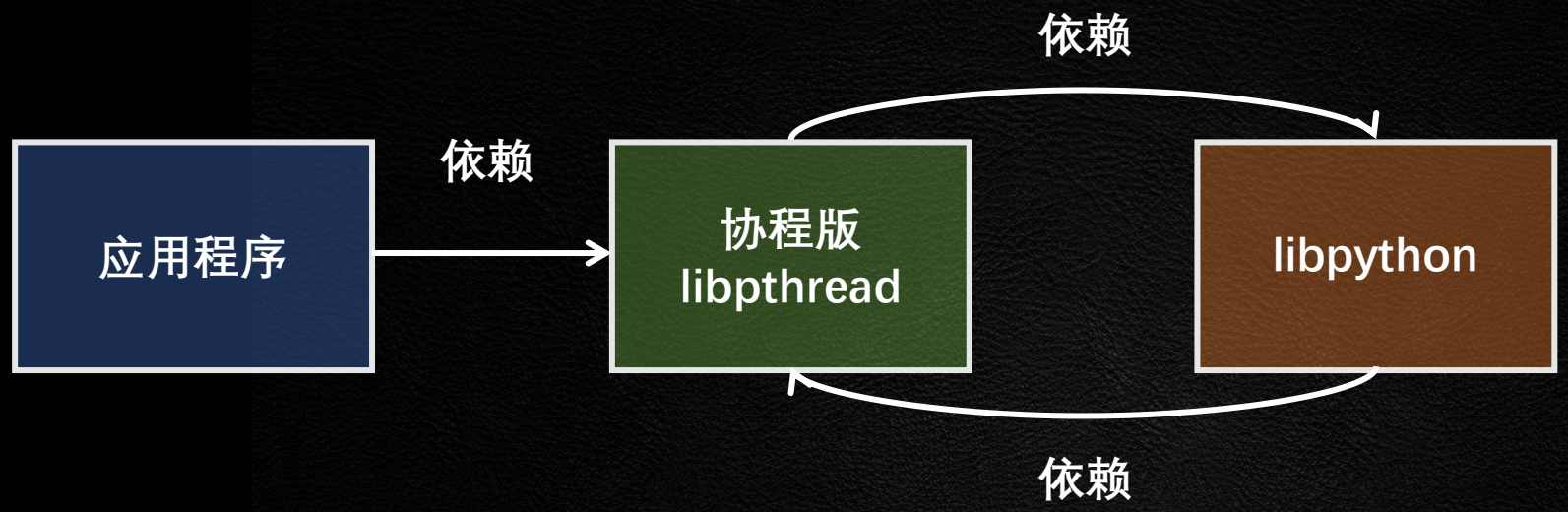
pthread_exit()：立即终止执行,禁止线程/协程栈重入

```
cdef public void pthread_exit():
    cdef PyObject *ret = gevent.getcurrent().parent.switch()
    if NULL == ret:
        PyErr_Print() # will meet "GreenletExit" here
    else:
        print ('thread still alive')

cdef void example():
    pthread_exit()
    print ('should not be here')

example()  # will print "should not be here"
```

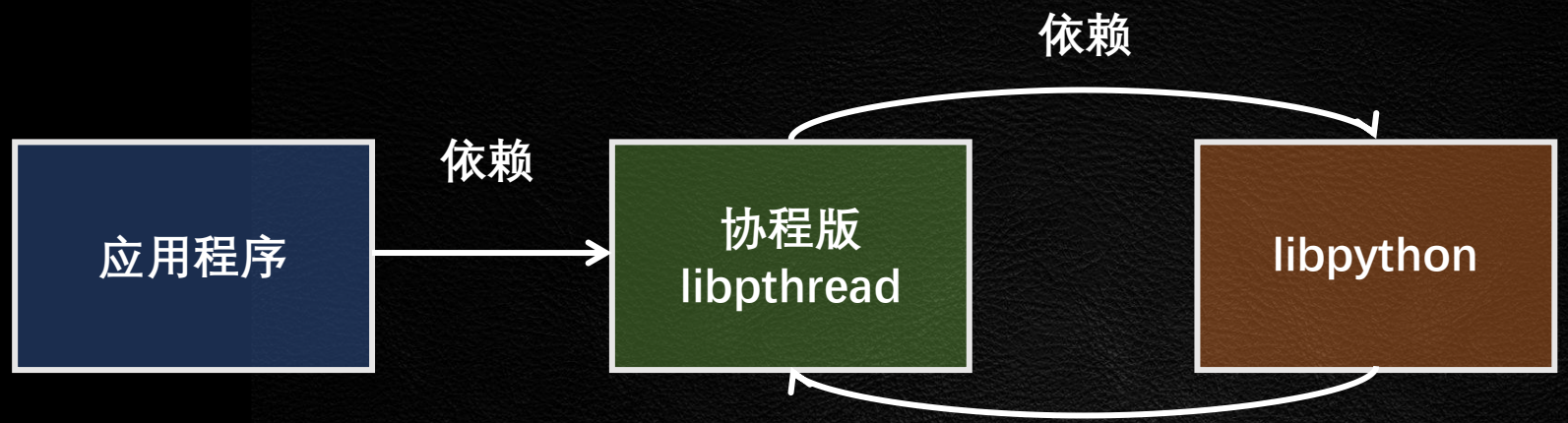
```
# Terminate calling thread.
#
# The registered cleanup handlers are called via exception handling
# so we cannot mark this function with __THROW.
cdef public void pthread_exit(void *retval):
    g = gevent.getcurrent()
    t = <Thread?>g.args[0]
    if retval != NULL:
        t.retval = retval
    g._exc_info = (None, None, None)
    g.value = None
    if g._links and not g._notifier:
        g._notifier = (<greenlet>g) \
                    .    parent     \
                    .     loop      \
                    . run_callback(g._notify_links)
    g.args = ()
    g.kwargs.clear()
    del g.run
    del t
    cdef cpython.object.PyObject *pMyHub = \
        <cpython.object.PyObject*>(<greenlet>g).parent
    (<PyGreenlet*>g).stack_start = NULL
    del g
    PyGreenlet_Switch(pMyHub, NULL)
~
~
~
```

循环依赖

循环依赖



循环依赖



pthread_mutex_init
pthread_mutex_lock
pthread_mutex_unlock
pthread_cond_init
pthread_cond_signal
pthread_cond_wait

GIL

灵魂拷问

- Python 本质上是单线程

  协程：巧了，我也是

- Python 线程主要用于解决 I/O 阻塞问题

  协程：这点我更擅长了

灵魂拷问

- Python 本质上是单线程

  协程：巧了，我也是

- Python 线程主要是为了避开GIL而生

  协程：这点我更擅长了

Python协程大规模使用
至少已有17年历史

- Python 本质上是单线程

协程：巧了，我也是

Python伪线程的意义何在？

- Python 线程主要用于避开 I/O 阻塞

协程：这点我更擅长了

```
# Lock a mutex.
cdef public int pthread_mutex_lock(pthread_mutex_t *mutex):
    if 0 == (<libc.stdint.uintptr_t*>mutex)[0]:
        return 0
    key = <unsigned long>(<libc.stdint.uintptr_t*>mutex)[0]
    try:
        pMutex = <Mutex?>mutexes[key]
    except KeyError:
        return libc.errno.ESRCH
    return pMutex.acquire(True, None)
```

访问原版接口

```
cimport libc.stdio
cimport libc.stdlib
cimport posix.dlfcn

cdef extern from '<dlfcn.h>':

    void *RTLD_NEXT  # ((void *) -1l)

cdef void* load_sym(char *symname, void *proxyfunc) nogil:
    cdef void *funcptr = posix.dlfcn.dlsym(RTLD_NEXT, symname)
    if NULL == funcptr:
        libc.stdio.fprintf(
            libc.stdio.stderr,
            b'Cannot load symbol \'%s\' %s',
            symname,
            posix.dlfcn.dlerror()
        )
        libc.stdlib.exit(1)
    if proxyfunc == funcptr:
        libc.stdio.fprintf(
            libc.stdio.stderr,
            b'circular reference detected, aborting!\n'
        )
        libc.stdlib.abort()
    return funcptr

cdef void *real_pthread_create = \
    load_sym(b'pthread_create', <void*>pthread_create)
```

C10K 计划

https://github.com/wilhelmshen/c10k/

PyCon China 2020

Python For Good

PyConChina 2020 | PYTHON 中国开发者大会 2020

THANK YOU

 wilhelmshen

 wilhelmshen

 wilhelmshen

 wilhelmshen