

# 2024.04.01-2024.04.07-work-log

## 工作进展

本阶段完成的任务有：继续研究阅读Rust编译器的标准库的实现，为rtsmart-std项目编写标准输出库out，使得用户在编写程序时可以使用print!()和println!()等rust标准库中常用的输出库

本阶段为新建的标准库项目编写了标准输出库。代码思路有借鉴Rust标准库的源代码，做了较多的简化，核心最终落在调用libc中的输出函数printf()。

## 资料收集

Rust标准库源代码：<https://github.com/rust-lang/rust/tree/master/library/std>

Rust标准库源代码解析：<https://github.com/Warrenren/inside-rust-std-library>、<https://rustwiki.org/zh-CN/std/>

## Out模块

### StdOut

首先创建一个标准输出结构体，为它实现core库中的Write这一trait，使得它能够完成标准输出的工作

```
struct StdOut;

impl fmt::Write for StdOut {
    fn write_str(&mut self, s: &str) -> fmt::Result {
        fn rtt_kputs(s: *const u8) {
            unsafe { printf(s as _); }
        }
        puts(s, rtt_kputs);
        Ok(())
    }
}
```

这里的printf()函数即为libc中调用的C库中的输出函数，这一代码通过puts函数进行输出，以下是puts函数的具体实现：

```
fn up_cast(a: usize, b: usize) -> usize {
    let r = a / b;
    return if a % b == 0 { r } else { r + 1 };
}

pub(crate) fn puts(str: &str, kp: fn(s: *const u8)) {
    let str = str.as_bytes();
    let mut buf = [0 as u8; 129];
    for i in 0..up_cast(str.len(), 128) {
        let end = min(128, str.len() - i * 128);
        for j in 0..end {
            buf[j] = str[(j + i * 128) as usize];
        }
        buf[end] = 0;
        kp(buf.as_ptr())
    }
}
```

```
}  
}
```

通过设计一个缓冲数组buf，每次写入最多128个字节，然后循环写入，直到写入完毕即可。

## print函数

有了标准输出结构体后，就可以用它来实现输出函数print

```
pub fn _print(args: fmt::Arguments) {  
    unsafe {  
        StdOut.write_fmt(args).unwrap_unchecked();  
    }  
}
```

由于StdOut实现了Write trait，因此可以直接调用write\_fmt方法进行格式化输出。

## print marco

常用的输出宏有println!()和print!(), 均为声明宏，为了仿造标准库的设计，我们也编写了对应的宏实现代码，直接调用前面写的print函数即可

```
#[macro_export]  
macro_rules! print {  
    ($($arg:tt)*) => ({  
        $crate::out::_print(format_args!($($arg)*));  
    });  
}  
  
#[macro_export]  
#[allow_internal_unstable(print_internals, format_args_nl)]  
macro_rules! println {  
    ($($arg:tt)*) => ({  
        $crate::out::_print(format_args_nl!($($arg)*));  
    });  
}
```

## debug和log宏

在实现print函数后，即完成了输出功能的设计，其他使用到输出功能的模块函数和宏也可以借助这一功能来实现

如dlog宏：

```
#[cfg(debug_assertions)]  
#[macro_export]  
macro_rules! dlog {  
    ($($arg:tt)*) => ({  
        $crate::println!("[DBG][{}:{}] {}",  
            $crate::out::file!(), $crate::out::line!(), format_args!($($arg)*));  
    });  
}
```

可以在调试模式下打印信息的同时包含文件名和行号等调试信息

log宏:

```
#[macro_export]
macro_rules! log {
    ($($arg:tt)*) => ({
        $crate::println!("[LOG][{}:{}] {}",
            $crate::out::file!(), $crate::out::line!(), format_args!($($arg)*));
    });
}
```

可以在打印日志的同时包含文件名和行号等调试信息

再比如deg宏:

```
#[cfg(debug_assertions)]
#[macro_export]
macro_rules! dbg {
    () => {
        $crate::println!("[{}:{}] ", $crate::out::file!(), $crate::out::line!());
    };
    ($val:expr $(,)? ) => {
        match $val {
            tmp => {
                $crate::println!("[{}:{}] {} = {:#?}",
                    $crate::out::file!(), $crate::out::line!(),
                    $crate::out::stringify!($val), &tmp);
            }
        }
    };
    ($($val:expr),+ $(,)? ) => {
        ($($crate::dbg!($val)),+,)
    };
}

#[cfg(not(debug_assertions))]
#[macro_export]
macro_rules! dbg {
    () => {};
    ($val:expr $(,)? ) => {};
    ($($val:expr),+ $(,)? ) => {};
}

#[cfg(not(debug_assertions))]
#[macro_export]
macro_rules! dlog {
    ($($arg:tt)*) => {};
}
```

在对象实现了debug这一trait时, 就可以通过转换为 `{:#?}` 格式输出, 打印出其对象的详细信息, 同时也有包含文件名和行号等调试信息

但如果对象没有实现debug这一trait, 那么就无法输出成功了

# 总结

---

本周的工作进展主要是为新建的标准库编写了标准输出库，这是我们编写的第一个面向用户使用的库，代码有借鉴Rust编译器源码中标准库的源代码。

下周或下下周我们计划会编译一个简单的rust程序，观察代码能否成功编译。并在qemu上测试一下标准输出库是否可用。