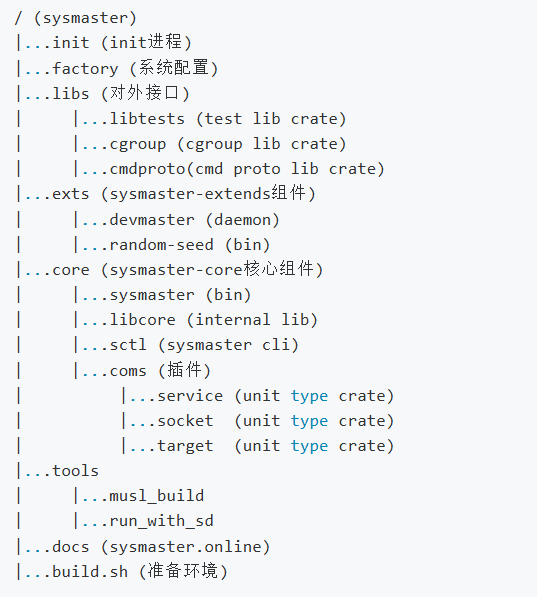
***（1）整个项目的构成***



***（2）runtime.rs的impl块简要介绍***

分析的功能主要位于init的runtime.rs文件，runtime.rs中不同impl 块的作用：

new：构造函数，用于初始化 Runtime 结构体。

register 和 deregister：用于注册和注销事件源。

load\_config：加载配置文件。

reap\_zombies：回收僵尸进程。

handle\_signal：处理接收到的信号。

handle\_timer：处理计时器事件。

handle\_socket：处理套接字事件。

pid\_is\_running：检查特定 PID 的进程是否在运行。

start\_bin：启动二进制文件。

runloop：运行事件循环。

is\_running 和 set\_state：检查和设置初始化程序的状态。

reload：重新加载配置。

is\_reexec 和 reexec：检查是否需要重新执行和执行重新启动。

kill\_sysmaster 和 exit：终止系统进程和退出程序。

***（3）init的进程回收机制***

在Linux系统中，子进程的回收是一个重要的管理任务，它涉及到资源的有效利用和系统的稳定性。如果父进程未能妥善回收其子进程，就可能导致僵尸进程（Zombie Process）的产生 。

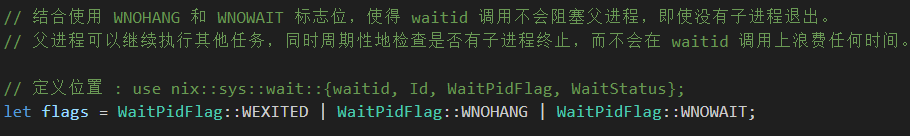
1. 僵尸进程占用系统资源：当子进程结束时，如果父进程没有通过wait()或waitpid()函数来回收子进程的资源和状态信息，子进程就会变成僵尸进程。僵尸进程会占用系统资源，如进程描述符（PCB）。

2）系统稳定性受到影响：僵尸进程的存在可能会影响系统的稳定性。如果系统资源被大量僵尸进程占用，可能会导致新的进程无法被创建，从而影响系统的正常运行。

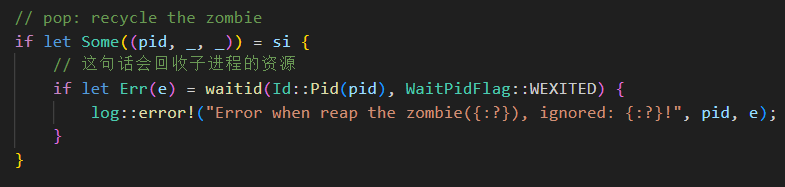
3）信号处理的延迟性：系统向父进程发送SIGCHLD信号来通知子进程的结束。如果父进程没有处理这个信号，子进程同样会变成僵尸进程。

master-init的init进程针对上述缺点的改进如下所示

1. 非阻塞事件驱动：



使用WNOHANG和WNOWAIT 标志位，使得waitid ()调用不会阻塞父进程。父进程可以继续执行其他任务，同时周期性地检查是否有子进程终止，而不会在 waitid()调用上浪费任何时间。使用非阻塞模式代替了原来的阻塞模式。



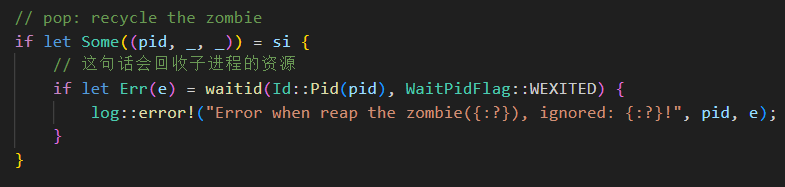
具体而言这句代码完成了子进程的回收

1. 使用mio库达成了高效的事件循环



这句代码指定了事件循环库为mio库，mio是Rust 语言中的一个快速、低级别的 I/O 库，专注于非阻塞 API 和事件通知，用于构建高性能的 I/O 应用程序，同时尽可能减少操作系统抽象的开销。

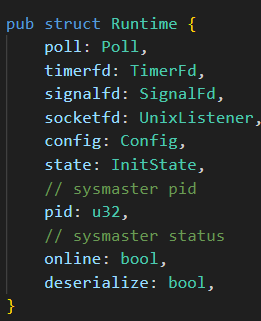
3）错误处理的改进



错误处理改成了日志打印而不是错误退出，提高了系统的稳定性

4）信号量的使用

Master-init 通过提供一种更简洁的接口来初始化和使用信号量，从而减少了信号量的使用复杂性。它可能通过封装信号量的创建和销毁过程，以及提供更直观的等待和释放操作，来帮助开发者更容易地管理线程间的同步。

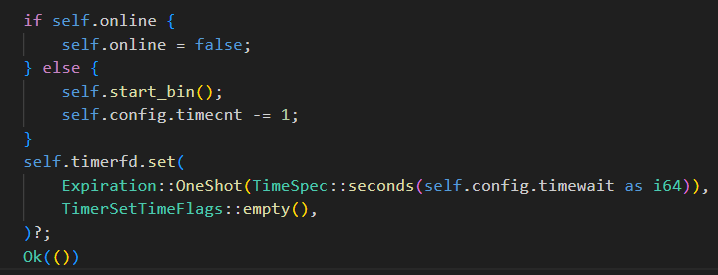


定义信号结构的位置

***（4）监控sysmaster-core的部分***

Init进程监控sysmaster-core的部分主要是handle\_timer和handle\_socket。handle\_timer负责系统的定时器事件，handle\_socket负责了系统联网的事件处理。

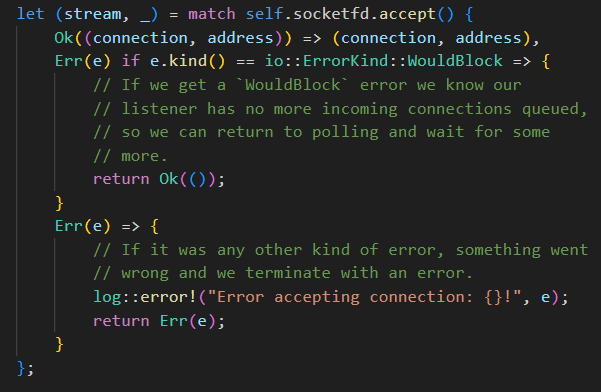
1. handle\_timer模块：



处理定时器事件，用于定期检查 sysmaster-core 的状态，并在必要时重新启动它

1. handle\_socket

配合master\_core的联网处理模块，使用accept()函数接受来自poll的传递的调用



更新对应sysmaster-core的状态值：online和pid

