# User

I have following source cpp code that compiles into dll library:

#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;
}

Further I have this test cpp code which calls dll function from above dll

#include <windows.h>
#include <iostream>
#include <vector>

typedef void (*GenerateRandomNumbersFuncC)(int, int, int);
typedef unsigned long long* (\*GetNumbersFuncC)();
typedef int (\*GetNumbersSizeFuncC)();

int main(int argc, char\* argv[]) {
HINSTANCE hGetProcIDDLL = LoadLibrary("boost\_rdrand\_ucrt.dll");

if (!hGetProcIDDLL) {

std::cout << "Could not load the dynamic library." << std::endl;

return EXIT\_FAILURE;

}

GenerateRandomNumbersFuncC generateRandomNumbersC = (GenerateRandomNumbersFuncC) GetProcAddress(hGetProcIDDLL, "generateRandomNumbersC");

GetNumbersFuncC getNumbersC = (GetNumbersFuncC) GetProcAddress(hGetProcIDDLL, "getNumbersC");

GetNumbersSizeFuncC getNumbersSizeC = (GetNumbersSizeFuncC) GetProcAddress(hGetProcIDDLL, "getNumbersSizeC");

if (!generateRandomNumbersC || !getNumbersC || !getNumbersSizeC) {

std::cout << "Could not locate the function." << std::endl;

return EXIT\_FAILURE;

}

int numNumbers = std::stoi(argv[1]);

int numWorkers = std::stoi(argv[2]);

int numThreads = std::stoi(argv[3]);

generateRandomNumbersC(numNumbers, numWorkers, numThreads);

unsigned long long\* numbers = getNumbersC();

int numbersSize = getNumbersSizeC();

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

std::cout << numbers[i] << std::endl;

}

FreeLibrary(hGetProcIDDLL);

return EXIT\_SUCCESS;

}

I could compile both of them successfully. But When I run the testcpp code like this in msys2 shell:

./testcpp\_ucrt 10 10 4 (the last three are parameters - 10 random numbers, 10 workers and 4 threads)

But the code returns 100 random numbers not 10 why?

# Bing

The reason you're seeing 100 random numbers instead of 10 is due to the way your code is structured. In your generateRandomNumbersC function, you're creating numWorkers number of Worker objects, and each Worker is generating numNumbers random numbers.

Here's the relevant part of your code:

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

Worker\* worker = new Worker(numNumbers);

workers.push\_back(worker);

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

}

In this loop, you're creating numWorkers Worker objects, and each Worker is initialized with numNumbers. This means each Worker will generate numNumbers random numbers.

So, if you run your program with the parameters 10 10 4, it means you're creating 10 workers, and each worker is generating 10 random numbers. Hence, you're getting 10 \* 10 = 100 random numbers in total.

If you want each worker to generate a portion of the total numbers such that the total number of generated numbers is 10, you should divide numNumbers by numWorkers when creating the Worker objects. Please note that this will work correctly when numNumbers is a multiple of numWorkers. If it's not, you'll need to handle the remainder. For example, if you want to generate 10 numbers with 3 workers, two workers can generate 3 numbers and one worker can generate 4 numbers. Or, you could have each worker generate 3 numbers and ignore the extra one. The choice depends on your specific requirements.

# User

But how to handle this in code:

# Bing

You can handle this in your code by dividing the total number of numbers (numNumbers) by the number of workers (numWorkers). Here's how you can modify your generateRandomNumbersC function:

extern "C" \_\_declspec(dllexport) void generateRandomNumbersC(int numNumbers, int numWorkers, int numThreads) {

boost::thread\_group threads;

std::vector<Worker\*> workers;

// Calculate the number of numbers each worker should generate

int numNumbersPerWorker = numNumbers / numWorkers;

int remainder = numNumbers % numWorkers;

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

// If there's a remainder, add one to the number of numbers for this worker

int numNumbersThisWorker = numNumbersPerWorker + (i < remainder ? 1 : 0);

Worker\* worker = new Worker(numNumbersThisWorker);

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;

}

}

In this code, numNumbersPerWorker is the number of numbers each worker should generate. The remainder is the number of extra numbers when numNumbers is not a multiple of numWorkers. For the first remainder workers, they generate one more number than numNumbersPerWorker. The rest of the workers generate numNumbersPerWorker numbers. This way, the total number of generated numbers is numNumbers. Please note that this code assumes numWorkers is not zero. If numWorkers can be zero, you should add code to handle this case.

I hope this helps! If you have any other questions, feel free to ask. 😊

# User

But how to handle this in code:

# Bing

You can handle this in your code by dividing the total number of numbers (numNumbers) by the number of workers (numWorkers). Here's how you can modify your generateRandomNumbersC function:

extern "C" \_\_declspec(dllexport) void generateRandomNumbersC(int numNumbers, int numWorkers, int numThreads) {

boost::thread\_group threads;

std::vector<Worker\*> workers;

// Calculate the number of numbers each worker should generate

int numNumbersPerWorker = numNumbers / numWorkers;

int remainder = numNumbers % numWorkers;

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

// If there's a remainder, add one to the number of numbers for this worker

int numNumbersThisWorker = numNumbersPerWorker + (i < remainder ? 1 : 0);

Worker\* worker = new Worker(numNumbersThisWorker);

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;

}

}

In this code, numNumbersPerWorker is the number of numbers each worker should generate. The remainder is the number of extra numbers when numNumbers is not a multiple of numWorkers. For the first remainder workers, they generate one more number than numNumbersPerWorker. The rest of the workers generate numNumbersPerWorker numbers. This way, the total number of generated numbers is numNumbers. Please note that this code assumes numWorkers is not zero. If numWorkers can be zero, you should add code to handle this case.

I hope this helps! If you have any other questions, feel free to ask. 😊

# User

Is this code true only if reminder is 1 or for any reminder? How to handle reminders instead of hard coding them as above? Can you change?

# Bing

The code I provided earlier works for any remainder, not just 1. The line int numNumbersThisWorker = numNumbersPerWorker + (i < remainder ? 1 : 0); is not hardcoding the remainder to be 1. It's adding 1 to numNumbersPerWorker for the first remainder workers.

Here's how it works:

* i < remainder ? 1 : 0 is a conditional expression that evaluates to 1 if i is less than remainder and 0 otherwise.
* For the first remainder workers (where i is less than remainder), this expression is 1, so one more number is generated by these workers.
* For the rest of the workers (where i is not less than remainder), this expression is 0, so numNumbersPerWorker numbers are generated by these workers.

This way, the total number of generated numbers is numNumbers, and the remainder is distributed evenly among the first remainder workers.

So, the code already handles remainders correctly for any value of remainder, and there's no need to change it. If you have any other questions, feel free to ask. 😊

# User

There seems to be some bug in following xlwings script which calls above dll function. I tried function in excel cell A1 like =intel\_rdrand\_boost(100,10,4). When I refresh of edit via F2, iot goes on adding 100 rnadom numbers each time. Besides. first time, it generated 110 instead of 100 random numbers. IS there some bug in dll source you provided above or xlwings script which I reproduce below:

import xlwings as xw
from cffi import FFI
import os

os.environ['PATH'] = r'D:\Programs\Msys2\ucrt64\bin;' + os.environ['PATH']

# 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:\Programs\Msys2\home\juhi123\Downloads\boost\_rdrand\_ucrt.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

# Bing

The issue you're experiencing is due to the global variable g\_numbers in your DLL code. Each time you call the generateRandomNumbersC function, it appends new numbers to g\_numbers without clearing the old numbers. This is why the number of random numbers keeps increasing each time you refresh or edit the cell in Excel.

To fix this issue, you need to clear g\_numbers at the beginning of the generateRandomNumbersC function. Here's how you can modify your DLL code:

extern "C" \_\_declspec(dllexport) void generateRandomNumbersC(int numNumbers, int numWorkers, int numThreads) {

// Clear the global numbers vector

g\_numbers.clear();

boost::thread\_group threads;

std::vector<Worker\*> workers;

// Calculate the number of numbers each worker should generate

int numNumbersPerWorker = numNumbers / numWorkers;

int remainder = numNumbers % numWorkers;

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

// If there's a remainder, add one to the number of numbers for this worker

int numNumbersThisWorker = numNumbersPerWorker + (i < remainder ? 1 : 0);

Worker\* worker = new Worker(numNumbersThisWorker);

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;

}

}

This modification ensures that g\_numbers is cleared each time generateRandomNumbersC is called, so it only contains the numbers generated in the current call.

I hope this helps! If you have any other questions, feel free to ask. 😊