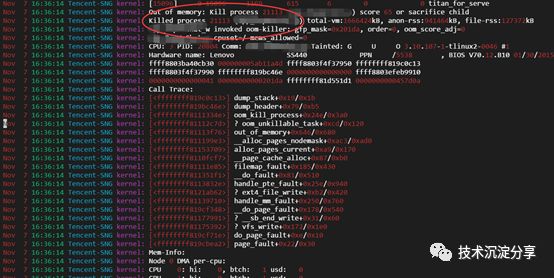
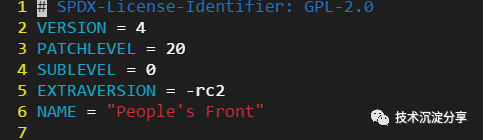
OOM（out of memory）在linux是比较常见一个问题。可能每位开发同学都遇到过。在linux下，有一种内存泄露情况是用户态（业务进程）或内核态（内核协议栈）占用内存不断增加，从而导致物理内存不足，系统根据一定规则（算法），然后拉起oom-killer，将候选进程kill掉，从而释放出内存以保证系统能正常运行。在出现这种情况的时候，可以通过查看/var/log/messages，记录了OOM时系统日志。如下图所示：



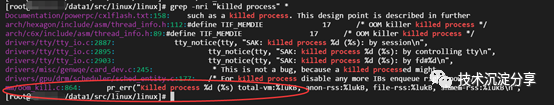
那么，到底系统是怎么选择要KILL哪些进程的呢？为什么要kill进程A，而不是进程B呢？遵循的是什么规则呢？大致的流程是这样的：

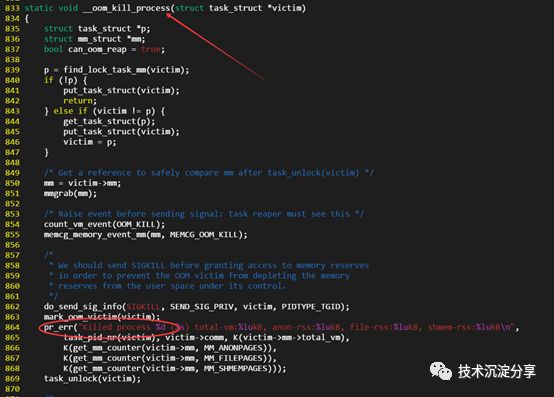
我们从日志和代码来剖析这个规则。

首先，下载内核代码，github上有，内部的tlinux代码也开源了（虽然把一些敏感代码去掉了，但不影响整体阅读和分析）。本文使用的是linux内核版本是4.20.0。

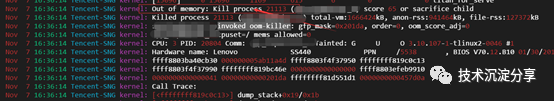


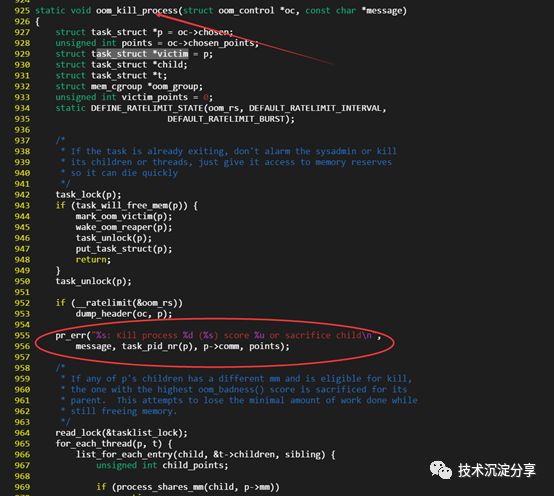
结合/var/log/messages的日志信息，先grep出代码位置。





再结合Out of memory:Kill process 21113日志可以追溯oom\_kill\_process函数

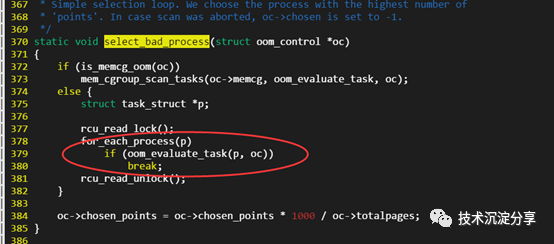




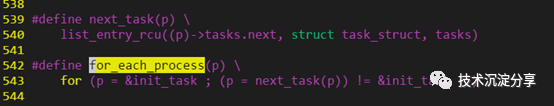
再向上追溯，定位到函数out\_of\_memory



找到了关键函数select\_bad\_process，通过函数名可以猜出选择规则在这里。进入看下



其中，for\_each\_process是宏，主要是系统所有进程，系统所有进程由task\_struct结构构成的一个链表组成（进程结构是task\_struct，这是一个超复杂的结构，是我目前为止见到的最长的一个struct，具体位置在include/linux/sched.h）的遍历操作（这里不做详细阐述）



进入oom\_evaluate\_task



