



# 课程实验报告

## 0x00 环境搭建与实验准备

课程名称	操作系统原理实验
专业名称	计算机科学与技术
学生姓名	陈政宇
学生学号	23336003
实验地点	东校园-实验中心大楼 B201
实验成绩	
实验日期	2025 年 3 月 6 日

# 目录

<b>1 Rust 编程实践 .....</b>	<b>3</b>
1.1 任务一：文件读取 .....	3
1.2 任务二：代码单元测试 .....	5
1.3 任务三：彩色文字终端 .....	6
1.4 任务四：枚举类型 .....	6
1.5 任务五：元组结构体 .....	6
<b>2 YSOS，启动! .....</b>	<b>6</b>
2.1 配置 Rust Toolchain .....	6
2.2 年轻人的第一个 UEFI 程序 .....	6
<b>3 思考题 .....</b>	<b>7</b>
3.1 现代操作系统的启动流程 .....	7
3.2 Makefile 过程 .....	7
3.3 Cargo 包管理工具 .....	7
3.4 #[entry] 与 main .....	7
<b>4 附加题 .....</b>	<b>7</b>
4.1 彩色的日志 .....	7
4.2 Rust 实现的简单 shell .....	7
4.3 线程模型 .....	7

## 图表

<b>图 1.1.1 任务一运行结果 .....</b>	<b>5</b>
<b>图 1.1.2 单元测试结果 .....</b>	<b>5</b>

# 1 Rust 编程实践

## 1.1 任务一：文件读取

### count\_down(seconds: u64)

该函数接收一个 u64 类型的参数，表示倒计时的秒数。函数应该每秒输出剩余的秒数，直到倒计时结束，然后输出 Countdown finished!。

利用简单的 for 循环和线程休眠实现倒计时功能，代码如下：

```
1 fn count_down(seconds: u64) {
2     for i in (1..=seconds).rev() {
3         println!("{}", seconds left...", i);
4         std::io::stdout().flush().unwrap();
5         std::thread::sleep(std::time::Duration::from_secs(1));
6     }
7     info!("Count down finished!");
8 }
```

Rust

### read\_and\_print

该函数接收一个字符串参数，表示文件的路径。函数应该尝试读取并输出文件的内容。如果文件不存在，函数应该使用 expect 方法主动 panic，并输出 File not found!。

采用了 std::fs::File::open 方法打开文件，然后使用 std::io::BufReader 读取文件内容。用 error! 宏记录错误日志，并使用 ? 运算符将错误向上传播，代码如下：

```
1 fn read_and_print(file_path: &str) -> Result<(), std::io::Error> {
2     let file = std::fs::File::open(file_path).map_err(|e| {
3         error!("Failed to open file {}: {}", file_path, e);
4         e
5     })?;
6     let reader = std::io::BufReader::new(file);
7     for line in reader.lines() {
```

Rust

```

8         println!("{}", line?);
9     }
10    info!("File read successfully!");
11    Ok(())
12 }

```

### ☰ file\_size

该函数接收一个字符串参数，表示文件的路径，并返回一个 `Result`。函数应该尝试打开文件，并在 `Result` 中返回文件大小。如果文件不存在，函数应该返回一个包含 `File not found!` 字符串的 `Err`。

实现方式与 `read_and_print` 基本一致，代码如下：

```

1  fn file_size(file_path: &str) -> Result<u64, std::io::Error> { Rust
2      let file = std::fs::File::open(file_path).map_err(|e| {
3          error!("Failed to open file {}: {}", file_path, e);
4          std::io::Error::new(std::io::ErrorKind::NotFound, "File not
5              found!")
6      })?;
7      let mut reader = std::io::BufReader::new(file);
8      let mut buffer = Vec::new();
9      reader.read_to_end(&mut buffer)?;
10     Ok(buffer.len() as u64)
11 }

```

### ☰ Final

在 `main` 函数中，按照如下顺序调用上述函数：

1. 首先调用 `count_down(5)` 函数进行倒计时
2. 然后调用 `read_and_print("/etc/hosts")` 函数尝试读取并输出文件内容
3. 最后使用 `std::io` 获取几个用户输入的路径，并调用 `file_size` 函数尝试获取文件大小，并处理可能的错误。

将上述代码合在 `main` 函数中分别调用即可，运行结果如图 1.1.1 所示（“haha”文件并不存在，因此触发了异常处理机制）。

```

__allenge@allenge: /Volumes/WorkSpace/Code/RustLearning/word_hello
└─$ cargo run
   Compiling word_hello v0.1.0
5 seconds left...
4 seconds left...
3 seconds left...
2 seconds left...
1 seconds left...
[INFO]: Count down finished!
##
# Host Database
#
# localhost is used to configure the loopback interface
# when the system is booting. Do not change this entry.
##
127.0.0.1    localhost
255.255.255.255 broadcasthost
::1         localhost
[INFO]: File read successfully!
Enter file paths (comma separated):
./test.txt, Cargo.toml, .gitignore, haha
[INFO]: File size of ./test.txt: 95 bytes
[INFO]: File size of Cargo.toml: 164 bytes
[INFO]: File size of .gitignore: 8 bytes
[ERROR]: Failed to open file haha: No such file or directory (os error 2)
===== [ERROR]: Failed to get file size for haha: File not found! =====

```

图 1.1.1: 任务一运行结果

```

__allenge@allenge: /Volumes/WorkSpace/Code/RustLearning/word_hello
└─$ cargo test
   Compiling word_hello v0.1.0
note: '[warn(non_snake_case)]' on by default
warning: `word_hello` (bin "word_hello" test) generated 6 warnings (run `cargo fix --bin "word_hello" --tests` to apply 1 suggestion)
Finished `test` profile [unoptimized + debuginfo] target(s) in 0.95s
Running unittests src/main.rs (target/debug/deps/word_hello-2feaa02bab451c31)

running 4 tests
test tests::test_area ... ok
test tests::test_unique_id ... ok
test tests::test_humanized_size ... ok
[INFO]: Hello, world!
[WARNING]: I'm a teapot!
===== [ERROR]: KERNEL PANIC! =====
test tests::LogTest ... ok

test result: ok. 4 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.00s

```

图 1.1.2: 单元测试结果

## 1.2 任务二：代码单元测试

### humanized\_size

1. 将字节数转换为人类可读的大小和单位。使用 1024 进制，并使用二进制前缀 (B, KiB, MiB, GiB) 作为单位
2. 补全格式化代码，使得你的实现能够通过如下测试：

```

1 #[test]
2 fn test_humanized_size() {
3     let byte_size = 1554056;
4     let (size, unit) = humanized_size(byte_size);
5     assert_eq!("Size : 1.4821 MiB", format!("{}", size, unit));
6 }

```

Rust

一个简单的进制转换，函数的具体实现如下：

```

1 fn humanized_size(size: u64) -> (f64, &'static str) {
2     let kb = 1024;
3     let mb = kb * 1024;
4     let gb = mb * 1024;
5     let tb = gb * 1024;
6     if size < kb {
7         (size as f64, "B")
8     } else if size < mb {
9         ((size as f64) / kb as f64, "KiB")
10    } else if size < gb {
11        ((size as f64) / mb as f64, "MiB")

```

Rust

```
12     } else if size < tb {  
13         ((size as f64) / gb as f64, "GiB")  
14     } else {  
15         ((size as f64) / tb as f64, "TiB")  
16     }  
17 }
```

测试代码补全如下：

```
1 #[test]  
2 fn test_humanized_size() {  
3     let byte_size = 1554056;  
4     let (size, unit) = humanized_size(byte_size);  
5     assert_eq!("Size : 1.4821 MiB", format!("Size : {} {}", size, unit));  
6 }
```

Rust

测试运行结果见图 1.1.2。

### 1.3 任务三：彩色文字终端

### 1.4 任务四：枚举类型

### 1.5 任务五：元组结构体

## 2 YSOS，启动！

### 2.1 配置 Rust Toolchain

### 2.2 年轻人的第一个 UEFI 程序

## 3 思考题

### 3.1 现代操作系统的启动流程

### 3.2 Makefile 过程

### 3.3 Cargo 包管理工具

### 3.4 #[entry] 与 main

## 4 附加题

### 4.1 彩色的日志

### 4.2 Rust 实现的简单 shell

### 4.3 线程模型