Global Interpreter Lock

Андрей Светлов

asvetlov.blogspot.com

Общеизвестные факты

Питон использует потоки OS

Одновременно работает только один поток

Во всех бедах виноват GIL

Interpreter State

```
struct PyInterpreterState
    struct PyInterpreterState *next;
    struct PyThreadState *tstate_head;

    PyObject *modules;
    PyObject *sysdict;
    PyObject *builtins;

    /* Информация о кодеках и т.д. */
```

Thread State

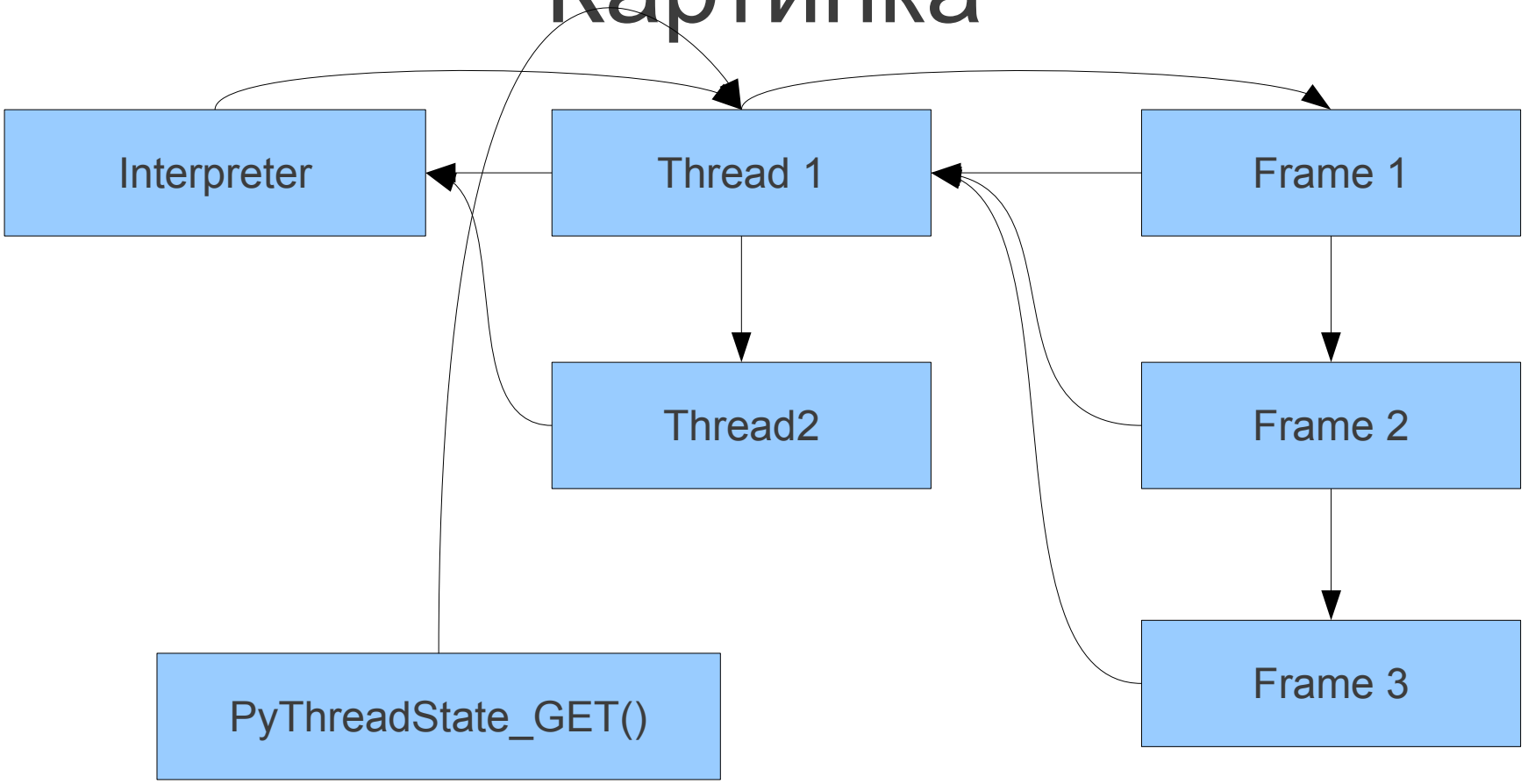
```
struct PyThreadState
    struct PyThreadState *next;
    PyInterpreterState *interp;

    struct PyFrameObject *frame;
    int recursion_depth;
    PyObject *exc_type;
    PyObject *exc_value;
    PyObject *exc_traceback;
    long thread_id;
```

Frame

```
struct PyFrameObject
    struct PyFrameObject *f_back;
    PyCodeObject *f_code;
    PyObject *f_builtins;
    PyObject *f_globals;
    PyObject *f_locals;
    PyThreadState *f_tstate;
    int f_lasti;
```

Картинка


Interpreter
Thread 1
Frame 1
Thread2
Frame 2
Frame 3
PyThreadState_GET()

GIL старый и новый

Старый	Новый (python 3.2)
Переключается каждые 100 инструкций	Переключается запросу, но не чаще чем раз в 5мс
Может промахиваться, тратя процессорное время (David Beazley)	Нет ложных срабатываний

Место переключения

```
def f():
    for i in xrange(100):
        print i

import dis
dis.dis(f)
```

```
2     0 SETUP_LOOP      25 (to 28)
      3 LOAD_GLOBAL     0 (xrange)
      6 LOAD_CONST      1 (100)
      9 CALL_FUNCTION   1
     12 GET_ITER
>>   13 FOR_ITER        11 (to 27)
     16 STORE_FAST      0 (i)
3    19 LOAD_FAST       0 (i)
     22 PRINT_ITEM
     23 PRINT_NEWLINE
     24 JUMP_ABSOLUTE   13
>>   27 POP_BLOCK
>>   28 LOAD_CONST      0 (None)
     31 RETURN_VALUE
```

Старый ceval

```
PyThreadState *tstate = PyThreadState_GET();
if (--_Py_Ticker < 0) {
    /* ... */
    _Py_Ticker = _Py_CheckInterval;
    if (interpreter_lock) {
        PyThreadState_Swap(NULL);
        PyThread_release_lock(
            interpreter_lock);
        /* Переключение */
        PyThread_acquire_lock(
            interpreter_lock, 1);
        PyThreadState_Swap(tstate);
    }
}
```

Новый ceval

```
if (_Py_atomic_load_relaxed(
            &gil_drop_request)) {
    PyThreadState_Swap(NULL);
    drop_gil(tstate);
    /* Переключение */
    take_gil(tstate);
    PyThreadState_Swap(tstate);
}
```

Управление переключением

- старый

```
sys.getcheckinterval()
sys.setcheckinterval(val)
100 тиков
```

- новый

```
sys.getswitchinterval()
sys.setswithcinterval(val)
0.005 сек (5 миллисекунд)
```

Промежуточный вывод:

Python 3.2 имеет гораздо лучший переключатель GIL

Создание потока

```
import threading

def f(count):
    for i in range(count):
        pass

th = threading.Thread(target=f, args=(10,))

th.start()
th.join()
```

Со стороны интерпретатора

```
struct bootstate {
    PyInterpreterState *interp;
    PyObject *func;
    PyObject *args;
    PyObject *keyw;
    PyThreadState *tstate;
};
```

Запуск потока

```
static PyObject *
thread_PyThread_start_new_thread(PyObject *self, PyObject *fargs){
    PyObject *func, *args, *keyw = NULL;
    PyArg_UnpackTuple(fargs, "start_new_thread", 2, 3,
                      &func, &args, &keyw);
    struct bootstate *boot = PyMem_NEW(struct bootstate, 1);
    boot->interp = PyThreadState_GET()->interp;
    boot->func = func; boot->args = args; boot->keyw = keyw;
    boot->tstate = _PyThreadState_Prealloc(boot->interp);
    Py_INCREF(func); Py_INCREF(args); Py_XINCREF(keyw);
    ident = PyThread_start_new_thread(t_bootstrap, (void*) boot);
    return PyLong_FromLong(ident);
}
```

Потоковый пускач

```
static void t_bootstrap(struct bootstate *boot) {
    PyThreadState * tstate = boot->tstate;
    tstate->thread_id = PyThread_get_thread_ident();
    _PyThreadState_Init(tstate); PyEval_AcquireThread(tstate);
    PyObject *res = PyEval_CallObjectWithKeywords(boot->func,
        boot->args, boot->keyw);
    if (res == NULL)
        /* Напечатать ошибку в sys.stderr */
    else
        Py_DECREF(res);
    Py_DECREF(boot->func); Py_DECREF(boot->args);
    Py_XDECREF(boot->keyw); PyMem_DEL(boot);
    PyThreadState_Clear(tstate); PyThreadState_DeleteCurrent();
    PyThread_exit_thread();
}
```

Python embedding

Поток создается не Питоном, но он должен вызывать питоновский код

Нужно регистрировать этот поток по подобию Modules/_threadmodule.c

- `PyEval_InitThreads`
- `PyThreadState_New`
- `_PyThreadState_Init`
- `PyEval_AcquireThread`

Embedding

- Никогда так не делайте, если есть выбор
- Это попросту неудобно
- Extending (написание Python C Extensions) решает все проблемы
- Если тем не менее создается именно embedding — соблюдайте аккуратность
- Запуск Питоновского кода, не имеещего ThreadState и не захватившего GIL ведет к краху интерпретатора

Расширения на C

```
Py_BEGIN_ALLOW_THREADS
n = read(self->fd, pbuf.buf, len);
Py_END_ALLOW_THREADS
```

- Единственная операция, доступная потоку не владеющему GIL — попытаться захватить его

Освобождение GIL

Освобождают

- Операции ввода-вывода (файлы-сокеты)
- Прочие системные вызовы
- Долго работающие алгоритмы (regex, lxml)

Не освоождают

- Вызовы для работы с питоновскими объектами (list, dict)
- C Extensions, созданные кривыми руками

Освобождение GIL

- Чем больше времени поток работает без GIL — тем лучше всем
- Лучше освободить один раз на функцию, чем сто раз освободить-захватить
- Все системные вызовы достаточно долгие. Вызовы сторонних библиотек, как правило — тоже.

Обработка сигналов

```
import threading
threads = []
running = True

def f():
    while running:
        pass

for i in range(1):
    th = threading.Thread(target=f)
    threads.append(th)
    th.start()

for th in threads:
    th.join()
```

- Python 2.7

Висит вечно. Не реагирует на <Ctrl+C>

- Python 3.2

Можно прервать посредством <Ctrl+C> + <Ctrl+C>

Добавляем обработчик сигнала

```
import signal
running = True

def sig_handler(sig_num,
                frame):
    global running
    running = False

signal.signal(
    signal.SIGINT,
    sig_handler)
```

- Python 2.7

Обработчик не вызывается

- Python 3.2

Успешно завершаются порожденные потоки, давая возможность главному потоку закончить работу

Почему висим?

- Сигнал должен выполнится в главном потоке.
- Главный поток блокирован `th.join()`
- Python 3.2 умеет обрабатывать сигналы, прерывающие блокировки.
- Для исполнения кода обработчика сигнала нужно, чтобы `ceval loop` получил управление и запустил т.н. `PendingCall`

Как чинить?

- Не ждать вечно
- Python 2.7 не имеет «родного» интерфейса для блокировки с таймаутом. Используется неблокирующие вызовы и `time.sleep()`
- Можно сделать служебный канал (pipe), сокет или еще какой объект с файловым дескриптором, который можно передать `select/poll`. `select` и `poll` имеют таймаут
- Использовать 3.2, у которого таймауты хорошие

Еще один вывод:
Python 3.2 гораздо лучше поддерживает потоки

Делайм таймаут на неблокирующем RLock

```
endtime = time.time() + timeout
delay = 0.0005 # 500 us -> initial delay of 1 ms
while True:
    gotit = waiter.acquire(0)
    if gotit:
        break
    remaining = endtime - time.time()
    if remaining <= 0:
        break
    delay = min(delay * 2, remaining, .05)
    time.sleep(delay)
# Когда переключается GIL?
```

Ожидание множества событий

- Windows имеет `WaitMultipleObjects`
- Другие системы такого не умеют
- `select.select`
- `threading.Barrier`

Атомарные операции

```
f = lambda: [x for x in l]
dis.dis(f)
  2           0 BUILD_LIST               0
              3 LOAD_GLOBAL              0 (l)
              6 GET_ITER
        >>    7 FOR_ITER                12 (to 22)
             10 STORE_FAST               0 (x)
             13 LOAD_FAST                0 (x)
             16 LIST_APPEND              2
             19 JUMP_ABSOLUTE            7
        >>   22 RETURN_VALUE
```

Неатомарно

```
def g(dct, key, default):
    if key in dct:
        return dct[key]
    else:
        dct[key] = default
        return default

dis.dis(g)
```

```
3     0 LOAD_FAST          1 (key)
      3 LOAD_FAST          0 (dct)
      6 COMPARE_OP         6 (in)
      9 POP_JUMP_IF_FALSE 20

4    12 LOAD_FAST          0 (dct)
     15 LOAD_FAST          1 (key)
     18 BINARY_SUBSCR
     19 RETURN_VALUE

6 >> 20 LOAD_FAST          2 (default)
     23 LOAD_FAST          0 (dct)
     26 LOAD_FAST          1 (key)
     29 STORE_SUBSCR

7    30 LOAD_FAST          2 (default)
     33 RETURN_VALUE
```

Атомарно?

```
def f():
    a = 1
    l.append(2)
    a += 3
    r = d.setdefault(
        key, value)
dis.dis(f)
```

```
3   0 LOAD_CONST     1 (1)
    3 STORE_FAST     0 (a)
4   6 LOAD_GLOBAL    0 (l)
    9 LOAD_ATTR      1 (append)
   12 LOAD_CONST     2 (2)
   15 CALL_FUNCTION  1
   18 POP_TOP
5  19 LOAD_FAST      0 (a)
   22 LOAD_CONST     3 (3)
   25 INPLACE_ADD
   26 STORE_FAST     0 (a)
6  29 LOAD_GLOBAL    2 (d)
   32 LOAD_ATTR      3 (setdefault)
   35 LOAD_GLOBAL    4 (key)
   38 LOAD_GLOBAL    5 (value)
   41 CALL_FUNCTION  2
   44 STORE_FAST     1 (r)
```

Кто виноват?

Когда уберут GIL?

Как жить в таких нечеловеческих условиях?

Распределенность

- Многопоточность —> многопроцессность
- Пакет multiprocessing
- Системы для передачи сообщений между потоками: ZeroMQ, AMQP (RabbitMQ и т.д.), Celery, Thift...
- Процессы живут в кластере. *Подсказка: кластер может состоять больше чем из одной машины*

Вопросы?

Андрей Светлов

asvetlov.blogspot.com