# User

Can you convert following C code (for compiling to shared dll library on windows ) to rust (shared dll library on windows) and give me rust command line to do so?

#include <pthread.h>
#include <immintrin.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>

unsigned long long \*numbers;
unsigned long long N = 999999999999999; // Change this to your desired maximum number

\_\_declspec(dllexport) int rdrand64\_step(unsigned long long \*rand)
{
return \_rdrand64\_step(rand);
}

\_\_declspec(dllexport) void generate\_random\_numbers(int num\_threads, int num\_numbers)
{
unsigned long long rand;
for (int thread\_num = 0; thread\_num < num\_threads; thread\_num++) {
for (int i = 0; i < num\_numbers / num\_threads; i++) {
do {
if (!rdrand64\_step(&rand)) {
printf("Failed to generate random number.\n");
}
rand = rand % (N + 1);
} while (rand < 100000000000000); // Ensure the number is 15 digits long
numbers[thread\_num \* num\_numbers / num\_threads + i] = rand;
}
}
}

\_\_declspec(dllexport) unsigned long long\* get\_numbers()
{
return numbers;
}

\_\_declspec(dllexport) void allocate\_memory(int num\_numbers)
{
numbers = (unsigned long long\*)malloc(num\_numbers \* sizeof(unsigned long long));
}

\_\_declspec(dllexport) void free\_memory()
{
free(numbers);
}

# Bing

Here's the equivalent Rust code for your C code. Please note that Rust doesn't have a direct equivalent for the \_rdrand64\_step function, so I've used the rand::Rng trait from the rand crate to generate random numbers.

use std::alloc::{alloc, dealloc, Layout};

use std::os::raw::c\_void;

use rand::Rng;

const N: u64 = 999999999999999; // Change this to your desired maximum number

#[no\_mangle]

pub extern "C" fn rdrand64\_step(rand: &mut u64) -> i32 {

let mut rng = rand::thread\_rng();

\*rand = rng.gen\_range(0..=N);

return 1;

}

#[no\_mangle]

pub extern "C" fn generate\_random\_numbers(num\_threads: i32, num\_numbers: i32) {

unsafe {

let rand: u64 = 0;

for thread\_num in 0..num\_threads {

for i in 0..num\_numbers / num\_threads {

loop {

if rdrand64\_step(&rand) == 0 {

println!("Failed to generate random number.");

}

if rand >= 100000000000000 {

break;

}

}

\*NUMBERS.offset((thread\_num \* num\_numbers / num\_threads + i) as isize) = rand;

}

}

}

}

static mut NUMBERS: \*mut u64 = std::ptr::null\_mut();

#[no\_mangle]

pub extern "C" fn get\_numbers() -> \*mut u64 {

unsafe { NUMBERS }

}

#[no\_mangle]

pub extern "C" fn allocate\_memory(num\_numbers: i32) {

unsafe {

let layout = Layout::from\_size\_align(num\_numbers as usize \* std::mem::size\_of::<u64>(), std::mem::align\_of::<u64>()).unwrap();

NUMBERS = alloc(layout) as \*mut u64;

}

}

#[no\_mangle]

pub extern "C" fn free\_memory() {

unsafe {

let layout = Layout::from\_size\_align(std::mem::size\_of\_val(&\*NUMBERS), std::mem::align\_of\_val(&\*NUMBERS)).unwrap();

dealloc(NUMBERS as \*mut u8, layout);

}

}

To compile this Rust code into a DLL, you can use the following command:

rustc --crate-type cdylib your\_file.rs

Please replace your\_file.rs with the name of your Rust source file. This will produce a .dll file in the same directory.

Please note that you'll need to add the rand crate to your Cargo.toml file:

[dependencies]

rand = "0.8"

You can then use cargo build --release to build your project. The resulting .dll file will be in the target/release directory.

Please ensure that you have the rand crate installed in your Rust environment. If not, you can add it by adding rand = "0.8" to your Cargo.toml file under [dependencies] and running cargo build. This will download and compile the rand crate and its dependencies.

# User

But where is Cargo.toml file? I find following folders and file under my rustup folder on D drive?

bin, downloads, tmp, files toolchains, update-hashes, settings.toml (file)

# Bing

The Cargo.toml file is not located in your rustup folder. It's actually located in the root directory of your Rust project. When you create a new Rust project using the cargo new command, a new directory is created with the name of your project, and inside this directory, the Cargo.toml file is automatically generated.

Here's how you can create a new Rust project:

1. Open a terminal.
2. Navigate to the directory where you want to create your new Rust project.
3. Run the command cargo new my\_project, replacing my\_project with the name you want to give to your project.
4. Navigate into your new project directory with cd my\_project.

Now, you should see the Cargo.toml file in this my\_project directory.

If you're writing a standalone Rust file without creating a new project, you won't have a Cargo.toml file. In this case, you can manually create a Cargo.toml file in the same directory as your Rust file and add the necessary dependencies there. However, it's recommended to use cargo new to create a new project because it sets up the correct directory structure and files for a Rust project.

Remember to add the rand crate to your Cargo.toml file under [dependencies] like this:

[dependencies]

rand = "0.8"

Then, you can use cargo build to build your project. This will download and compile the rand crate and its dependencies. The resulting .dll file will be in the target/debug directory for debug builds, or target/release for release builds (if you run cargo build --release).

Please let me know if you need further assistance! 😊

# User

When I tried to compile your given code in rust, I got compilation errors as:

error[E0432]: unresolved import rand
--> rust\_rand\_dll.rs:3:5
|
3 | use rand::Rng;
| ^^^^ maybe a missing crate rand?
|
= help: consider adding extern crate rand to use the rand crate

warning: unused import: std::os::raw::c\_void
--> rust\_rand\_dll.rs:2:5
|
2 | use std::os::raw::c\_void;
| ^^^^^^^^^^^^^^^^^^^^
|
= note: #[warn(unused\_imports)] on by default

error[E0433]: failed to resolve: use of undeclared crate or module rand
--> rust\_rand\_dll.rs:9:19
|
9 | let mut rng = rand::thread\_rng();
| ^^^^ use of undeclared crate or module rand

error[E0308]: mismatched types
--> rust\_rand\_dll.rs:21:38
|
21 | if rdrand64\_step(&rand) == 0 {
| ------------- ^^^^^ types differ in mutability
| |
| arguments to this function are incorrect
|
= note: expected mutable reference &mut \_
found reference &\_
note: function defined here
--> rust\_rand\_dll.rs:8:19
|
8 | pub extern "C" fn rdrand64\_step(rand: &mut u64) -> i32 {
| ^^^^^^^^^^^^^ --------------

error: aborting due to 3 previous errors; 1 warning emitted

# Bing

I apologize for the confusion. Let's address these errors one by one:

1. **Unresolved import `rand`**: This error occurs because the rand crate is not found in your project. You need to add it to your Cargo.toml file under [dependencies] like this:

[dependencies]

rand = "0.8"

Then run cargo build to download and compile the rand crate and its dependencies.

1. **Unused import `std::os::raw::c\_void`**: This warning is because c\_void is imported but not used in the code. You can safely remove the line use std::os::raw::c\_void; from your code.
2. **Failed to resolve: use of undeclared crate or module `rand`**: This error is similar to the first one. It's because the rand crate is not found. The solution is the same: add rand to your Cargo.toml file and run cargo build.
3. **Mismatched types**: This error is because the rand variable is not mutable, but the rdrand64\_step function expects a mutable reference. You should declare rand as mutable like this: let mut rand: u64 = 0;.

Here's the corrected code:

use std::alloc::{alloc, dealloc, Layout};

use rand::Rng;

const N: u64 = 999999999999999; // Change this to your desired maximum number

#[no\_mangle]

pub extern "C" fn rdrand64\_step(rand: &mut u64) -> i32 {

let mut rng = rand::thread\_rng();

\*rand = rng.gen\_range(0..=N);

return 1;

}

#[no\_mangle]

pub extern "C" fn generate\_random\_numbers(num\_threads: i32, num\_numbers: i32) {

unsafe {

let mut rand: u64 = 0;

for thread\_num in 0..num\_threads {

for i in 0..num\_numbers / num\_threads {

loop {

if rdrand64\_step(&mut rand) == 0 {

println!("Failed to generate random number.");

}

if rand >= 100000000000000 {

break;

}

}

\*NUMBERS.offset((thread\_num \* num\_numbers / num\_threads + i) as isize) = rand;

}

}

}

}

static mut NUMBERS: \*mut u64 = std::ptr::null\_mut();

#[no\_mangle]

pub extern "C" fn get\_numbers() -> \*mut u64 {

unsafe { NUMBERS }

}

#[no\_mangle]

pub extern "C" fn allocate\_memory(num\_numbers: i32) {

unsafe {

let layout = Layout::from\_size\_align(num\_numbers as usize \* std::mem::size\_of::<u64>(), std::mem::align\_of::<u64>()).unwrap();

NUMBERS = alloc(layout) as \*mut u64;

}

}

#[no\_mangle]

pub extern "C" fn free\_memory() {

unsafe {

let layout = Layout::from\_size\_align(std::mem::size\_of\_val(&\*NUMBERS), std::mem::align\_of\_val(&\*NUMBERS)).unwrap();

dealloc(NUMBERS as \*mut u8, layout);

}

}

I hope this helps! Let me know if you have any other questions. 😊

# User

Whether I have to copy .rs file to the newly created cargo project folder?

# Bing

Yes, you should place your .rs file in the src directory of your newly created Cargo project. Here's the typical structure of a Cargo project:

my\_project

├── Cargo.toml

└── src

└── main.rs

In your case, you can replace main.rs with your .rs file or you can add your .rs file to the src directory and then import it in main.rs. After you've added your .rs file, you can run cargo build to compile your project. 😊

# User

Following compilation error:

PS D:\Downloads\rust\_rand\_dll> cargo build
Compiling rust\_rand\_dll v0.1.0 (D:\Downloads\rust\_rand\_dll)
error[E0601]: main function not found in crate rust\_rand\_dll
--> src\main.rs:54:2
|
54 | }
| ^ consider adding a main function to src\main.rs

For more information about this error, try rustc --explain E0601.
error: could not compile rust\_rand\_dll (bin "rust\_rand\_dll") due to 1 previous error

# Bing

The error message is indicating that a main function was not found in your crate. This is because by default, Cargo expects to build a binary crate, which requires a main function as the entry point of the program.

However, since you're trying to build a dynamic library (.dll), you should tell Cargo to build a library instead. You can do this by adding the following lines to your Cargo.toml file:

[lib]

name = "rust\_rand\_dll"

crate-type = ["cdylib"]

Here, name should be the name of your library, and crate-type should be ["cdylib"] to indicate that you want to compile a C-compatible dynamic library.

After adding these lines, you can run cargo build again. This time, Cargo will build a dynamic library instead of a binary, so it won't look for a main function.

Please let me know if you have any other questions! 😊

# User

Now following error compilation:

error: failed to parse manifest at D:\Downloads\rust\_rand\_dll\Cargo.toml

Caused by:
can't find library rust\_rand\_dll, rename file to src/lib.rs or specify lib.path

# Bing

I apologize for the confusion. The error message is indicating that Cargo is looking for a source file named lib.rs in the src directory, but it's not finding it. This is because by default, Cargo expects the source file for a library to be named lib.rs.

You have two options to resolve this:

1. **Rename your `.rs` file to `lib.rs`**: If your Rust source file is not named lib.rs, you can rename it to lib.rs and place it in the src directory.
2. **Specify the path in `Cargo.toml`**: If you want to keep the current name of your .rs file, you can specify the path to your source file in the Cargo.toml file. Here's how you can do it:

[lib]

name = "rust\_rand\_dll"

path = "src/your\_file.rs"

crate-type = ["cdylib"]

Replace your\_file.rs with the name of your Rust source file.

After making these changes, you should be able to run cargo build without any errors. Please let me know if you have any other questions! 😊

# User

See the following outputs of cargo build. I can't seem to find dll.

PS D:\Downloads\rust\_rand\_dll> cargo build
Compiling rust\_rand\_dll v0.1.0 (D:\Downloads\rust\_rand\_dll)
Finished dev [unoptimized + debuginfo] target(s) in 0.87s

# Bing

The cargo build command compiles your code and places the output in the target/debug directory for debug builds. Since you're building a dynamic library (.dll), the output file will have a .dll extension.

You can find the .dll file in the target/debug directory of your project. The file will be named after your project, so in your case, it should be rust\_rand\_dll.dll.

Here's how you can navigate to it:

1. Open a terminal.
2. Navigate to your project directory: cd D:\Downloads\rust\_rand\_dll
3. Navigate to the target/debug directory: cd target\debug

Now, you should see your rust\_rand\_dll.dll file in this directory.

If you want to build a release version of your dynamic library, you can run cargo build --release. The output will then be placed in the target/release directory and optimizations will be applied to your code, making it run faster.

Please let me know if you have any other questions! 😊

# User

The dll file size is very big as compared to dll compiled using gcc. Can't it be reduced?

# Bing

Yes, the size of the DLL file generated by Rust can be larger than one compiled with GCC for a few reasons, including the fact that Rust includes more debugging information by default and statically links to its standard library. However, there are a few strategies you can use to reduce the size of your Rust DLL file:

1. **Build in release mode**: By default, cargo build compiles your code in debug mode, which includes additional debugging information that can increase the size of the resulting binary. You can build in release mode with the --release flag, like so: cargo build --release. This will apply optimizations and remove debugging information, resulting in a smaller binary.
2. **Strip the binary**: Stripping a binary removes all symbol table and relocation information from the binary. After building your project, you can use the strip command to reduce the size of the binary. Here's how you can do it:

strip target/release/your\_binary.dll

1. **Use the `opt-level = "z"` or `opt-level = "s"` option**: These options optimize for size. The "z" option optimizes for size, but prioritizes reducing size over improving speed even more than the "s" option. You can add these options under the [profile.release] section in your Cargo.toml file, like so:

[profile.release]

opt-level = "z"

or

[profile.release]

opt-level = "s"

Please note that these methods may make your program run slower, as they remove debugging information and apply optimizations that prioritize size over speed. Always test your program thoroughly after applying these methods to ensure it still behaves as expected. 😊