```
/srcs (cloc)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
Rust                            69            563            916           5843
D                              418            899              0           2880
JSON                           389              0              0            391
make                             3             32              7            115
TOML                             6             24              9             92
Assembly                         3              3             26             86
Dockerfile                       1              4              5             31
Markdown                         1              8              0             10
-------------------------------------------------------------------------------
SUM:                           890           1533            963           9448
-------------------------------------------------------------------------------

├── bootloader(内核依赖的运行在 M 特权级的 SBI 实现，本项目中我们使用 RustSBI)
│   └── rustsbi-qemu.bin(可运行在 qemu 虚拟机上的预编译二进制版本)
├── easy-fs(rCore 提供从内核中独立出来的一个简单的文件系统 EasyFileSystem 的实现)
│   ├── Cargo.toml
│   └── src
│       ├── bitmap.rs(位图抽象)
│       ├── block_cache.rs(块缓存层，将块设备中的部分块缓存在内存中)
│       ├── block_dev.rs(声明块设备抽象接口 BlockDevice，需要库的使用者提供其实现)
│       ├── efs.rs(实现整个 EasyFileSystem 的磁盘布局)
│       ├── layout.rs(一些保存在磁盘上的数据结构的内存布局)
│       ├── lib.rs
│       └── vfs.rs(提供虚拟文件系统的核心抽象)
├── initfs(Redox 提供的独立文件系统)
│   ├── Cargo.toml
│   └── src
│       ├── lib.rs(文件系统抽象)
│       ├── types.rs(相关数据结构)
├── LICENSE
├── os(内核实现放在 os 目录下)
│   ├── build.rs(基于应用名的应用构建器)
│   ├── Cargo.toml(内核实现的一些配置文件)
│   ├── Makefile
│   └── src
│       ├── config.rs(内核的一些配置，包括内存管理的相关配置)
│       ├── console.rs(将打印字符的 SBI 接口封装实现格式化输出)
│       ├── drivers(块设备驱动)
│       │   ├── block
│       │   │   ├── mod.rs(将不同平台上的块设备全局实例化为 BLOCK_DEVICE 提供给其他模块使用)
│       │   │   └── virtio_blk.rs(Qemu 平台的 virtio-blk 块设备)
│       │   └── mod.rs
│       ├── fs(文件系统接口)
│       │   ├── inode.rs(OSInode)
│       │   ├── mod.rs
│       │   └── stdio.rs(标准输入输出)
│       ├── entry.asm(设置内核执行环境的的一段汇编代码)
│       ├── lang_items.rs(需要提供给 Rust 编译器的一些语义项，目前包含内核 panic 时的处理逻辑)
│       ├── link_app.S(构建产物，由 os/build.rs 输出)
│       ├── linker-qemu.ld(控制内核内存布局的链接脚本以使内核运行在 qemu 虚拟机上)(修改：将跳板页引入内存布局)
│       ├── main.rs(内核主函数)
│       ├── mm(内存管理)
│       │   ├── address.rs(物理/虚拟 地址/页号的 Rust 抽象)
│       │   ├── frame_allocator.rs(物理页帧分配器)
```

```
│       │   ├── heap_allocator.rs(内核动态内存分配器)
│       │   ├── memory_set.rs(引入地址空间 MemorySet 及逻辑段 MemoryArea 等)
│       │   ├── mod.rs(定义了 mm 模块初始化方法 init)
│       │   └── page_table.rs(多级页表抽象 PageTable 以及其他内容)
│       ├── sbi.rs(调用底层 SBI 实现提供的 SBI 接口)
│       ├── sync(同步子模块 sync ，目前唯一功能是提供 UPSafeCell)
│       │   ├── mod.rs
│       │   └── up.rs(包含 UPSafeCell，它可以帮助我们以更 Rust 的方式使用全局变量)
│       ├── syscall(系统调用子模块 syscall)
│       │   ├── fs.rs(包含文件 I/O 相关的 syscall)
│       │   ├── mod.rs(提供 syscall 方法根据 syscall ID 进行分发处理)
│       │   └── process.rs(包含任务处理相关的 syscall)
│       ├── task(task 子模块，主要负责任务管理)
│       │   ├── context.rs(引入 Task 上下文 TaskContext)
│       │   ├── manager.rs(任务管理器)
│       │   ├── mod.rs(全局任务管理器和提供给其他模块的接口，支持进程)
│       │   ├── pid.rs(进程标识符和内核栈的 Rust 抽象)
│       │   ├── processor.rs(处理器管理结构)
│       │   ├── switch.rs(将任务切换的汇编代码解释为 Rust 接口 __switch)
│       │   ├── switch.S(任务切换的汇编代码)
│       │   └── task.rs(任务控制块 TaskControlBlock 和任务状态 TaskStatus 的定义，支持进程管理机制)
│       ├── timer.rs(计时器相关)
│       └── trap(Trap 相关)
│           ├── context.rs(包含 Trap 上下文 TrapContext)
│           ├── mod.rs(包含 Trap 处理入口 trap_handler，有时钟中断相应处理，基于地址空间，支持进程系统调用)
│           └── trap.S(包含 Trap 上下文保存与恢复的汇编代码)
├── rust-toolchain(控制整个项目的工具链版本)
└── user(应用测例)
├── Cargo.toml
├── Makefile
└── src
    ├── bin(基于用户库 user_lib 开发的应用，每个应用放在一个源文件中)
    │   ├── covid.rs(生产者消费者测试，灵感来源于期末考试)
    │   ├── exit.rs(exit 测试)
    │   ├── hello_world.rs(经典 hello world)
    │   ├── huge_write.rs(文件写测试)
    │   ├── initproc.rs(用户运行的 init 程序)
    │   ├── lock.rs(锁测试)
    │   ├── ls.rs(打印当前目录)
    │   ├── nonlock.rs(无锁测试)
    │   ├── send.rs(IPC测试)
    │   ├── sleep.rs(任务调度测试)
    │   ├── thread.rs(多线程测试)
    │   ├── user_shell.rs(shell)
    │   └── yield.rs(主动移交控制)
    ├── console.rs(通内核态，用户态实际调用内核态)
    ├── lang_items.rs
    ├── lib.rs(用户库 user_lib)
    ├── linker.ld(应用的链接脚本，将所有应用放在各自地址空间中固定的位置)
    └── syscall.rs(包含 syscall 方法生成实际用于系统调用的汇编指令，各个具体的 syscall 都是通过 syscall 来实现的)
```