Introduction to Analysis of Algorithms

CTypes and Python

Andres Mendez-Vazquez

September 5, 2020

Outline

Outline

We have different ways

First one, the Python API

- A set of functions, macros and variables that provide access to most aspects of the Python run-time system.
    - The Python API is incorporated in a C source file by including the header "Python.h".

Example

```
static PyObject *
spam_system(PyObject *self, PyObject *args) {
    const char *command;
    int sts;
    if (!PyArg_ParseTuple(args, "s", \&command))
        return NULL;
     sts = system(command);
    return PyLong_FromLong(sts);
}
```

We have different ways

First one, the Python API

- A set of functions, macros and variables that provide access to most aspects of the Python run-time system.
  - The Python API is incorporated in a C source file by including the header "Python.h".

Example

```
static PyObject *
spam_system(PyObject *self, PyObject *args) {
    const char *command;
    int sts;
    if (!PyArg_ParseTuple(args, "s", \&command))
        return NULL;
     sts = system(command);
    return PyLong_FromLong(sts);
}
```

Problem

We have a series of macros included at the Python.h

- **PyObject** is an object structure that you use to define object types for Python.
- **PyArg_ParseTuple** parses the arguments you'll receive from your Python program into local variables.
- etc

    - In our case, we need something faster

Problem

We have a series of macros included at the Python.h

- **PyObject** is an object structure that you use to define object types for Python.
- **PyArg_ParseTuple** parses the arguments you'll receive from your Python program into local variables.
- etc

This can be used to build specific libraries totally transparent for Python

- However, this is fine for the final module after development....
  - In out case, we need something faster...

Problem

We have a series of macros included at the Python.h

- **PyObject** is an object structure that you use to define object types for Python.
- **PyArg_ParseTuple** parses the arguments you'll receive from your Python program into local variables.
- etc

This can be used to build specific libraries totally transparent for Python

- However, this is fine for the final module after development....
  - In out case, we need something faster...

Problem

We have a series of macros included at the Python.h

- **PyObject** is an object structure that you use to define object types for Python.
- **PyArg_ParseTuple** parses the arguments you'll receive from your Python program into local variables.
- etc

This can be used to build specific libraries totally transparent for Python

- However, this is fine for the final module after development....
  - In out case, we need something faster...

Problem

We have a series of macros included at the Python.h

- **PyObject** is an object structure that you use to define object types for Python.
- **PyArg_ParseTuple** parses the arguments you'll receive from your Python program into local variables.
- etc

This can be used to build specific libraries totally transparent for Python

- However, this is fine for the final module after development....
  - In out case, we need something faster...

Outline

Cython

They say

- “Cython is an optimising static compiler for both the Python programming language and the extended Cython programming language (based on Pyrex). It makes writing C extensions for Python as easy as Python itself.”

- You have a steep curve of learning
- The footprint of the programs is too large
- Not everything is at the documentation

Cython

They say

- “Cython is an optimising static compiler for both the Python programming language and the extended Cython programming language (based on Pyrex). It makes writing C extensions for Python as easy as Python itself.”

However, it is not that simple

- Cython is hevealy dependent of their implementation on a OS
- You have a steep curve of learning
- The footprint of the programs is too large
- Not everything is at the documentation....

Cython

They say

- "Cython is an optimising static compiler for both the Python programming language and the extended Cython programming language (based on Pyrex). It makes writing C extensions for Python as easy as Python itself."

However, it is not that simple

- Cython is hevealy dependent of their implementation on a OS
- You have a steep curve of learning
- The footprint of the programs is too large
- Not everything is at the documentation....

We want something that allows to pass all the info from one env to another

- So, you only deal with the specific language problems

Cython

They say

- "Cython is an optimising static compiler for both the Python programming language and the extended Cython programming language (based on Pyrex). It makes writing C extensions for Python as easy as Python itself."

However, it is not that simple

- Cython is hevealy dependent of their implementation on a OS
- You have a steep curve of learning
- The footprint of the programs is too large
- Not everything is at the documentation....

We want something that allows to pass all the info from one env to another

- So, you only deal with the specific language problems

Cython

They say

- “Cython is an optimising static compiler for both the Python programming language and the extended Cython programming language (based on Pyrex). It makes writing C extensions for Python as easy as Python itself.”

However, it is not that simple

- Cython is hevealy dependent of their implementation on a OS
- You have a steep curve of learning
- The footprint of the programs is too large
- Not everything is at the documentation....

We want something that allows to pass all the info from one env to another

- So, you only deal with the specific language problems

Cython

They say

- "Cython is an optimising static compiler for both the Python programming language and the extended Cython programming language (based on Pyrex). It makes writing C extensions for Python as easy as Python itself."

However, it is not that simple

- Cython is hevealy dependent of their implementation on a OS
- You have a steep curve of learning
- The footprint of the programs is too large
- Not everything is at the documentation....

We want something that allows to pass all the info from one env to another

- So, you only deal with the specific language problems

Outline

Our Choice

CTypes

- ctypes is a foreign function library for Python.
- It provides C compatible data types, and allows calling functions in DLLs or shared libraries.
- It can be used to wrap these libraries in pure Python.

Our Choice

CTypes

- ctypes is a foreign function library for Python.
- It provides C compatible data types, and allows calling functions in DLLs or shared libraries.
- It can be used to wrap these libraries in pure Python.

Something Notable

- ctypes exports the cdll, and on Windows windll and oledll objects, for loading dynamic link libraries.

Our Choice

CTypes

- ctypes is a foreign function library for Python.
- It provides C compatible data types, and allows calling functions in DLLs or shared libraries.
- It can be used to wrap these libraries in pure Python.

Something Notable

- ctypes exports the cdll, and on Windows windll and oledll objects, for loading dynamic link libraries.

Our Choice

CTypes

- ctypes is a foreign function library for Python.
- It provides C compatible data types, and allows calling functions in DLLs or shared libraries.
- It can be used to wrap these libraries in pure Python.

Something Notable

- ctypes exports the cdll, and on Windows windll and oledll objects, for loading dynamic link libraries.

Structure of the development

First

- Generate your C function

cmult.h

```
#ifndef INSERTIONSORT_FILE
#define INSERTIONSORT_FILE
    /*cmult.h*/
    float cmult(int int_param, float float_param);
#endif
```

Structure of the development

First

- Generate your C function

cmult.h

```
#ifndef INSERTIONSORT_FILE
#define INSERTIONSORT_FILE
    /*cmult.h*/
    float cmult(int int_param, float float_param);
#endif
```

Finally

cmult.c

```
// cmult.c
#include <stdio.h>
#include "cmult.h"
float cmult(int int_param, float float_param) {
    float return_value = int_param * float_param;
    printf(" In cmult : int: %d float %.1f returning %.1f\n", int_param,
float_param, return_value);
    return return_value;
}
```

As you can see

- It is pure C...

Finally

cmult.c

```
// cmult.c
#include <stdio.h>
#include "cmult.h"
float cmult(int int_param, float float_param) {
    float return_value = int_param * float_param;
    printf(" In cmult : int: %d float %.1f returning %.1f\n", int_param,
float_param, return_value);
    return return_value;
}
```

As you can see

- It is pure C...

How do we facilitate the compilation

For this, we can use invoke from python

- Invoke is a Python (2.7 and 3.4+) task execution tool & library,

Let us to take a look at

- task.py

How do we facilitate the compilation

For this, we can use invoke from python

- Invoke is a Python (2.7 and 3.4+) task execution tool & library,

Let us to take a look at

- task.py

gcc

-c

- Compile or assemble the source files, but do not link

-Wall

- This enables all the warnings about constructions that some users consider questionable

-Werror

- Make all warnings into errors.

gcc

-c

- Compile or assemble the source files, but do not link

-Wall

- This enables all the warnings about constructions that some users consider questionable

-Werror

- Make all warnings into errors.

gcc

-c

- Compile or assemble the source files, but do not link

-Wall

- This enables all the warnings about constructions that some users consider questionable

-Werror

- Make all warnings into errors.

Then

-fpic

- Generate position-independent code (PIC) suitable for use in a shared library, if supported for the target machin

-I

- Include the API extensions, in this case, /usr/include/python3.8

Then

-fpic

- Generate position-independent code (PIC) suitable for use in a shared library, if supported for the target machin

-I

- Include the API extensions, in this case, /usr/include/python3.8

Linking the objects being generated

"gcc -shared -o libcmult.so cmult.o"

- Linkage with the libraries is done here...

-shared

- Produce a shared object which can then be linked with other objects to form an executable.

-o

- Place the primary output in file file.
  - In this case so and o
    - A dynamic .so library can be loaded and unloaded at runtime and you have a better flexibility

Linking the objects being generated

"gcc -shared -o libcmult.so cmult.o"

- Linkage with the libraries is done here...

-shared

- Produce a shared object which can then be linked with other objects to form an executable.

-o

- Place the primary output in file file.
  - In this case so and o
    - A dynamic .so library can be loaded and unloaded at runtime and you have a better flexibility

Linking the objects being generated

"gcc -shared -o libcmult.so cmult.o"

- Linkage with the libraries is done here...

-shared

- Produce a shared object which can then be linked with other objects to form an executable.

-o

- Place the primary output in file file.
  - In this case so and o
    - A dynamic .so library can be loaded and unloaded at runtime and you have a better flexibility

Outline

We have to load this libraries

pathlib can be used

- pathlib.Path().absolute() / "libcmult.so"

Then, using ctypes can be used for this

- c_lib = ctypes.CDLL(libname)

Then

- Some extra setup needs to be done

We have to load this libraries

pathlib can be used

- pathlib.Path().absolute() / "libcmult.so"

Then, using ctypes can be used for this

- c_lib = ctypes.CDLL(libname)

Then

- Some extra setup needs to be done

We have to load this libraries

pathlib can be used

- pathlib.Path().absolute() / "libcmult.so"

Then, using ctypes can be used for this

- c_lib = ctypes.CDLL(libname)

Then

- Some extra setup needs to be done

Moving Python Objects to C

First the output needs to be setup even the void one

- c_lib.cmult.restype = ctypes.c_float

The arguments also need to be converted

- answer = c_lib.cmult(x, ctypes.c_float(y))

Moving Python Objects to C

First the output needs to be setup even the void one

- c_lib.cmult.restype = ctypes.c_float

The arguments also need to be converted

- answer = c_lib.cmult(x, ctypes.c_float(y))

Now

We need to have more complex examples

- So we can look at the more interesting problems