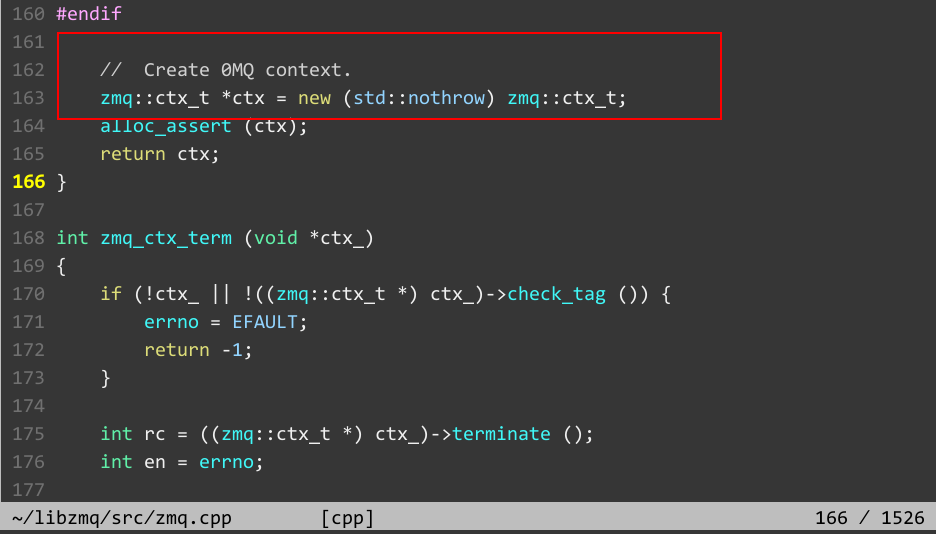
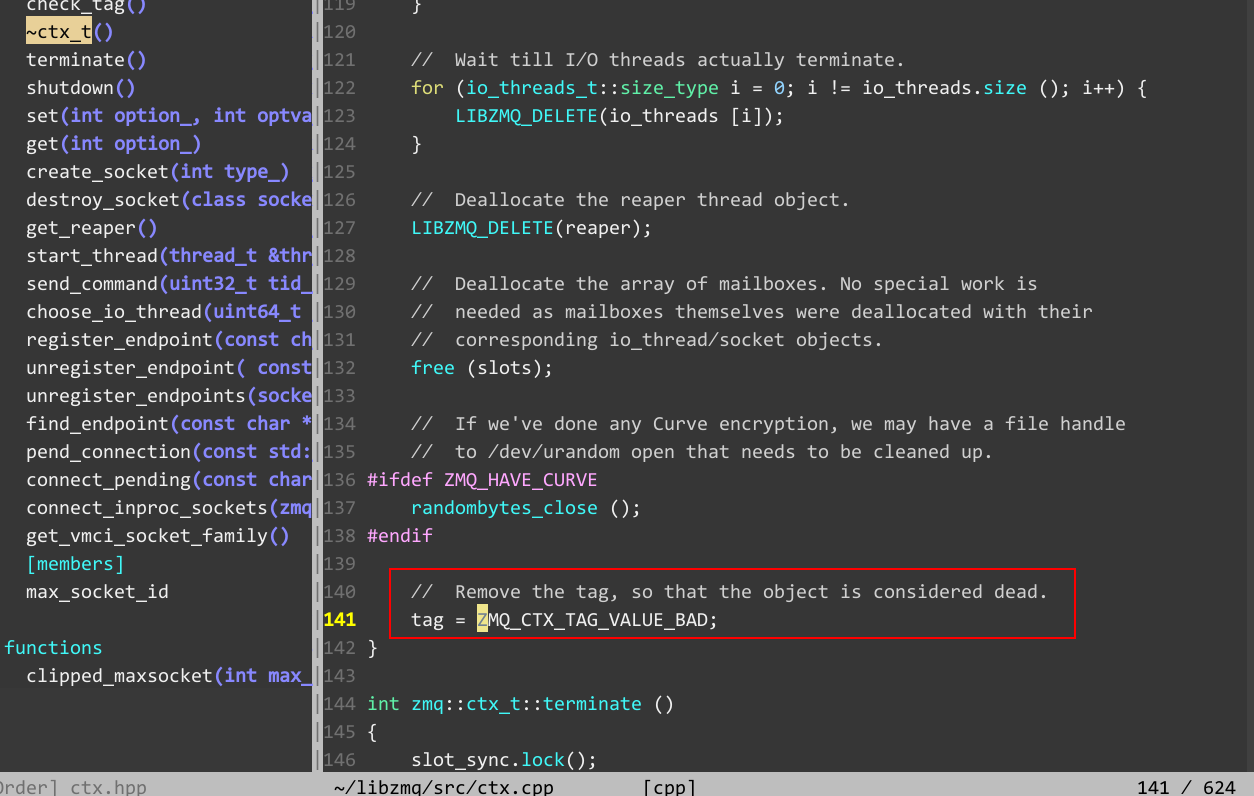
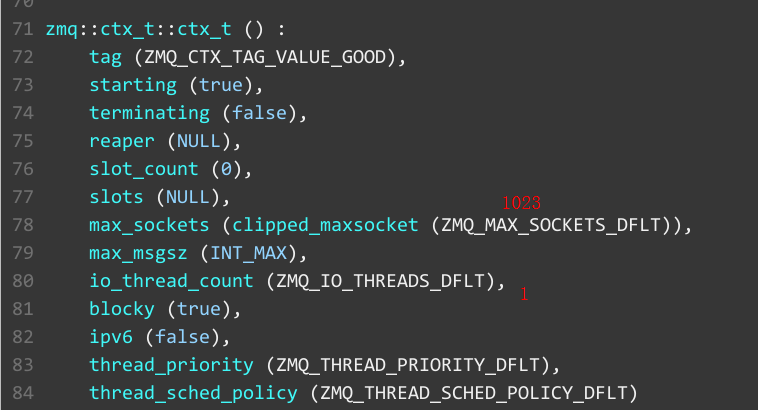
1. 不抛出异常的 new 操作符



1. ctx\_t 类的析构函数中会将 tag 置位为ZMQ\_CTX\_TAG\_VALUE\_BAD

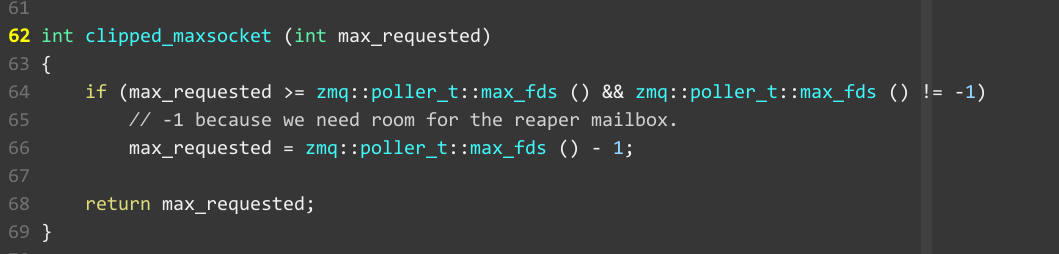


1. ctx\_t 初始化参数



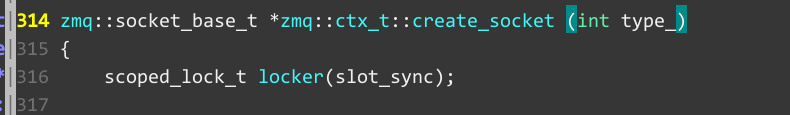
默认 io 线程启动 1 个

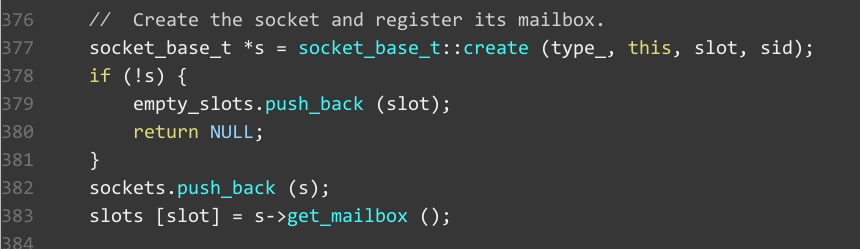
max\_sockets 数目还与系统支持的最大文件描述符有关。



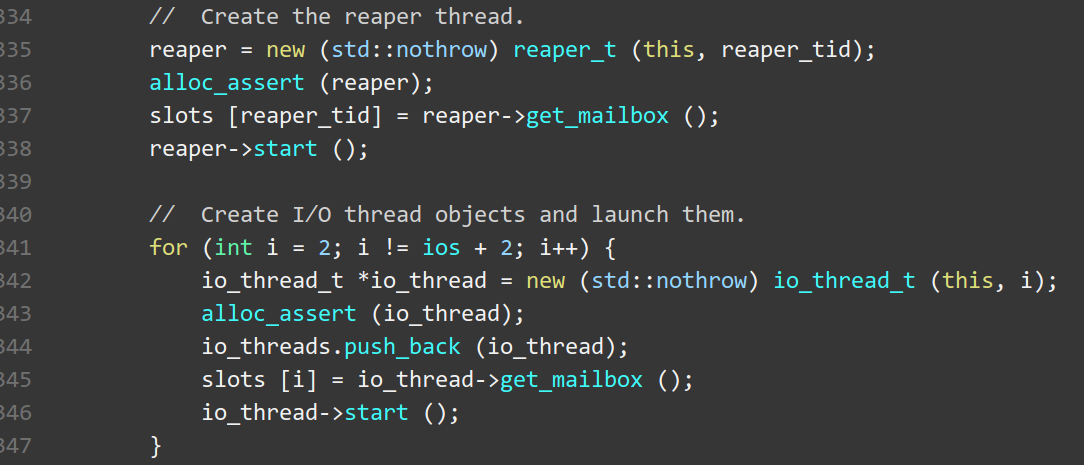
1. zmq\_socket() 创建 socket 过程

实际将调用的是



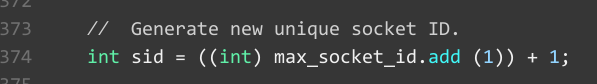


第一次调用会启动两个线程：reaper 和 io



创建 socket 涉及到下面几点:

1. 创建一个 unique socket ID

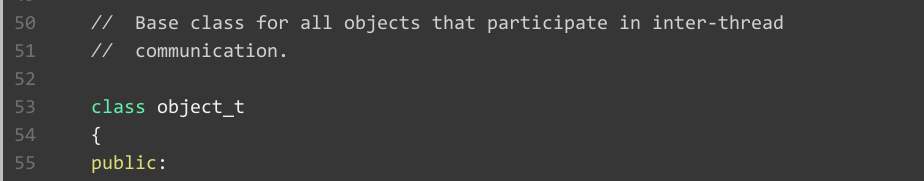


1. 注册 socket 的 mailbox
2. 将 socket 信息添加到 ctx 中（调用 zmq\_ctx\_term 时需要通知 socket）
3. 探究 reaper 线程

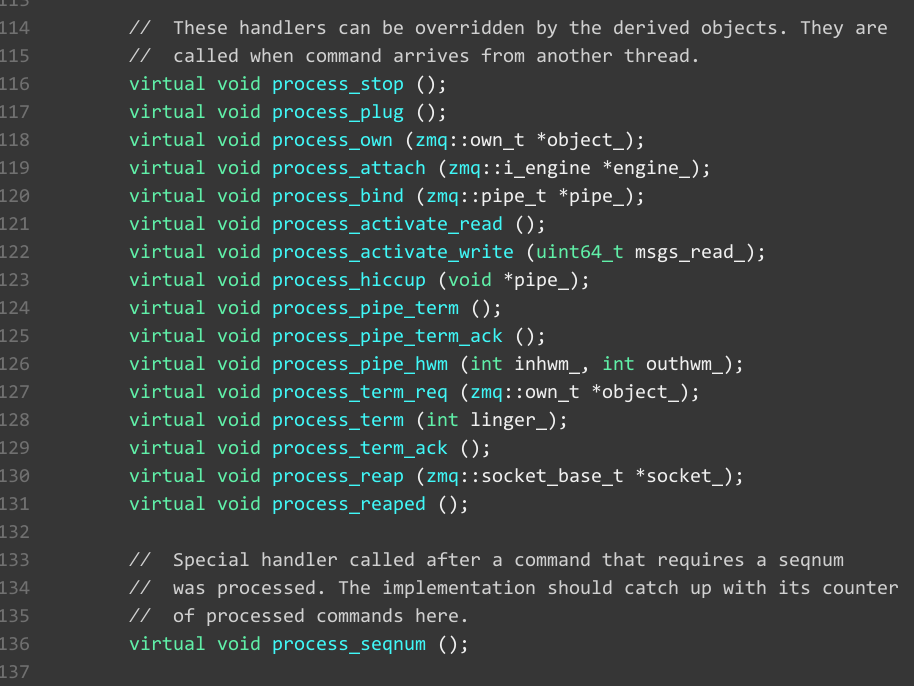
（reaper.hpp/cpp）

继承两个类: object\_t 和 i\_poll\_events

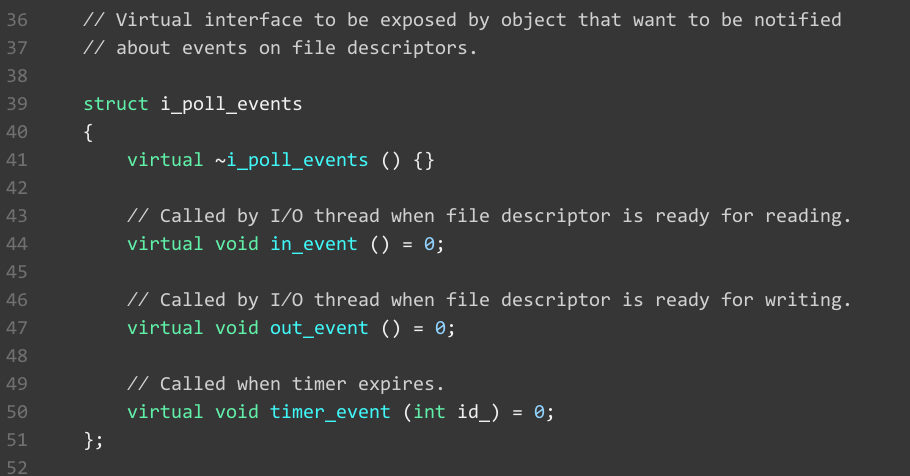
1）object\_t 类



有一组虚函数



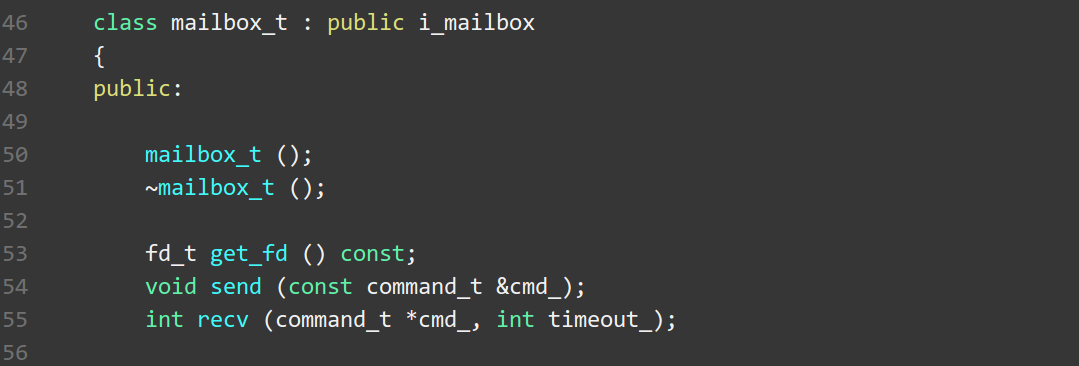
2) i\_poll\_events 类



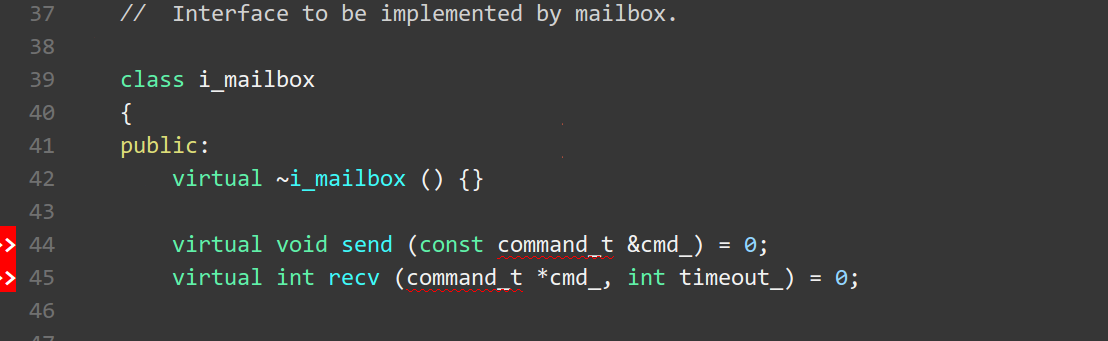
包含 mailbox\_t 类的成员变量

1）mailbox\_t 类

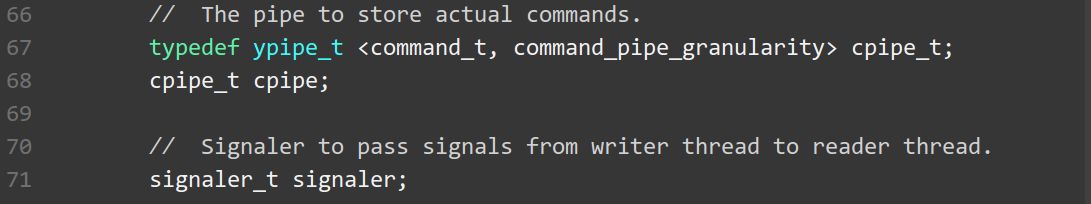
继承自 i\_mailbox\_t 类



i\_mailbox\_t 为接口类，有两个纯虚函数

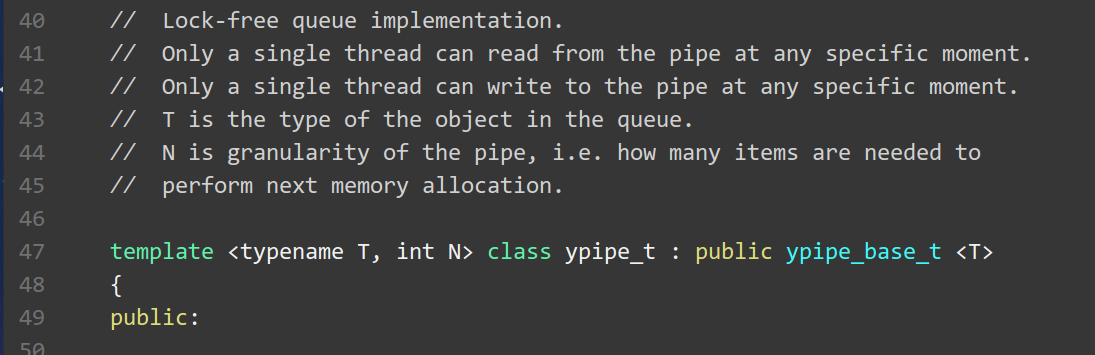


mailbox\_t 类包含有 ypipe\_t 和 signaler 两个类

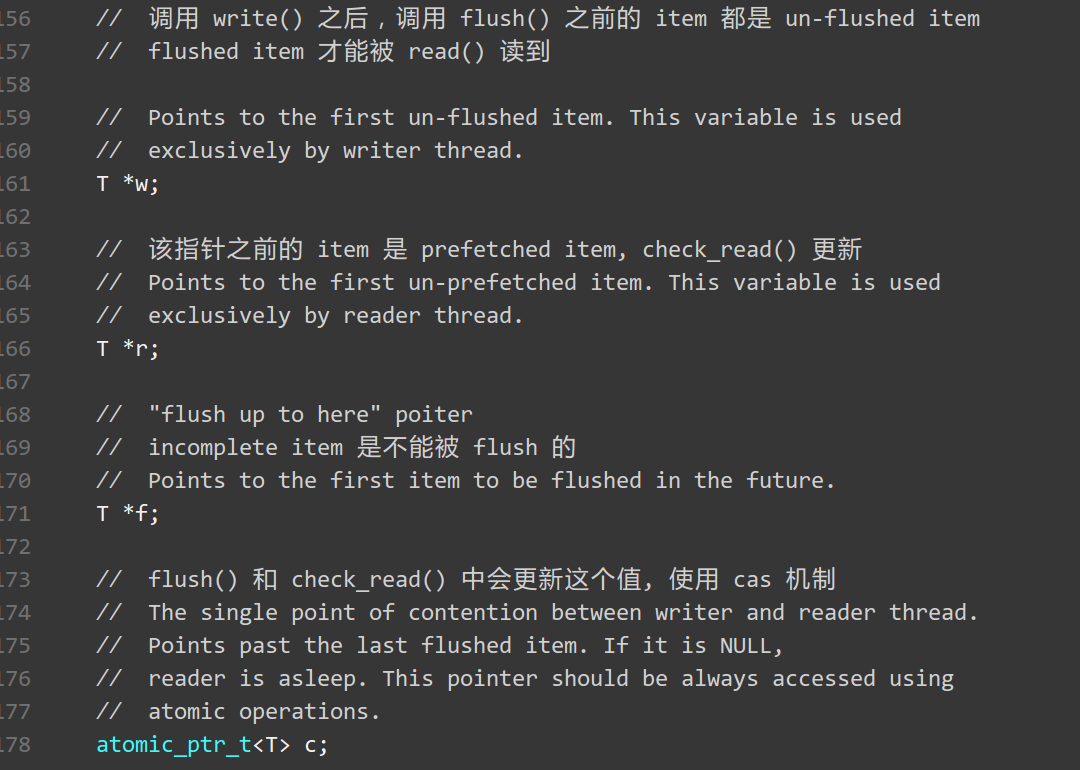


1.1）ypipe\_t 类

无锁队列

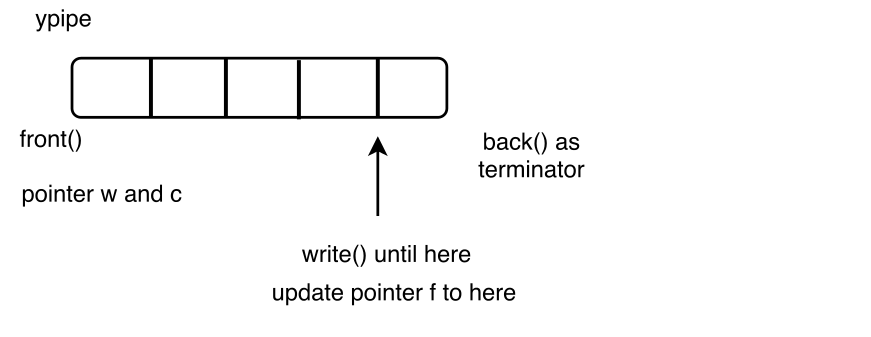


主要有 4 个指针：w, r, f, c

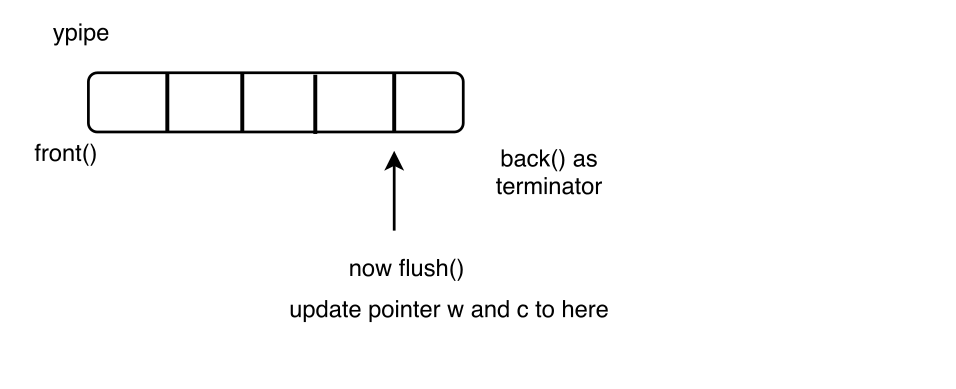


下面展示 ypipe 的工作方式

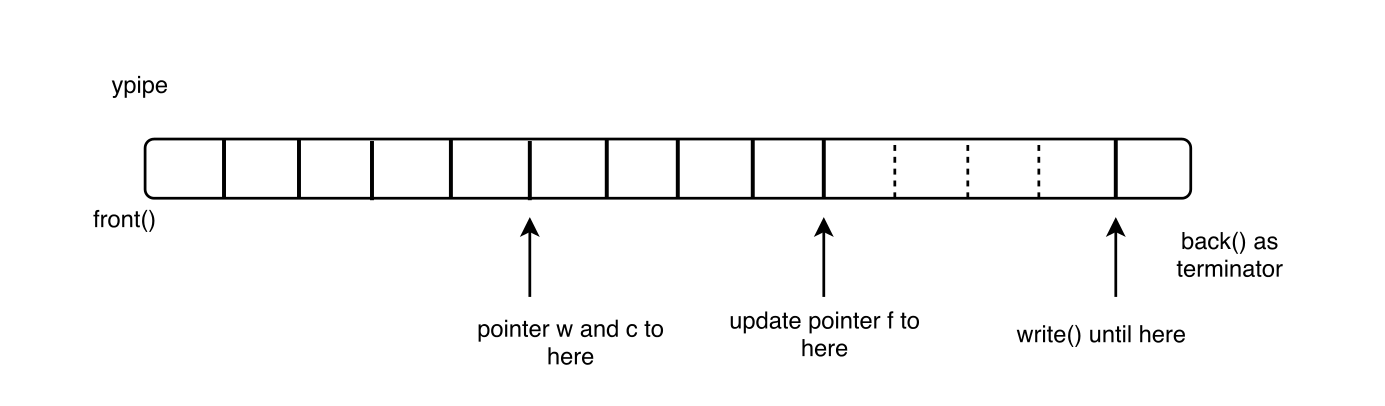
(1) 初始时 4 个指针都指向 front()

(2) 调用 write() 写入4 个元素，此时会更新 指针 f，若 4 个元素都是 complete 类型, 则 f 会指向 back()，即 terminator 元素；指针 w 和 c 不变。如下图：

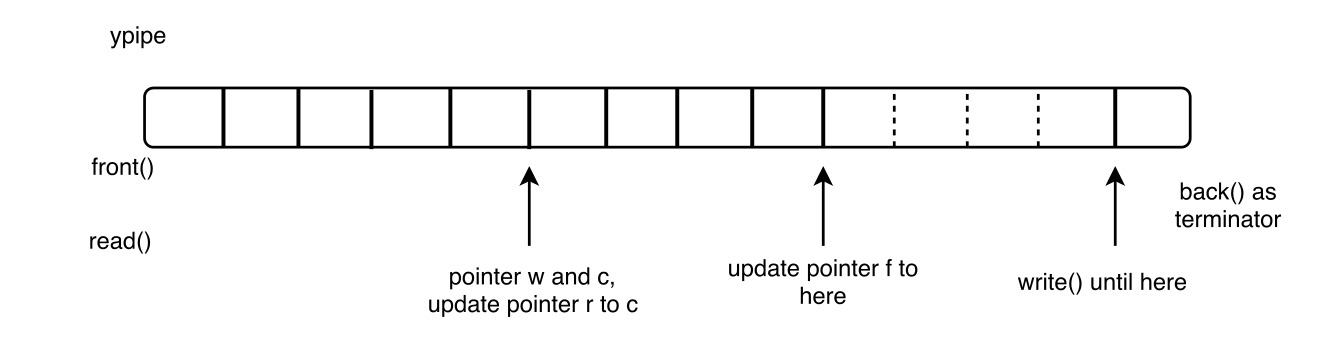
(3) 调用 flush() 刷新，此时指针 w 和 c 会更新为指针 f 的值。（若之前c的值为 null，则flush会返回false，暗示上一次reader没有读到数据，进入了asleep）



(4) 继续调用 write()，但最后一个元素为 incomplete 类型，如下图中虚线框所示，最后一个元素写入了4个部分，并且仍然是 incomplete；此时指针f将指向这个incomplete元素的首部分；



(5) 调用 read(), 更新指针 r 为指针c的值，返回 front() 元素

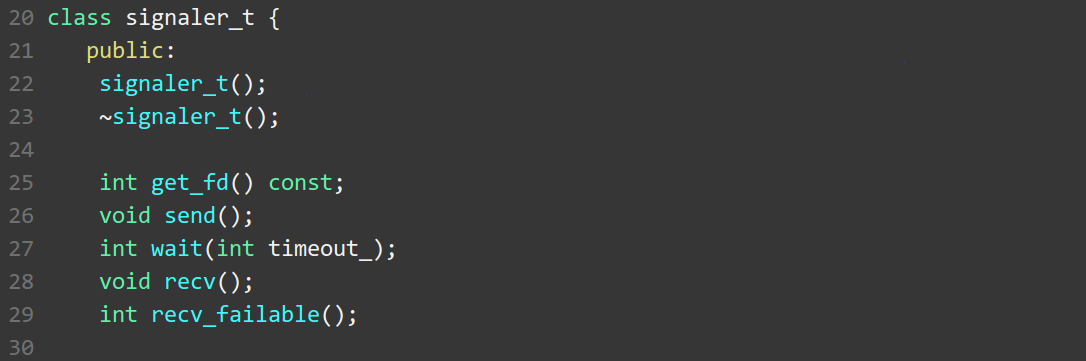


(6) 之后对 read() 的调用将不更新指针 r；直到r 指向 front()，此时又将 r 更新为c值，若 r 还是指向 front()，说明没有元素可读，此时将c的值置为null。

1.2）signaler\_t 类

阅读 man eventfd

封装eventfd()，写线程和读线程间的事件通信 （wait/notify）

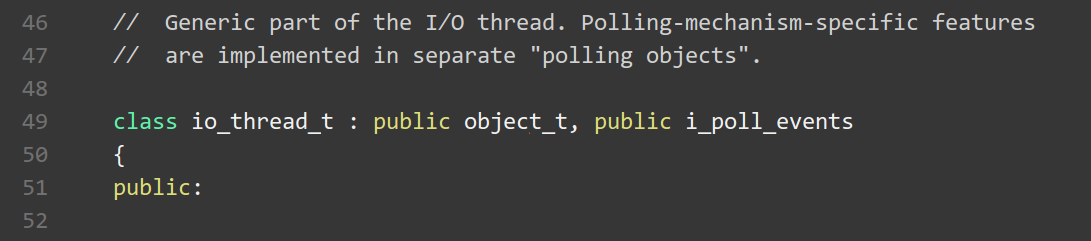


poller\_t 类

封装多路复用 api

reaper\_t 启动的线程执行的就是 poller\_t 的 loop() 方法

6. 探索 io\_thread\_t 类



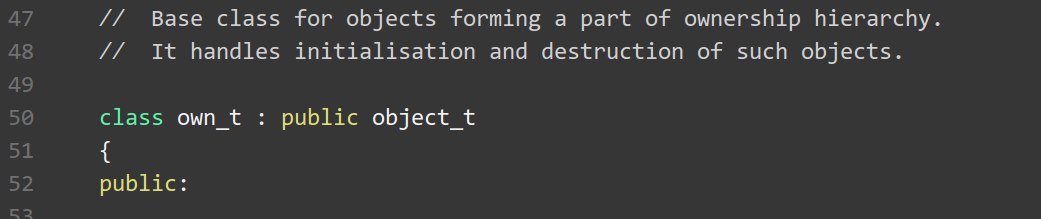
Q: epoll\_wait 为一个独立线程，在另一个线程中 删除监听 fd 会怎样？

A: epoll\_wait 会剔除删除的监听 fd.

See: <https://stackoverflow.com/questions/7058737/is-epoll-thread-safe>

线程执行的是 poll\_t::loop()

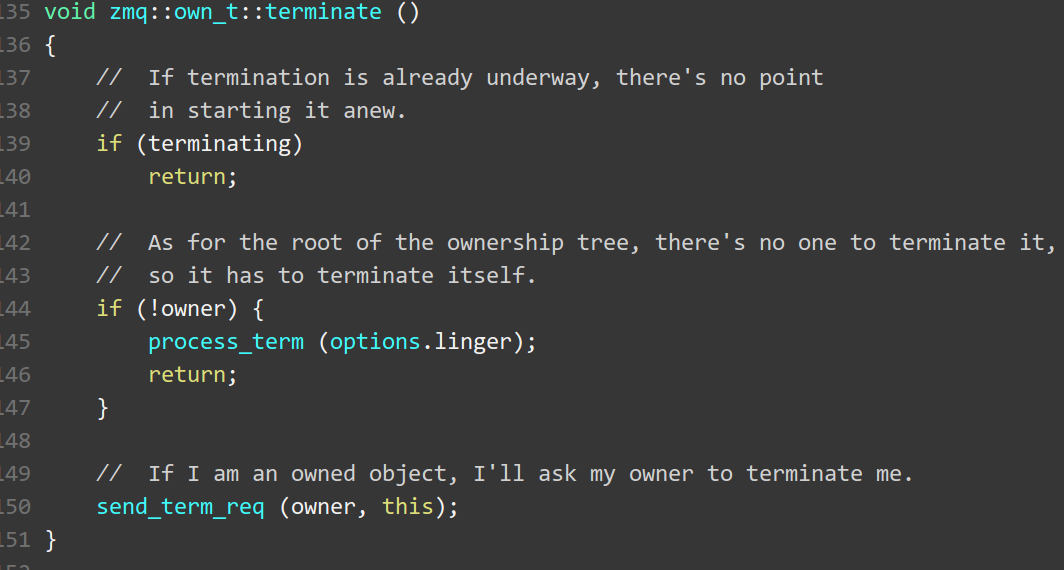
7. own\_t 类



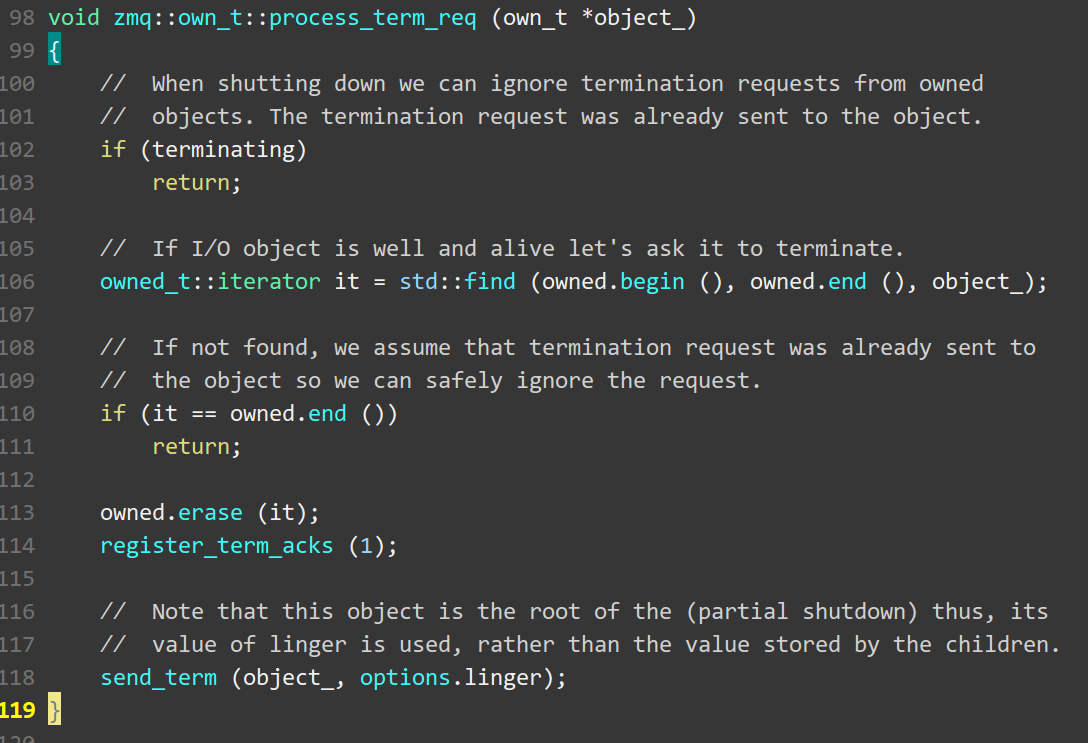
归属关系形成树形结构，节点为线程，节点的终止(terminator)需要其父节点发送 termination 信号。

父节点需要等其所有的子节点都终止后才能终止，父节点发送 term req信号给子节点请求子节点终止，子节点需要发送 term ack 信号给父节点确认其已终止。

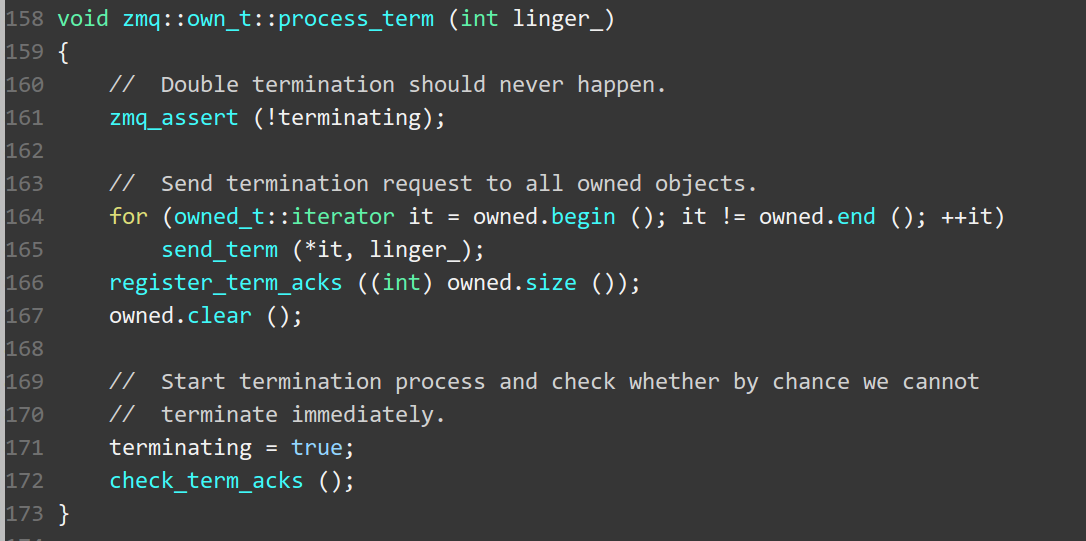
子节点向父节点发送 term req



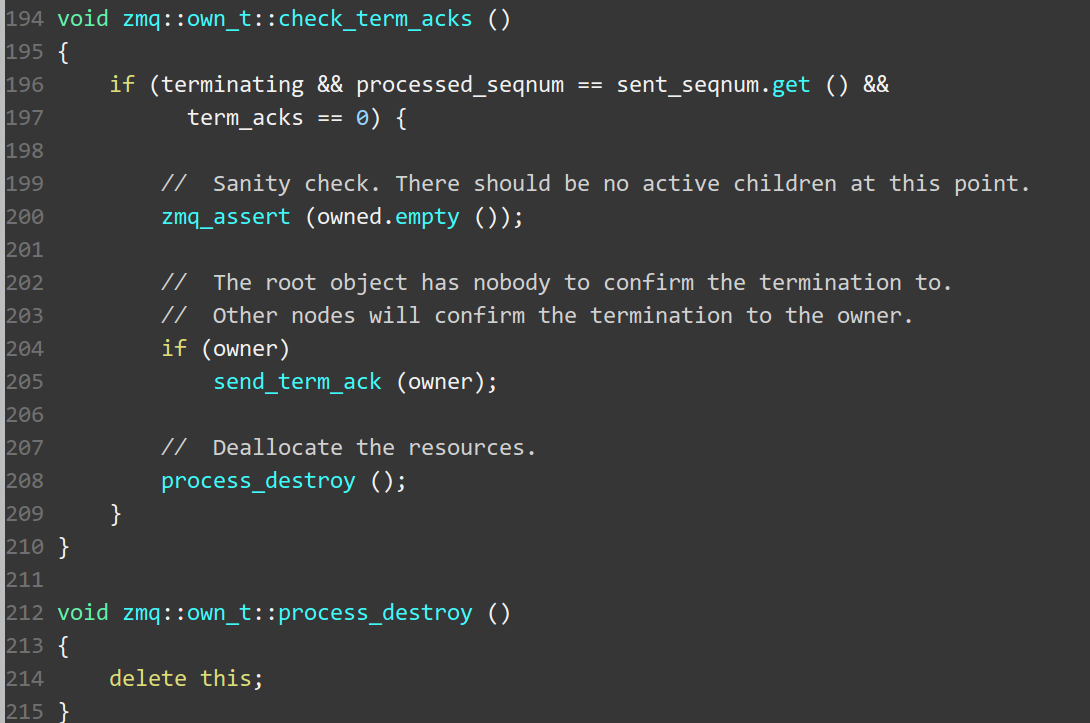
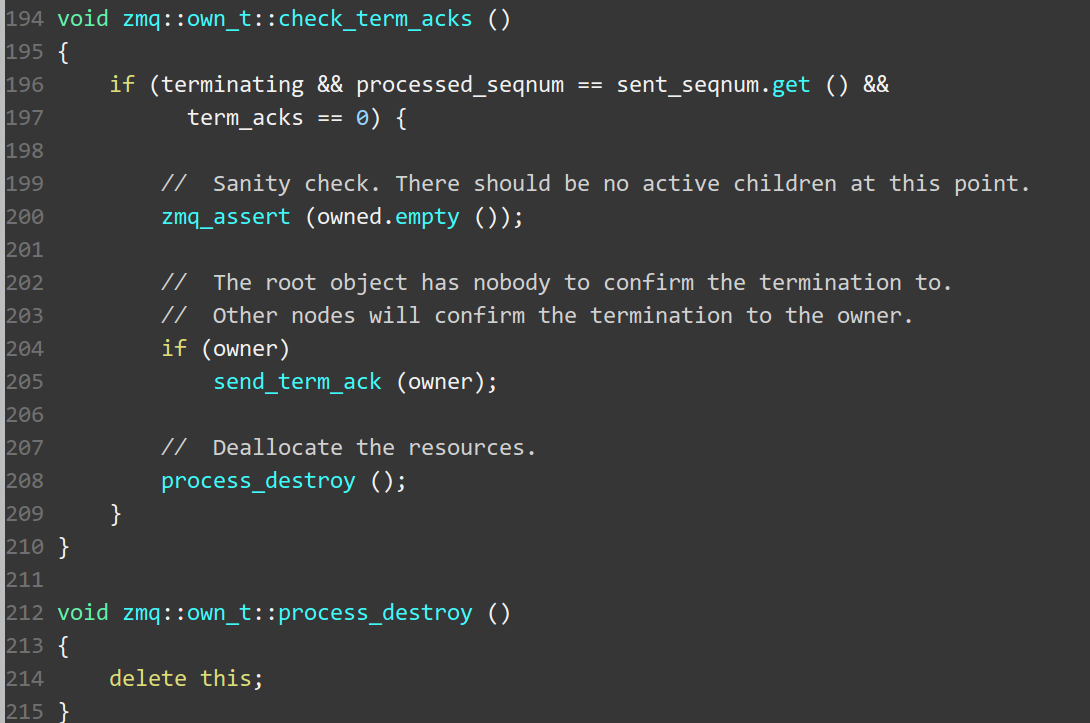
Owner 处理 term req



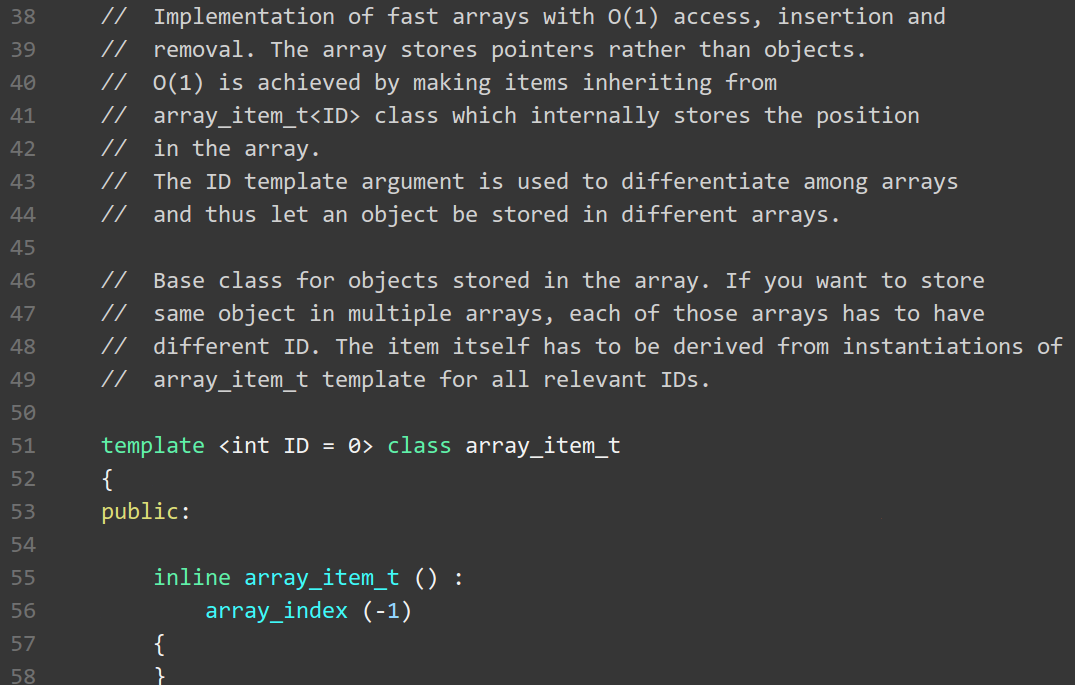
执行 term 命令



父节点收到所有子节点终止信号 ack 后，销毁自身



8. array\_item\_t 类



9. pipe\_t 类

包含一个读管道和一个写管道

有一个对端管道 peer (pipe\_t) 类，对端的读写管道和自身正好相反

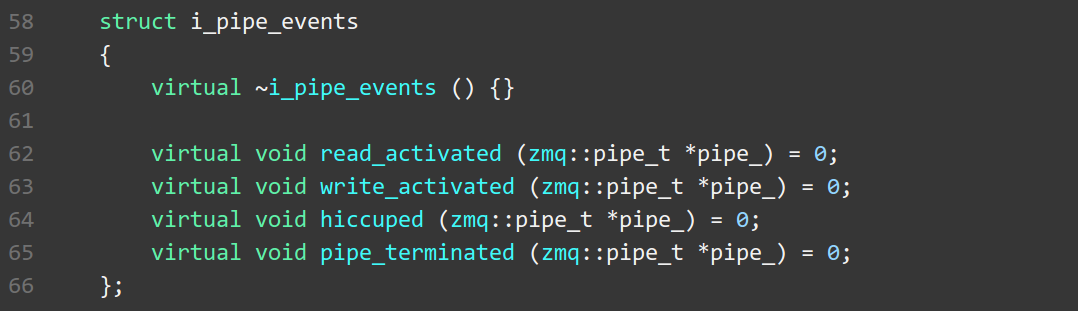
有一个类似 TCP 的状态转换机

TODO 画出状态机

10. i\_pipe\_events 类

接口类

pipe\_t 类产生事件的处理函数接口



11. socket\_base\_t 类

1）inproc

connect 和 bind 的顺序无所谓，实现方式：使用将 pending connect 添加到容器中，当bind调用时，查找该容器，将对应的 pending connect 连接上

