Rust 编程

1. safe/unsafe

1.1 unsafe rust是 safe rust的一种超集。在unsafe rust中并不会禁用safe rust的常规的安全检查。

1.2 Unsafe rust只是在进行如下几种操作时，不会提供安全检查：

1. 解引用裸指针
2. 调用unsafe的函数或方法
3. 访问或修改可变静态变量
4. 实现unsafe trait
5. 读写union中的字段
   1. Unsafe rust 和 safe rust 的区分带来以下三方面的结果
6. unsafe rust由于不需要部分安全检查，性能存在一定的提升。
7. unsafe rust 内存安全完全交由开发者来保证。
8. 区分编译器和开发者的职责，如果代码出现问题，可以优先排查unsafe部分代码。

1.4

|  |  |
| --- | --- |
|  |  |
| safe function | 在任何上下文或任何输入条件下都不会触发未定义行为（UB） |
| Unsafe function | 要达到safe状态，有预设条件 |

safe code ( safe function);

unsafe {

unsafe code (

safe function;

unsafe function;

);

}

safe code(safe function);

* 1. Safe function/unsafe function都可以嵌套unsafe block

safe code (

safe function(

unsafe { … }

);

);

unsafe {

unsafe code (

safe function (unsafe { … } );

unsafe function(unsafe { … } );

);

}

safe code(safe function);

* 1. 一些误解

Safe 代表不会出现UB。

Unsafe 代表必然会出现UB。

* 1. 被unsafe标记的函数或方法只能在unsafe块中调用

fn main() {

let x = vec![108, 108];

let x = unsafe {

String::from\_utf8\_unchecked(x);

}

}

pub unsafe fn from\_utf8\_unchecked(…)

pub unsafe trait XX {. }

* 1. 访问和修改可变静态变量必须在unsafe块中

static mut TEST: u64 = 100;

fn main() {

let x: u64 = 3;

unsafe {

Let y = TEST + 1;

TEST = y;

}

}

* 1. 原生指针

\*const T 和 \*mut T

let addr = 0xffff7400；

let r = addr as \*const String;

unsafe {

let x = r ////

}

1. 任务

分析代码：

<https://os.phil-opp.com/zh-CN/minimal-rust-kernel/>

并用qemu模拟运行.

输出：关键代码分析+运行结果