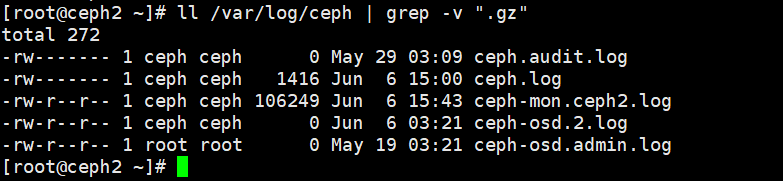
ceph作为大型软件，必须要设计自己的log, 本章内容就分析下ceph的log实现。

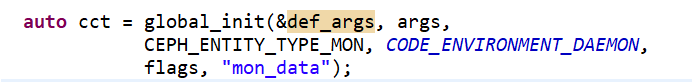
ceph的每一个monitor,osd进程都有自己的日志文件，记录进程的运行信息，日志文件存放在/var/log/ceph中



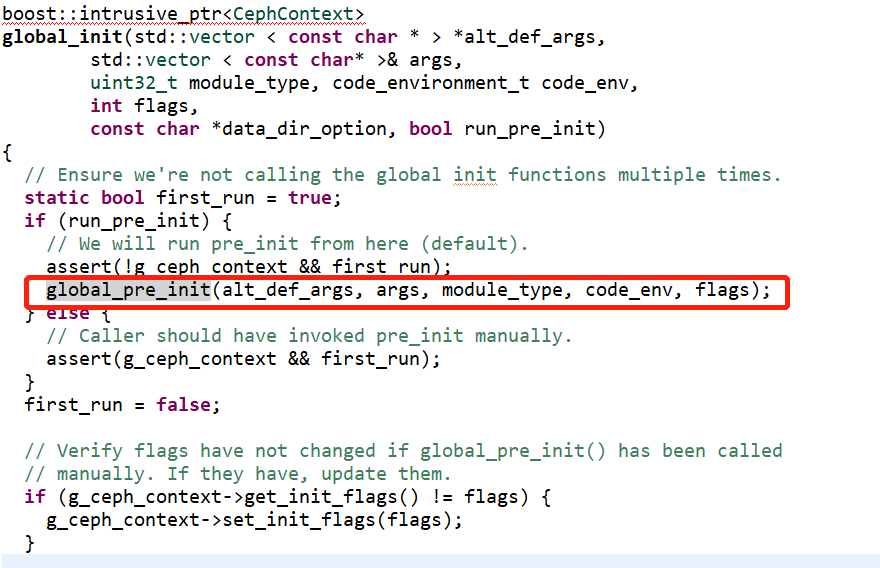
通过跟踪分析monitor的实现，了解log的实现。

ceph-mon 初始化，代码在ceph\_mon.cc中，main()函数开始：

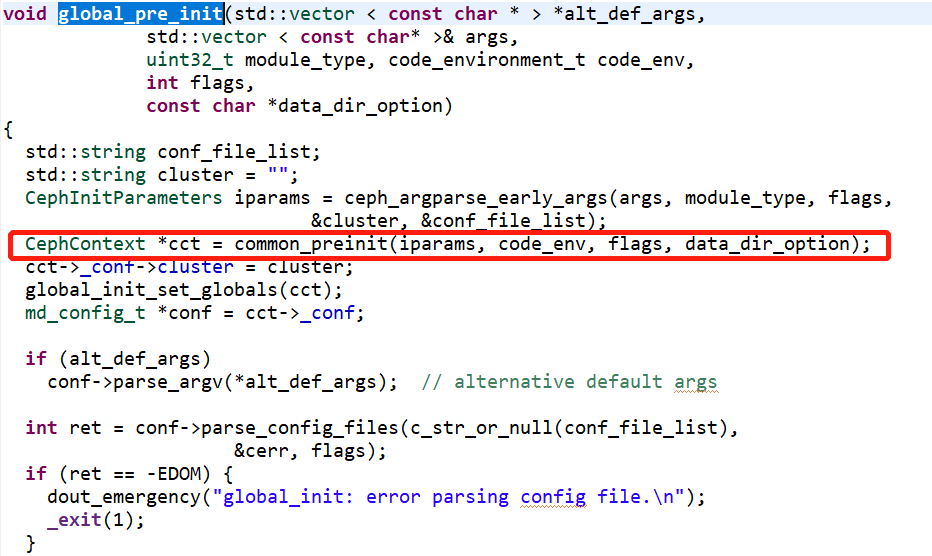
调用global\_init()函数



在global\_init()函数中，会继续调用global\_pre\_init()函数



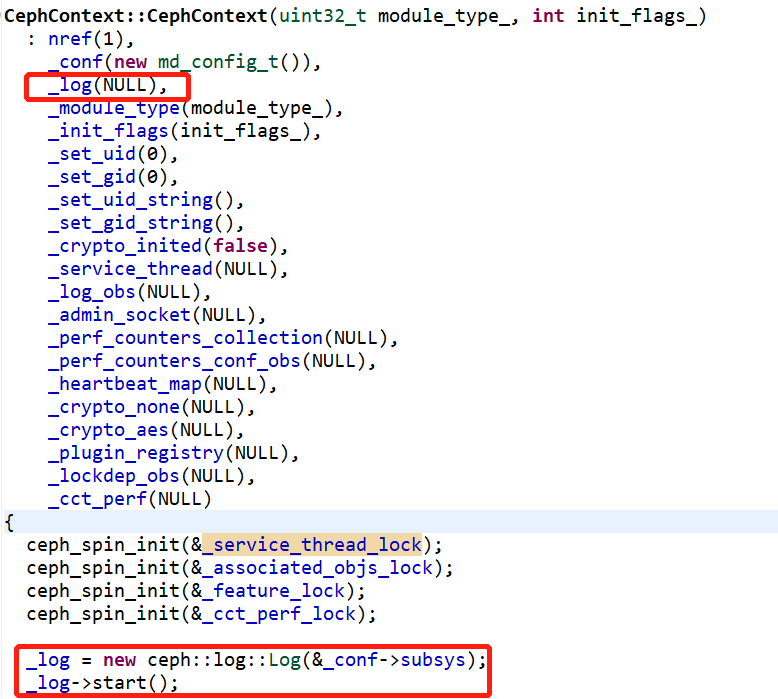
在global\_pre\_init()函数中会调用common\_preinit()函数构建CephContext, 每一个后台进程都有一个数据结构CephContext代表这个进程的上下文信息。



进入common\_preinit()函数：



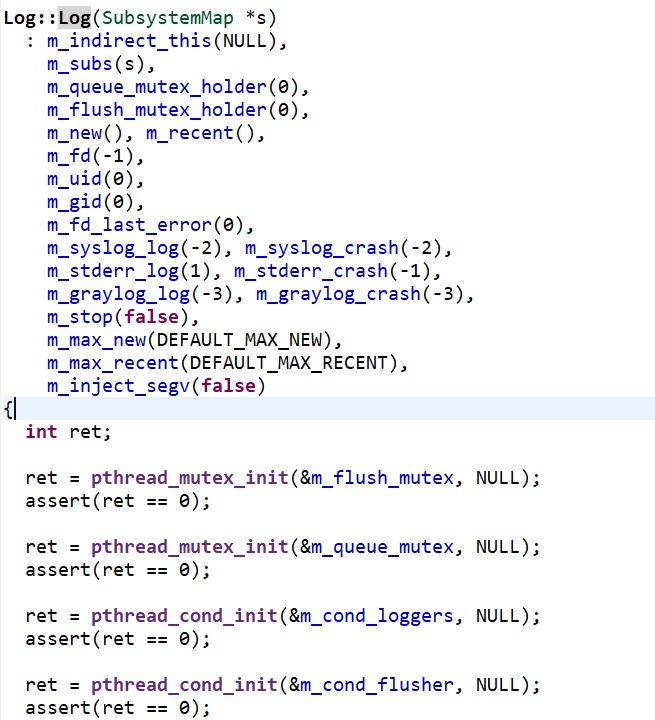
函数new 了一个CephContext对象：

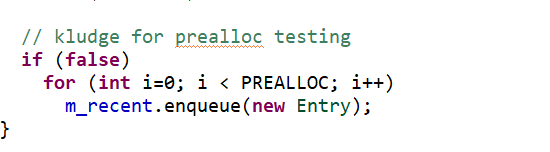


首先创建成员变量\_log，然后new log实例，初始化这个实例：

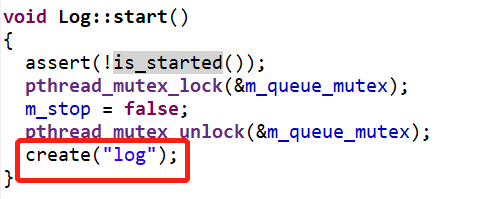
初始化互斥量  m\_flush\_mutex 和m\_queue\_mutex

初始化条件变量 m\_con\_logger 和m\_cond\_flusher

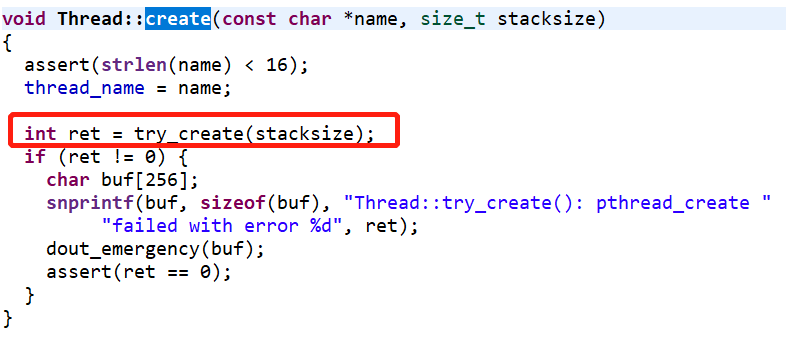




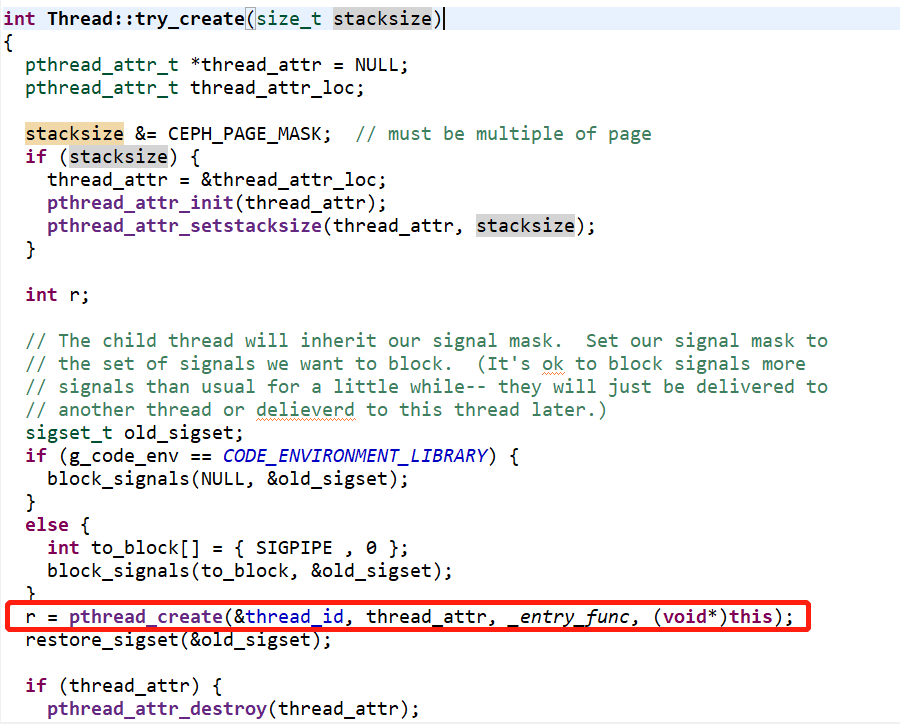
接下来启动了log线程



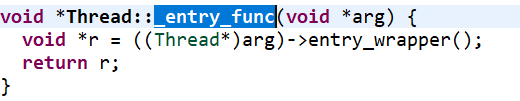
函数 create(),其实就是调用Thread::create()函数，因为class Log : private Thread



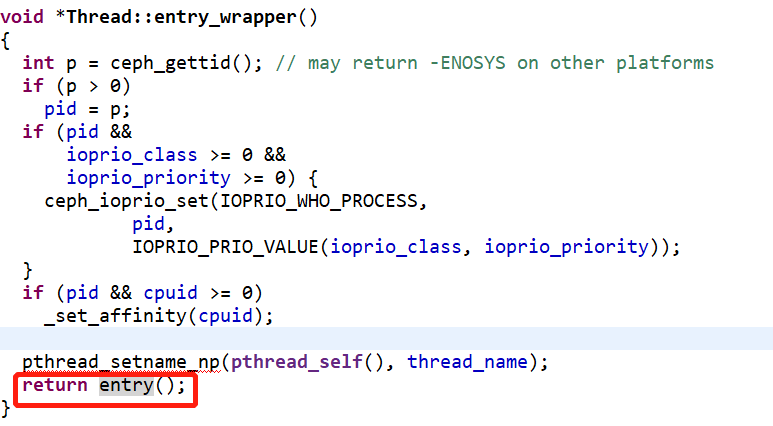
try\_create()函数就是创建log线程：



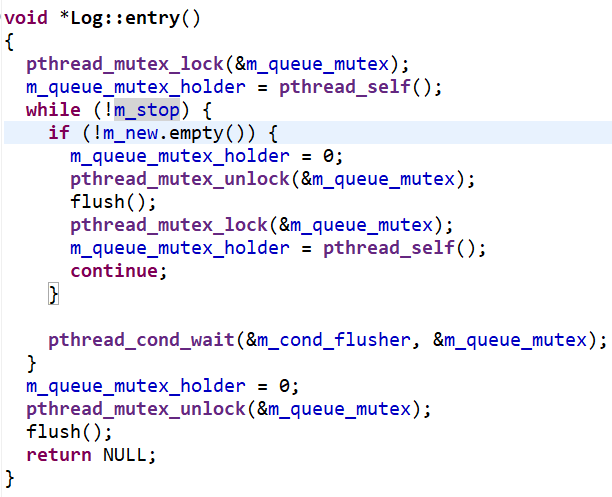
函数pthread\_create()就是Linux创建线程函数。执行的函数体就是\_entry\_func，参数是log实例



即执行：



entry()其实就是执行log实例的entry()函数。



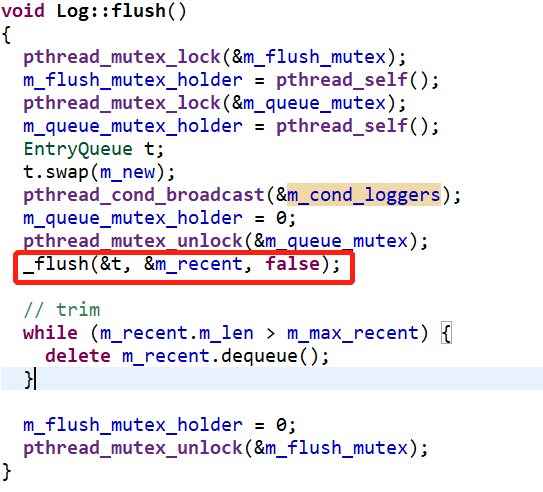
这个函数有一个while循环，这个循环的逻辑为：

m\_stop来控制线程是否终止

如果m\_new 这个队列不空，就调用flush，负责写入log

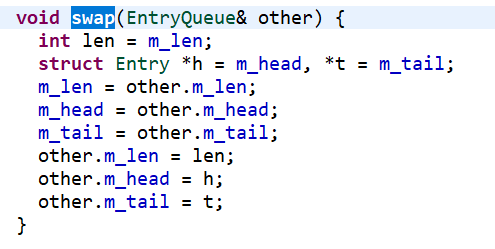
如果队列空了，条件等待，有新的log出现在队列上，会通知到这个线程

接着看flush()函数：



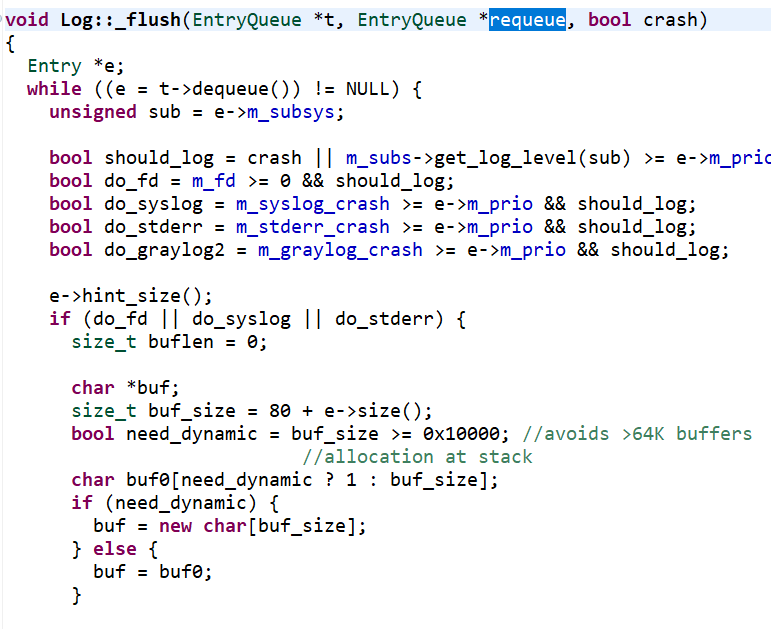
首先，创建一个临时队列，t，执行swap将m\_new里面的log接了过去。所谓接过去，不过是将指针指向队列内容的事情交给临时队列，m\_new 头指针和尾指针置成NULL

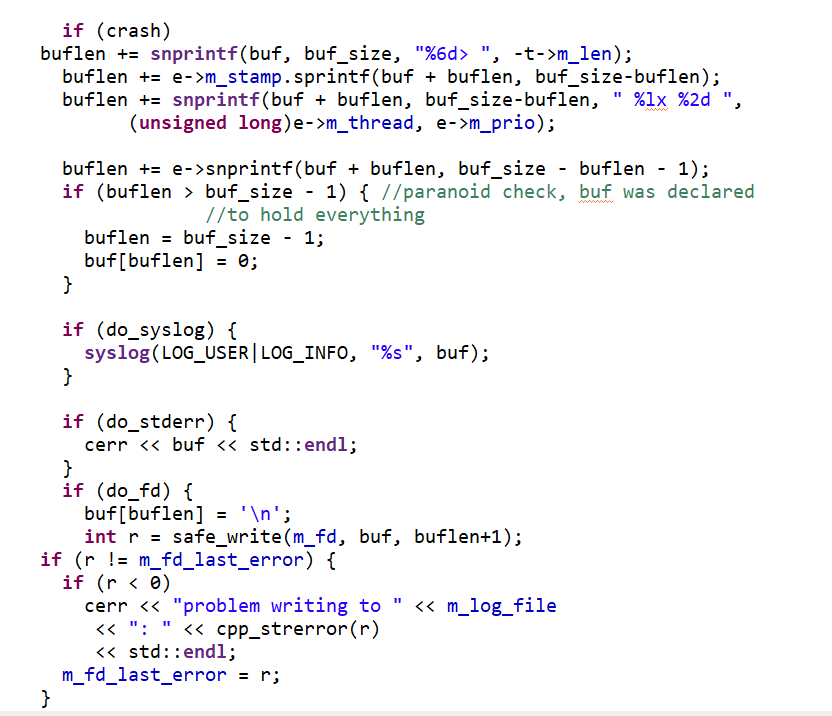
做了这件事之后，m\_new又变成了空的队列，好心地给其他线程发了广播之后，就可以解锁m\_queue\_mutex互斥量了。

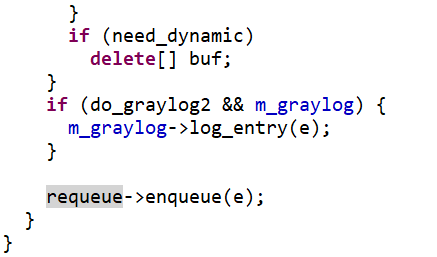


调用\_flush()函数写日志：

有的log需要写入日志文件，有的需要写入syslog，有些日志需要写入stderr  
同时，日志还有一个优先级的概念，因此会根据should\_log 来控制。







从这个函数知道：

ceph维护了一个m\_recent队列，所有的消息都会存放到该队列中去，哪怕优先级比较低，不会打印到日志文件中去，就是\_flush 函数中的 enqueue做的事情。

但当然该队列有大小限制，超过大小会删除一部分队列里的日志

