# User

I have following dll code:

#define BOOST\_BIND\_GLOBAL\_PLACEHOLDERS
#include <windows.h>
#include <boost/thread.hpp>
#include <boost/bind.hpp>
#include <iostream>
#include <vector>
#include <immintrin.h>

class Worker {
public:
Worker(int numNumbers) : numNumbers(numNumbers) {}

void operator()() {

for(int i = 0; i < numNumbers; ++i) {

unsigned long long randomNumber;

\_rdrand64\_step(&randomNumber);

randomNumber = randomNumber % 900000000000000 + 100000000000000;

boost::lock\_guard<boost::mutex> guard(mutex);

numbers.push\_back(randomNumber);

}

}

std::vector<unsigned long long> getNumbers() {

boost::lock\_guard<boost::mutex> guard(mutex);

return numbers;

}

private:
int numNumbers;
std::vector<unsigned long long> numbers;
boost::mutex mutex;
};

std::vector<unsigned long long> g\_numbers;

extern "C" \_\_declspec(dllexport) void generateRandomNumbersC(int numNumbers, int numWorkers, int numThreads) {
boost::thread\_group threads;
std::vector<Worker\*> workers;

for(int i = 0; i < numWorkers; ++i) {

Worker\* worker = new Worker(numNumbers);

workers.push\_back(worker);

threads.create\_thread(boost::ref(\*worker));

}

threads.join\_all();

for(auto& worker : workers) {

std::vector<unsigned long long> numbers = worker->getNumbers();

g\_numbers.insert(g\_numbers.end(), numbers.begin(), numbers.end());

delete worker;

}

}

extern "C" \_\_declspec(dllexport) unsigned long long\* getNumbersC() {
return g\_numbers.data();
}

extern "C" \_\_declspec(dllexport) int getNumbersSizeC() {
return g\_numbers.size();
}

BOOL APIENTRY DllMain(HMODULE hModule, DWORD ul\_reason\_for\_call, LPVOID lpReserved) {
switch (ul\_reason\_for\_call) {
case DLL\_PROCESS\_ATTACH:
case DLL\_THREAD\_ATTACH:
case DLL\_THREAD\_DETACH:
case DLL\_PROCESS\_DETACH:
break;
}
return TRUE;
}

And following xlwings code:

import xlwings as xw
import sys
import numpy as np
import pandas as pd
from cffi import FFI
import os
import csv
import time

print(sys.executable)
ffi = FFI()
ffi.cdef("""
void generateRandomNumbersC(int numNumbers, int numWorkers, int numThreads);
unsigned long long\* getNumbersC();
int getNumbersSizeC();
""")

C\_boost = ffi.dlopen('D:\OneDrive - 0yt2k\Compiled dlls & executables\boost\_rdrand.dll')
@xw.func
def intel\_rdrand\_boost(NUM\_NUMBERS, NUM\_WORKERS, NUM\_THREADS):
NUM\_NUMBERS = int(NUM\_NUMBERS)
NUM\_WORKERS = int(NUM\_WORKERS)
NUM\_THREADS = int(NUM\_THREADS)
numbers\_ptr = C\_boost.generateRandomNumbers(NUM\_NUMBERS, NUM\_WORKERS, NUM\_THREADS)
numbers = [[int(numbers\_ptr[i])] for i in range(NUM\_NUMBERS)]
return numbers

But I cannot load the dll library in xlwings. Can you give me example python script to test loading of dll in python without xlwings?

# Bing

Sure, you can use the ctypes library in Python to load a DLL. Here's an example:

import ctypes

# Load the DLL

dll = ctypes.CDLL('D:\\OneDrive - 0yt2k\\Compiled dlls & executables\\boost\_rdrand.dll')

# Define the function types

dll.generateRandomNumbersC.argtypes = [ctypes.c\_int, ctypes.c\_int, ctypes.c\_int]

dll.generateRandomNumbersC.restype = None

dll.getNumbersC.argtypes = []

dll.getNumbersC.restype = ctypes.POINTER(ctypes.c\_ulonglong)

dll.getNumbersSizeC.argtypes = []

dll.getNumbersSizeC.restype = ctypes.c\_int

# Call the functions

dll.generateRandomNumbersC(10, 2, 2)

numbers\_ptr = dll.getNumbersC()

numbers\_size = dll.getNumbersSizeC()

# Get the numbers

numbers = [numbers\_ptr[i] for i in range(numbers\_size)]

print(numbers)

This script loads the DLL, defines the argument types and return types of the functions, calls the functions, and prints the generated numbers.

Please replace the DLL path and function arguments with your actual values. Also, make sure that the DLL is compatible with your Python interpreter (i.e., they are both 32-bit or both 64-bit). If they are not compatible, you may encounter problems when loading the DLL.

Remember to run this script with appropriate permissions as it involves file operations. If you still encounter issues, the problem might be with the DLL itself. In that case, you might want to check if the DLL is correctly built and exported.

Please note that this is a simple example and does not include error handling. You might want to add try-except blocks to handle potential errors in a real-world application.

Let me know if this helps or if you have any other questions!

# User

But I need to use cffi.

# Bing

Sure, you can use the cffi library in Python to load a DLL. Here's an example:

from cffi import FFI

# Create FFI object

ffi = FFI()

# Define the C declarations

ffi.cdef("""

void generateRandomNumbersC(int numNumbers, int numWorkers, int numThreads);

unsigned long long\* getNumbersC();

int getNumbersSizeC();

""")

# Load the DLL

dll = ffi.dlopen('D:\\OneDrive - 0yt2k\\Compiled dlls & executables\\boost\_rdrand.dll')

# Call the functions

dll.generateRandomNumbersC(10, 2, 2)

numbers\_ptr = dll.getNumbersC()

numbers\_size = dll.getNumbersSizeC()

# Get the numbers

numbers = [numbers\_ptr[i] for i in range(numbers\_size)]

print(numbers)

This script loads the DLL, defines the C declarations of the functions, calls the functions, and prints the generated numbers.

Please replace the DLL path and function arguments with your actual values. Also, make sure that the DLL is compatible with your Python interpreter (i.e., they are both 32-bit or both 64-bit). If they are not compatible, you may encounter problems when loading the DLL.

Remember to run this script with appropriate permissions as it involves file operations. If you still encounter issues, the problem might be with the DLL itself. In that case, you might want to check if the DLL is correctly built and exported.

Please note that this is a simple example and does not include error handling. You might want to add try-except blocks to handle potential errors in a real-world application.

Let me know if this helps or if you have any other questions!

# User

That test was successful. Now can you extend above and define xlwings UDF with all decorators like @xw.func and others?

# Bing

Sure, here's how you can define a User Defined Function (UDF) in xlwings using the @xw.func decorator:

import xlwings as xw

from cffi import FFI

# Create FFI object

ffi = FFI()

# Define the C declarations

ffi.cdef("""

void generateRandomNumbersC(int numNumbers, int numWorkers, int numThreads);

unsigned long long\* getNumbersC();

int getNumbersSizeC();

""")

# Load the DLL

dll = ffi.dlopen('D:\\OneDrive - 0yt2k\\Compiled dlls & executables\\boost\_rdrand.dll')

@xw.func

def intel\_rdrand\_boost(NUM\_NUMBERS, NUM\_WORKERS, NUM\_THREADS):

# Convert input parameters to integers

NUM\_NUMBERS = int(NUM\_NUMBERS)

NUM\_WORKERS = int(NUM\_WORKERS)

NUM\_THREADS = int(NUM\_THREADS)

# Call the functions

dll.generateRandomNumbersC(NUM\_NUMBERS, NUM\_WORKERS, NUM\_THREADS)

numbers\_ptr = dll.getNumbersC()

numbers\_size = dll.getNumbersSizeC()

# Get the numbers

numbers = [[int(numbers\_ptr[i])] for i in range(numbers\_size)]

return numbers

This script defines a UDF named intel\_rdrand\_boost that can be called from Excel. The function takes three arguments (NUM\_NUMBERS, NUM\_WORKERS, NUM\_THREADS), calls the DLL functions, and returns the generated numbers.

Please replace the DLL path and function arguments with your actual values. Also, make sure that the DLL is compatible with your Python interpreter (i.e., they are both 32-bit or both 64-bit). If they are not compatible, you may encounter problems when loading the DLL.

Remember to run this script with appropriate permissions as it involves file operations. If you still encounter issues, the problem might be with the DLL itself. In that case, you might want to check if the DLL is correctly built and exported.

Please note that this is a simple example and does not include error handling. You might want to add try-except blocks to handle potential errors in a real-world application.

Let me know if this helps or if you have any other questions!

# User

Does xlwings not compatible with cpp? But if python gets it correctly as we tested above, why xlwings throws exception: See below when I load the library (dll):

Here is my xkwings py script code:

import xlwings as xw
from cffi import FFI

# Create FFI object

ffi = FFI()

# Define the C declarations

ffi.cdef("""
void generateRandomNumbersC(int numNumbers, int numWorkers, int numThreads);
unsigned long long\* getNumbersC();
int getNumbersSizeC();
""")

# Load the DLL

dll = ffi.dlopen('D:\Dev\boost\_rdrand.dll')

@xw.func
def intel\_rdrand\_boost(NUM\_NUMBERS, NUM\_WORKERS, NUM\_THREADS):
# Convert input parameters to integers
NUM\_NUMBERS = int(NUM\_NUMBERS)
NUM\_WORKERS = int(NUM\_WORKERS)
NUM\_THREADS = int(NUM\_THREADS)

# Call the functions

dll.generateRandomNumbersC(NUM\_NUMBERS, NUM\_WORKERS, NUM\_THREADS)

numbers\_ptr = dll.getNumbersC()

numbers\_size = dll.getNumbersSizeC()

# Get the numbers

numbers = [[int(numbers\_ptr[i])] for i in range(numbers\_size)]

return numbers

I have ensured that dll exists at the location above.

See errors when I load dll in excel:

pythoncom error: Python error invoking COM method.

Traceback (most recent call last):
File "D:\Programs\Python\Lib\site-packages\win32com\server\policy.py", line 305, in *Invoke*
return self.*invoke*(dispid, lcid, wFlags, args)
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
File "D:\Programs\Python\Lib\site-packages\win32com\server\policy.py", line 310, in *invoke*
return S\_OK, -1, self.*invokeex*(dispid, lcid, wFlags, args, None, None)
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
File "D:\Programs\Python\Lib\site-packages\win32com\server\policy.py", line 639, in *invokeex*
return func(\*args)
^^^^^^^^^^^
File "D:\Programs\Python\Lib\site-packages\xlwings\com\_server.py", line 231, in Call
return ToVariant(getattr(obj, method)(\*pargs, \*\*kwargs))
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
File "D:\Programs\Python\Lib\site-packages\xlwings\udfs.py", line 703, in import\_udfs
module = get\_udf\_module(module\_name, xl\_workbook)
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
File "D:\Programs\Python\Lib\site-packages\xlwings\udfs.py", line 381, in get\_udf\_module
module = import\_module(module\_name)
^^^^^^^^^^^^^^^^^^^^^^^^^^
File "D:\Programs\Python\Lib\importlib\_*init*\_.py", line 90, in import\_module
return \_bootstrap.\_gcd\_import(name[level:], package, level)
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
File "<frozen importlib.\_bootstrap>", line 1387, in \_gcd\_import
File "<frozen importlib.\_bootstrap>", line 1360, in \_find\_and\_load
File "<frozen importlib.\_bootstrap>", line 1331, in \_find\_and\_load\_unlocked
File "<frozen importlib.\_bootstrap>", line 935, in \_load\_unlocked
File "<frozen importlib.\_bootstrap\_external>", line 995, in exec\_module
File "<frozen importlib.\_bootstrap>", line 488, in \_call\_with\_frames\_removed
File "d:\dev\rdrand\_boost.py", line 15, in <module>
dll = ffi.dlopen('D:\Dev\boost\_rdrand.dll')
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
File "D:\Programs\Python\Lib\site-packages\cffi\api.py", line 150, in dlopen
lib, function\_cache = \_make\_ffi\_library(self, name, flags)
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
File "D:\Programs\Python\Lib\site-packages\cffi\api.py", line 832, in \_make\_ffi\_library
backendlib = \_load\_backend\_lib(backend, libname, flags)
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
File "D:\Programs\Python\Lib\site-packages\cffi\api.py", line 828, in \_load\_backend\_lib
return backend.load\_library(path, flags)
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
OSError: cannot load library 'D:\Dev\boost\_rdrand.dll': error 0x7e

# Bing

The error message OSError: cannot load library 'D:\Dev\boost\_rdrand.dll': error 0x7e indicates that the DLL could not be loaded. The error code 0x7e corresponds to ERROR\_MOD\_NOT\_FOUND, which means the specified module could not be found.

This error can occur due to several reasons:

1. **The DLL is not found**: Make sure the DLL is located at the specified path (D:\Dev\boost\_rdrand.dll).
2. **Dependency issues**: The DLL might depend on other DLLs that are not found. You can use tools like Dependency Walker or DumpBin to check the dependencies of the DLL.
3. **Architecture mismatch**: Make sure the architecture (32-bit or 64-bit) of the DLL matches with the Python interpreter you're using.
4. **Running in a different environment**: If you're running the script in a different environment (like Excel via xlwings), it might not have the same PATH as your Python environment. You can try adding the directory of your DLL to the PATH environment variable.

Here's how you can add a directory to the PATH in Python:

import os

os.environ['PATH'] = r'D:\Dev;' + os.environ['PATH']

Add this code before loading the DLL.

Please check these points and let me know if the issue persists. If the problem continues, it would be helpful to have more information about the DLL and the environment you're running the script in. For example, how was the DLL built? Are you able to load other DLLs? What version of Python and xlwings are you using? Are you running the script in Excel or in a Python environment?

# User

Is dependency walker available in msys2 shell. I am using that to compile my C / CPP codes.

# Bing

Dependency Walker is not typically available in the MSYS2 shell. It's a separate utility that you can download and run on Windows to inspect DLLs[**3**](https://stackoverflow.com/questions/60906084/what-is-the-correct-way-to-ship-a-program-built-using-msys2-and-mingw-to-end-use).

However, if you're using MSYS2 and Mingw-w64 for building your applications, you might not need Dependency Walker. MSYS2 provides an environment for POSIX programs and also includes a package manager named pacman that you can use to install lots of other things[**2**](https://stackoverflow.com/questions/61027176/how-can-i-simply-use-pure-mingw-gcc-after-installing-msys2).

When you build an application with MSYS2 and Mingw-w64, the compiled binaries (exe or dll) can be distributed as standalone products. You can do a static build with gcc or clang, producing a single standalone executable, or you can do a build that depends on DLLs which you distribute with your executable[**1**](https://dependencywalker.com/).

If you still want to check the dependencies of your DLL, you could use tools like ldd or objdump that are available in the MSYS2 shell. These tools can provide similar information to Dependency Walker.

For example, to check the dependencies of a DLL with ldd, you can use the following command in the MSYS2 shell:

ldd /path/to/your.dll

This will list the DLLs that your DLL depends on.

Remember, the tools and methods you use can depend on your specific needs and the nature of the DLL you're working with. If you're facing a specific issue, feel free to provide more details so I can give more targeted advice.

Source(s)

1. [Dependency Walker (depends.exe) Home Page](https://dependencywalker.com/)

2. [How can I simply use pure Mingw GCC after installing MSYS(2)?](https://stackoverflow.com/questions/61027176/how-can-i-simply-use-pure-mingw-gcc-after-installing-msys2)

3. [c - What is the correct way to ship a program built using MSYS2 and ...](https://stackoverflow.com/questions/60906084/what-is-the-correct-way-to-ship-a-program-built-using-msys2-and-mingw-to-end-use)

4. [MSYS2 development environment for Windows | David Grayson](https://www.davidegrayson.com/windev/msys2/)

5. [How can I install MinGW-w64 and MSYS2? - Stack Overflow](https://stackoverflow.com/questions/30069830/how-can-i-install-mingw-w64-and-msys2)

# User

See output of my ldd in msys2 shell:

$ ldd boost\_rdrand.dll
ntdll.dll => /c/Windows/SYSTEM32/ntdll.dll (0x7ffc94810000)
KERNEL32.DLL => /c/Windows/System32/KERNEL32.DLL (0x7ffc92880000)
KERNELBASE.dll => /c/Windows/System32/KERNELBASE.dll (0x7ffc921b0000)
msvcrt.dll => /c/Windows/System32/msvcrt.dll (0x7ffc93560000)
libboost\_thread-mt.dll => /mingw64/bin/libboost\_thread-mt.dll (0x7ffc63db0000)
libstdc++-6.dll => /mingw64/bin/libstdc++-6.dll (0x7ffc1dca0000)
libgcc\_s\_seh-1.dll => /mingw64/bin/libgcc\_s\_seh-1.dll (0x7ffc54210000)
libwinpthread-1.dll => /mingw64/bin/libwinpthread-1.dll (0x7ffc84090000)
libwinpthread-1.dll => /mingw64/bin/libwinpthread-1.dll (0x201d34c0000)
CRYPTBASE.DLL => /c/Windows/SYSTEM32/CRYPTBASE.DLL (0x7ffc912f0000)
bcryptPrimitives.dll => /c/Windows/System32/bcryptPrimitives.dll (0x7ffc91e10000)

I think main suspects are: libboost\_thread-mt.dll => /mingw64/bin/libboost\_thread-mt.dll (0x7ffc63db0000)

Or am I wrong. See I also tried another dll which is successfully called from xlwings. See output of ldd for that dll:

$ ldd rdrand\_multithreaded\_new.dll
ntdll.dll => /c/Windows/SYSTEM32/ntdll.dll (0x7ffc94810000)
KERNEL32.DLL => /c/Windows/System32/KERNEL32.DLL (0x7ffc92880000)
KERNELBASE.dll => /c/Windows/System32/KERNELBASE.dll (0x7ffc921b0000)
msvcrt.dll => /c/Windows/System32/msvcrt.dll (0x7ffc93560000)
libwinpthread-1.dll => /mingw64/bin/libwinpthread-1.dll (0x7ffc84090000)
CRYPTBASE.DLL => /c/Windows/SYSTEM32/CRYPTBASE.DLL (0x7ffc912f0000)
bcryptPrimitives.dll => /c/Windows/System32/bcryptPrimitives.dll (0x7ffc91e10000)

Can you locate the bug based on above inputs?

# Bing

Based on the output of ldd, it seems like the boost\_rdrand.dll depends on libboost\_thread-mt.dll, which is not a dependency of the rdrand\_multithreaded\_new.dll that works correctly with xlwings. This could indeed be the source of the issue.

When you're running your Python script through xlwings (and therefore Excel), it might not have the same PATH environment variable as your MSYS2 shell. This means it might not be able to find the libboost\_thread-mt.dll even though it's located in the /mingw64/bin/ directory.

You can try adding the directory containing libboost\_thread-mt.dll to the PATH in your Python script before loading the DLL. Here's how you can do it:

import os

# Add the directory to PATH

os.environ['PATH'] = r'/mingw64/bin;' + os.environ['PATH']

# Now load the DLL

dll = ffi.dlopen('D:\\Dev\\boost\_rdrand.dll')

This will make libboost\_thread-mt.dll available to your Python script when it's run through xlwings and Excel.

Please give this a try and let me know if it helps! If the issue persists, there might be other factors at play, and we'll need to investigate further. For example, there could be issues with the way the DLL was built, or there could be other missing dependencies.

# User

Can I use like this:

os.environ['PATH'] = r'D:\Programs\Msys2\mingw64\bin;' + os.environ['PATH']

Please correct if there is error. Above is actual path of that boost dll.

# Bing

Yes, your code is correct. You're adding the directory D:\\Programs\\Msys2\\mingw64\\bin to the PATH environment variable in Python. This directory will be searched when loading DLLs. Here's your code:

import os

# Add the directory to PATH

os.environ['PATH'] = r'D:\\Programs\\Msys2\\mingw64\\bin;' + os.environ['PATH']

# Now load the DLL

dll = ffi.dlopen('D:\\Dev\\boost\_rdrand.dll')

This should help Python find the libboost\_thread-mt.dll when you're running your script through xlwings and Excel. If the issue persists, please let me know! We can continue to troubleshoot the problem. Remember to replace the paths with your actual paths. Also, ensure that the Python interpreter you're using (through xlwings) has the necessary permissions to access these directories and files.

# User

So after that it worked!

# Bing

That's great to hear! I'm glad the solution worked for you. If you have any more questions or need further assistance, feel free to ask. Happy coding! 😊