SegmentationFault Fest

ALBERTO VARA

DESCARGAR PRESENTACIÓN/CODIGO FUENTE

github.com/avara1986/python-talks/2022_10_pycones

MOTIVACION

MOTIVACION


I COULD RESTRUCTURE THE PROGRAM'S FLOW
OR USE ONE LITTLE 'GOTO' INSTEAD.
EH, SCREW GOOD PRACTICE. HOW BAD CAN IT BE?
goto main_sub3;
COMPILE

ESCENARIO

- Instrumentar código
- Reducir al mínimo el tiempo extra (overhead)
- Python llega a un punto que no es optimo

ESCENARIO

- Instrumentar código
  - "Wrappear" codigo
  - Monkey Patching
- Reducir al mínimo el tiempo extra (overhead)
- Python llega a un punto que no es optimo

func_1 -> 3.06 µs. func_6 -> 5.43 µs

```
def func_1(a, b, c):
    return f"hello {a} {b} {c}"

def func_2(a, b, c):
    return func_1(a, b, c)

def func_3(a, b, c):
    return func_2(a, b, c)

def func_4(a, b, c):
    return func_3(a, b, c)

def func_5(a, b, c):
    return func_4(a, b, c)
```

HERRAMIENTAS/LIBRERÍAS

- Cython
- Pyperf
- Py-spy
- Gdb

EN MATERIA

- Rendimiento
- Segmentation Faults

PERFORMANCE

BÚSQUEDA BINARIA PYTHON

```python
# binarysearch.py
def binary_search(l: List[int], value: int, low: int, high: int):
    if high >= low:
        mid = (high + low) // 2
        if l[mid] == value:
            return mid
        elif l[mid] > value:
            return binary_search(l, value, low, mid - 1)
        else:
            return binary_search(l, value, mid + 1, high)
    else:
        return -1
```

BÚSQUEDA BINARIA PYTHON

```
SEARCH_LIST = [i for i in range(100000)]
```

```
# binarysearch.py
def benchmark_binary_search(loops: int = 10) -> None:
    len_search_list = len(SEARCH_LIST)
    for i in range(0, loops):
        binary_search(SEARCH_LIST, 66666, 0, len_search_list)
```

BÚSQUEDA BINARIA CYTHON

```
# _binarysearch.pyx
cdef int cython_binary_search(l: List[int], value: int, low: int, high
    if high >= low:
        mid = (high + low) // 2
        if l[mid] == value:
            return mid
        elif l[mid] > value:
            return cython_binary_search(l, value, low, mid - 1)
        else:
            return cython_binary_search(l, value, mid + 1, high)
    else:
        return -1
```

BÚSQUEDA BINARIA C++

```
# src/binarysearch.hpp
int binsearch(vector<int> my_list, int value, int low, int high) {
    if (high >= low) {
        int mid = (low + high) / 2;
        if (my_list[mid] == value) {
            return mid;
        } else if (my_list[mid] > value) {
            return binsearch(my_list, value, low, mid - 1);
        } else {
            return binsearch(my_list, value, mid + 1, high);
        }
    } else {
        return -1;
    }
```

BÚSQUEDA BINARIA C++

```
# _binarysearch.pyx
cdef extern from "src/binarysearch.hpp":
    int binsearch(vector[int] l, int value, int low, int high)

cpdef void benchmark_native_binary_search(loops=10):
    cdef int len_search_list = len(SEARCH_LIST)
    for i in range(0, loops):
        binsearch(SEARCH_LIST, 66666, 0, len_search_list)
```

BENCHMARKS CON PYPERF

```
# Inicializamos el runner
runner = pyperf.Runner(add_cmdline_args=add_cmdline_args)
cmd = runner.argparser

# Personalizamos los parámetros del cliente
cmd.add_argument("-f", "--function_name", default="", type=str)
cmd.add_argument("-i", "--inner_loops", default=10, type=int)
parsed_args = runner.parse_args()
function_name = parsed_args.function_name
inner_loops = int(parsed_args.inner_loops)

if function_name == "binary_search":
    runner.bench_func("happy_path_{}".format(inner_loops),
                      benchmark_binary_search, inner_loops)
```

BENCHMARKS CON PYPERF

```
python benchmark.py -f native_binary_search -i 10 -o native_binary_sear
python benchmark.py -f cython_binary_search -i 10 -o cython_binary_sear
python benchmark.py -f binary_search -i 10 -o binary_search_10.json

python -m pyperf compare_to --table binary_search_10.json \
                            cython_binary_search_10.json \
                            native_binary_search_10.json
```

Sin optimización. 10 bucles

Python	Cython	C++
30.9 us	21.2 us: 1.46x faster	42.5 ms: 1372.82x slower

```cpp
#include <chrono>
#include "binarysearch.hpp"


int main(int argc, char *argv[]) {
    vector<int> arr;
    for (int i = 0; i < 1000000; i++) arr.push_back(i);

    std::chrono::steady_clock::time_point begin = std::chrono::steady_c
    for (int i = 0; i < 10; i++) {
        binsearch(arr, 66666, 0, arr.size());
    }
    std::chrono::steady_clock::time_point end = std::chrono::steady_clo
    cout << "Elapsed time: " << std::chrono::duration_cast<std::chrono
```

```
cmake . && make && ./binarysearch
```

```
./binarysearch
Elapsed time: 2.994[ms]
Elapsed time: 2.994.433[μs][microseconds]
Elapsed time: 2.994.433.429[ns][nanoseconds]
```

BÚSQUEDA BINARIA C++

```cpp
int binsearch(vector<int> my_list, int value, int low, int high) {
    if (high >= low) {
        int mid = (low + high) / 2;
        if (my_list[mid] == value) {
            return mid;
        } else if (my_list[mid] > value) {
            return binsearch(my_list, value, low, mid - 1);
        } else {
            return binsearch(my_list, value, mid + 1, high);
        }
    } else {
        return -1;
    }
}
```

BÚSQUEDA BINARIA C++

```cpp
int binsearch(vector<int>& my_list, int value, int low, int high) {
    if (high >= low) {
        int mid = (low + high) / 2;
        if (my_list[mid] == value) {
            return mid;
        } else if (my_list[mid] > value) {
            return binsearch(my_list, value, low, mid - 1);
        } else {
            return binsearch(my_list, value, mid + 1, high);
        }
    } else {
        return -1;
    }
}
```

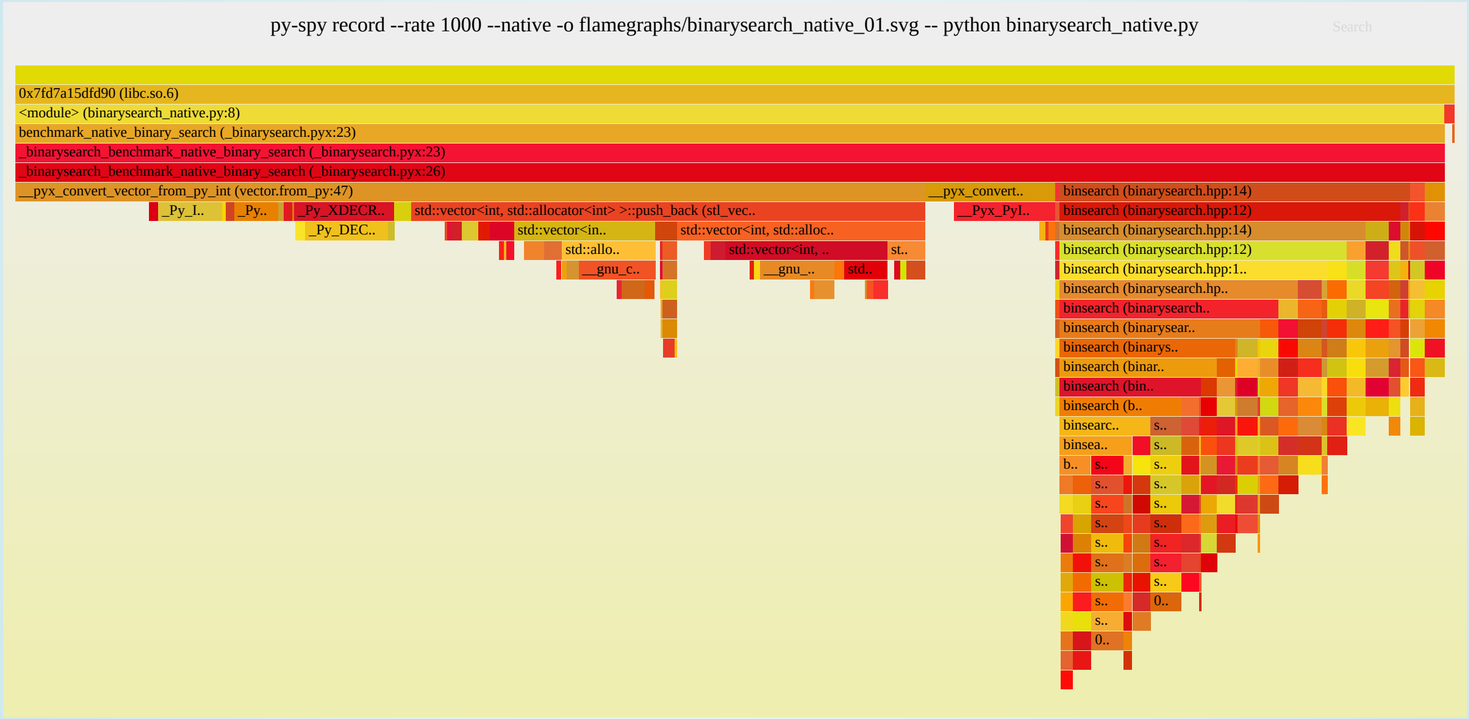
```
cmake . && make && ./binarysearch
```

```
./binarysearch
Elapsed time: 0[ms]
Elapsed time: 2[µs][microseconds]
Elapsed time: 2.827[ns][nanoseconds]
```

Con optimización. 10 bucles

Python	Cython	C++
30.6 us	20.8 us: 1.47x faster	24.2 ms: 792.40x slower

```
py-spy record --rate 1000 --native \
    -o flamegraphs/binarysearch_native_01.svg -- python binarysearch_na
```


py-spy record --rate 1000 --native -o flamegraphs/binarysearch_native_01.svg -- python binarysearch_native.py
Search
0x7fd7a15dfd90 (libc.so.6)
<module> (binarysearch_native.py:8)
benchmark_native_binary_search (_binarysearch.pyx:23)
_binarysearch_benchmark_native_binary_search (_binarysearch.pyx:23)
_binarysearch_benchmark_native_binary_search (_binarysearch.pyx:26)
__pyx_convert_vector_from_py_int (vector.from_py:47)
__pyx_convert..
binsearch (binarysearch.hpp:14)
_Py_I..
_Py..
_Py_XDECR..
std::vector<int, std::allocator<int> >::push_back (stl_vec..
__Pyx_PyI..
binsearch (binarysearch.hpp:12)
_Py_DEC..
std::vector<in..
std::vector<int, std::alloc..
binsearch (binarysearch.hpp:14)
std::allo..
std::vector<int, ..
st..
binsearch (binarysearch.hpp:12)
__gnu_c..
__gnu_..
std..
binsearch (binarysearch.hpp:1..
binsearch (binarysearch.hp..
binsearch (binarysearch..
binsearch (binarysear..
binsearch (binarys..
binsearch (binar..
binsearch (bin..
binsearch (b..
binsearc..
s..
binsea..
b..
0..

```
cpdef void benchmark_native_binary_search(loops=10):
    cdef int len_search_list = len(SEARCH_LIST)
    for i in range(0, loops):
        binsearch(SEARCH_LIST, 66666, 0, len_search_list)
```

```
cpdef void benchmark_native_binary_search(loops=10):
    cdef int len_search_list = len(SEARCH_LIST)
    cdef vector[int] my_list = SEARCH_LIST
    for i in range(0, loops):
        binsearch(&my_list, 66666, 0, len_search_list)
```

BÚSQUEDA BINARIA C++

```cpp
int binsearch(vector<int>& my_list, int value, int low, int high) {
    if (high >= low) {
        int mid = (low + high) / 2;
        if (my_list[mid] == value) {
            return mid;
        } else if (my_list[mid] > value) {
            return binsearch(my_list, value, low, mid - 1);
        } else {
            return binsearch(my_list, value, mid + 1, high);
        }
    } else {
        return -1;
    }
}
```

BÚSQUEDA BINARIA C++

```
int binsearch(vector<int>* my_list, int value, int low, int high) {
    if (high >= low) {
        int mid = (low + high) / 2;
        if ((*my_list)[mid] == value) {
            return mid;
        } else if ((*my_list)[mid] > value) {
            return binsearch(my_list, value, low, mid - 1);
        } else {
            return binsearch(my_list, value, mid + 1, high);
        }
    } else {
        return -1;
    }
}
```

Otimización con punteros. 10 bucles

Python	Cython	C++
31.6 us	21.0 us: 1.51x faster	2.55 ms: 80.80x slower

Y si...

Otimización con punteros. 1.000 bucles

Python	Cython	C++
3.06 ms	1.97 ms: 1.55x faster	2.66 ms: 1.15x faster

Optimización con punteros. 10.000 bucles

Python	Cython	C++
30.2 ms	19.5 ms: 1.55x faster	3.91 ms: 7.73x faster

Optimización con punteros. 100.000 bucles

Python	Cython	C++
304 ms	193 ms: 1.58x faster	14.9 ms: 20.40x faster

SEGMENTATION FAULT (FEST!)

```
strings_and_bytes_dict = {
    "barba": "coa",
    "aaa": b"value_aaa",
    "bbb": b"value_bbb",
    "ccc": "value_ccc",
}
bytes_dict = BytesDict(strings_and_bytes_dict)
print(bytes_dict)
```

```
cdef class BytesDict(object):
    cdef dict bytes_dict

    def __init__(self, dict strings_dict):
        self.bytes_dict = {}
        cdef char * ptr
        cdef char * bytes_key = NULL
        cdef char * bytes_value = NULL

        for k, v in strings_dict.items():
            k = self._string_to_bytes(k, &ptr)
            bytes_key = ptr
            v = self._string_to_bytes(v, &ptr)
            bytes_value = ptr
```

```
cdef object _string_to_bytes(self, object string, char ** ptr):
    if isinstance(string, six.binary_type):
        ptr[0] = PyBytes_AsString(string)
    elif isinstance(string, six.text_type):
        if ptr[0] == NULL:
            string = PyUnicode_AsEncodedString(string, "utf-8", "surrog
            ptr[0] = PyBytes_AsString(string)
    return string
```

```
python setup.py build_ext --inplace
python segmentation.py
[1]    40755 segmentation fault (core dumped)  python segmentation.py
```

```
gdb --args python segmentation.py
Reading symbols from python...
(No debugging symbols found in python)
(gdb)
```

```
(gdb) r
Starting program: /home/alberto.vara/mio/python-talks/2022_10_pycones/v
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so

Program received signal SIGSEGV, Segmentation fault.
0x000000000006b842e in PyBytes_FromString ()
gdb)
```

```
gdb) backtrace
#0  0x00000000006b842e in PyBytes_FromString ()
#1  0x00007ffff7fb4b48 in __pyx_f_13_segmentation_9BytesDict__update_dict (
#2  0x00007ffff7fb4918 in __pyx_pf_13_segmentation_9BytesDict___init__ (__p
#3  0x00007ffff7fb3fa0 in __pyx_pw_13_segmentation_9BytesDict_1__init__ (__
#4  0x00000000005f745f in _PyObject_MakeTpCall ()
#5  0x0000000000570d55 in _PyEval_EvalFrameDefault ()
#6  0x0000000000569dba in _PyEval_EvalCodeWithName ()
#7  0x00000000006902a7 in PyEval_EvalCode ()
#8  0x0000000000067f951 in ?? ()
#9  0x0000000000067f9cf in ?? ()
#10 0x0000000000067fa71 in ?? ()
#11 0x0000000000681b97 in PyRun_SimpleFileExFlags ()
#12 0x000000000006b0d22 in Py_RunMain ()
```

```
gdb) backtrace
#0  0x000000000006b842e in PyBytes_FromString ()
#1  0x00007ffff7fb4b48 in __pyx_f_13_segmentation_9BytesDict__update_dict (
#2  0x00007ffff7fb4918 in __pyx_pf_13_segmentation_9BytesDict___init__ (__p
#3  0x00007ffff7fb3fa0 in __pyx_pw_13_segmentation_9BytesDict_1__init__ (__
#4  0x00000000005f745f in _PyObject_MakeTpCall ()
#5  0x0000000000570d55 in _PyEval_EvalFrameDefault ()
#6  0x0000000000569dba in _PyEval_EvalCodeWithName ()
#7  0x00000000006902a7 in PyEval_EvalCode ()
#8  0x000000000067f951 in ?? ()
#9  0x000000000067f9cf in ?? ()
#10 0x000000000067fa71 in ?? ()
#11 0x0000000000681b97 in PyRun_SimpleFileExFlags ()
#12 0x000000000006b0d33 in Py_RunMain ()
```

```
gdb) backtrace
#0  0x00000000006b842e in PyBytes_FromString ()
#1  0x00007ffff7fb4b48 in __pyx_f_13_segmentation_9BytesDict__update_dict (
#2  0x00007ffff7fb4918 in __pyx_pf_13_segmentation_9BytesDict___init__ (__p
#3  0x00007ffff7fb3fa0 in __pyx_pw_13_segmentation_9BytesDict_1__init__ (__
#4  0x00000000005f745f in _PyObject_MakeTpCall ()
#5  0x0000000000570d55 in _PyEval_EvalFrameDefault ()
#6  0x000000000569dba in _PyEval_EvalCodeWithName ()
#7  0x00000000006902a7 in PyEval_EvalCode ()
#8  0x00000000067f951 in ?? ()
#9  0x00000000067f9cf in ?? ()
#10 0x00000000067fa71 in ?? ()
#11 0x0000000000681b97 in PyRun_SimpleFileExFlags ()
#12 0x000000000006b0d22 in Py_RunMain ()
```

```
for k, v in strings_dict.items():
    k = self._string_to_bytes(k, &ptr)
    bytes_key = ptr
    v = self._string_to_bytes(v, &ptr)
    bytes_value = ptr
    self._update_dict(bytes_key, bytes_value)
```

```
cdef object _string_to_bytes(self, object string, char ** ptr):
    if isinstance(string, six.binary_type):
        ptr[0] = PyBytes_AsString(string)
    elif isinstance(string, six.text_type):
        if ptr[0] == NULL:
            string = PyUnicode_AsEncodedString(string, "utf-8", "surrogatepa
            ptr[0] = PyBytes_AsString(string)
    return string
```

```
cdef object _string_to_bytes(self, object string, char ** ptr):
    ptr[0] = NULL
    if isinstance(string, six.binary_type):
        ptr[0] = PyBytes_AsString(string)
    elif isinstance(string, six.text_type):
        if ptr[0] == NULL:
            string = PyUnicode_AsEncodedString(string, "utf-8", "surrogatepa
            ptr[0] = PyBytes_AsString(string)
    return string
```

```
    def __init__(self, dict strings_dict):
        self.bytes_dict = {}
        cdef char * ptr
        cdef char * bytes_key = NULL
        cdef char * bytes_value = NULL

        for k, v in strings_dict.items():
            k = self._string_to_bytes(k, &ptr)
            bytes_key = ptr
```

```
    def __init__(self, dict strings_dict):
        self.bytes_dict = {}
        cdef char * ptr = NULL
        cdef char * bytes_key = NULL
        cdef char * bytes_value = NULL

        for k, v in strings_dict.items():
            k = self._string_to_bytes(k, &ptr)
            bytes_key = ptr
```

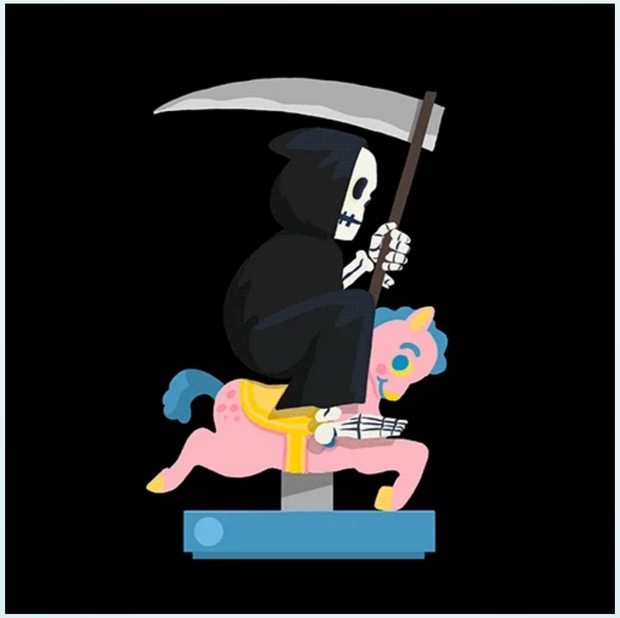
```
python setup.py build_ext --inplace
python segmentation.py
```

```
{b'barba': b'value_aaa', b'value_aaa': b'value_bbb', b'value_bbb': b'va
```

CONCLUSIONES

PASAR VARIABLES ENTRE LENGUAJES TIENE UN PEAJE

P.E: Como la latencia de red entre microservicios



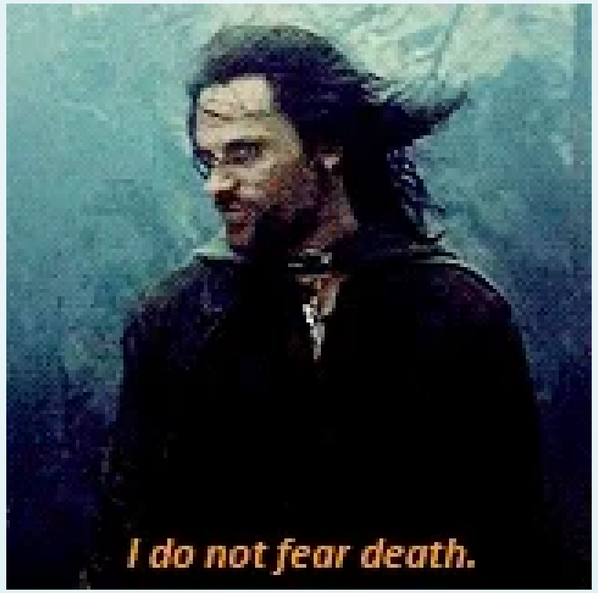
NO TODO SE PUEDE MIGRAR

Para compensar el peaje, el numero de operaciones tiene que ser considerable, lo que dificulta una migración

CAMBIO DE PARADIGMA

IMHO, Cambio de paradigma lenguajes como Python, PHP, JS a compilados C o C++ es mayor

¿HE MENCIONADO EL USO DE MEMORIA EN C++?



¡MUCHAS GRACIAS POR VUESTRO TIEMPO!

We are hiring!
www.datadoghq.com/careers/