# KFileMonitor程序-内存泄漏分析报告

【摘要】

本报告采用VMMap内存分析工具、VLD内存泄露分析工具以及VS内置诊断工具，对**KFileMonitor程序**的内存占用情况、内存泄露问题以及内存优化方案进行说明。报告中将对各个分析工具的使用进行说明，分析以降低特定功能的内存使用量、排查内存占用异常上涨为目的，通过中断分析调用栈、记录内存分配比较差异的方法来进行。

【关键词】

内存泄漏、内存优化

**一、项目内存占比分析**

VMMap是对进程所用虚拟、物理内存进行分析的工具，可以统计进程所使用的虚拟、物理内存并以条形图与文字的形式进行呈现。VMMap的过滤和刷新功能可以及时观察进程内存使用的来源以及占用情况。

运行SVG编辑器项目，并使用VMMap工具进行检测，可首先观察到如下界面，图1.1。

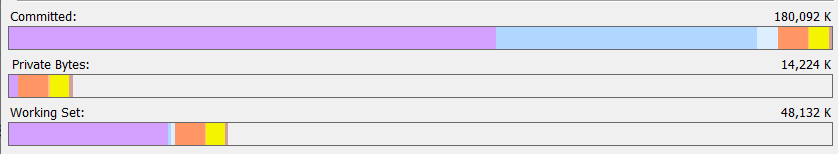
KFileMonitor程序**启动后**，进程内存总体信息状态如下：

图1.1 VMMap中进程内存总体情况（启动后）

其中顶部为进程基本信息，下部为各类类型空间占用条形图。Committed表示进程向操作系统所申请的需要使用到的虚拟内存大小；Private Bytes表示进程所私用使用的虚拟内存大小；Working Set表示进程目前所使用的物理内存大小。

从图中可知，KFileMonitor程序**启动后**，虚拟内存分配空间大小为180,092K，项目所私有使用的虚拟内存空间大小为14,224K，进程所实际使用的物理内存空间大小为48,132K。

在工具的中部区域，对进程内存总体信息进行了进一步分析

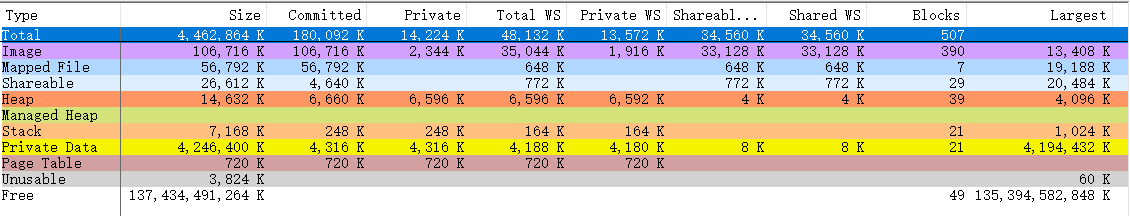


图1.2 VMMap中进程内存总体情况次详细信息（启动后）

横向表头分别代表总体内存大小、申请内存大小、私有内存大小，总体物理内存带下、私有物理内存大小、共享物理内存大小等类型。纵向表头代表总体、进程自身可执行文件以及包含的所有库、内存映射文件、共享内存、堆区空间、栈区空间等类型。

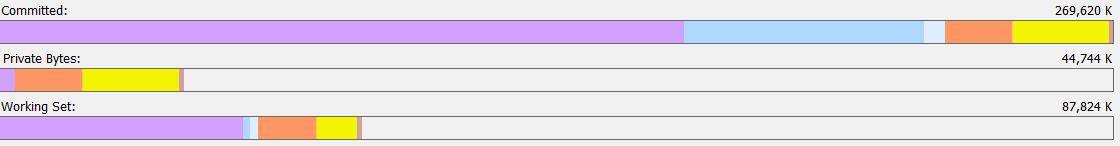
KFileMonitor程序运行一段时间后后，进程内存总体信息状态如下：

图1.3 VMMap中进程内存总体情况

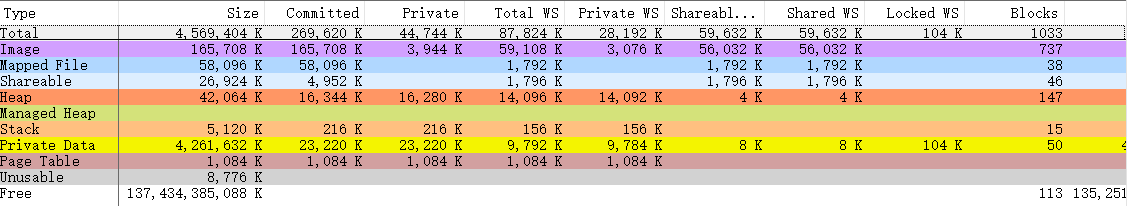


图1.4 VMMap中进程内存总体情况次详细信息

从图中可知，KFileMonitor程序执行时，虚拟内存分配空间大小为269,620K，项目所私有使用的虚拟内存空间大小为44,744K，进程所实际使用的物理内存空间大小为87,824K。

其中new与malloc等分配的空间为堆区域，栈区域空间会主动释放，因此进一步对堆区空间进行分析。

KFileMonitor程序**启动后**，堆区空间状态如下：

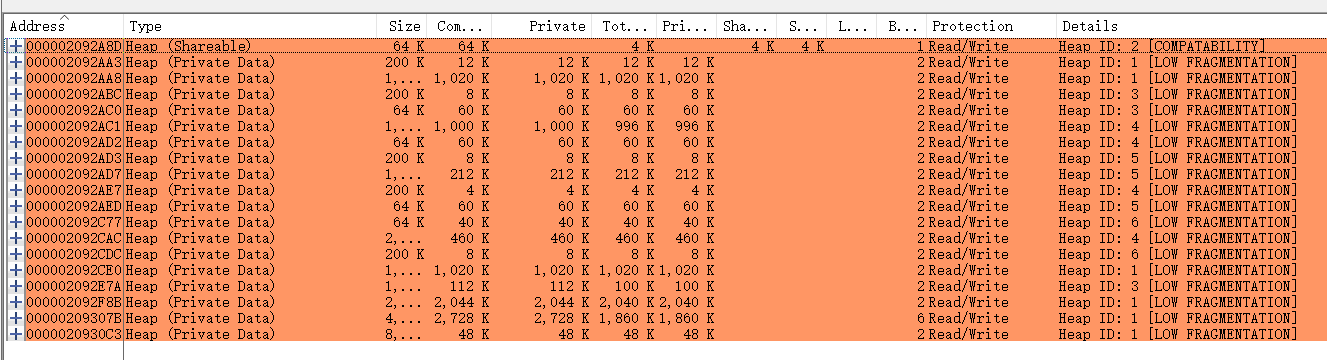


图1.8 VMMap中进程堆区空间状态（启动后）

 KFileMonitor程序运行一段时间后，堆区空间状态如下：

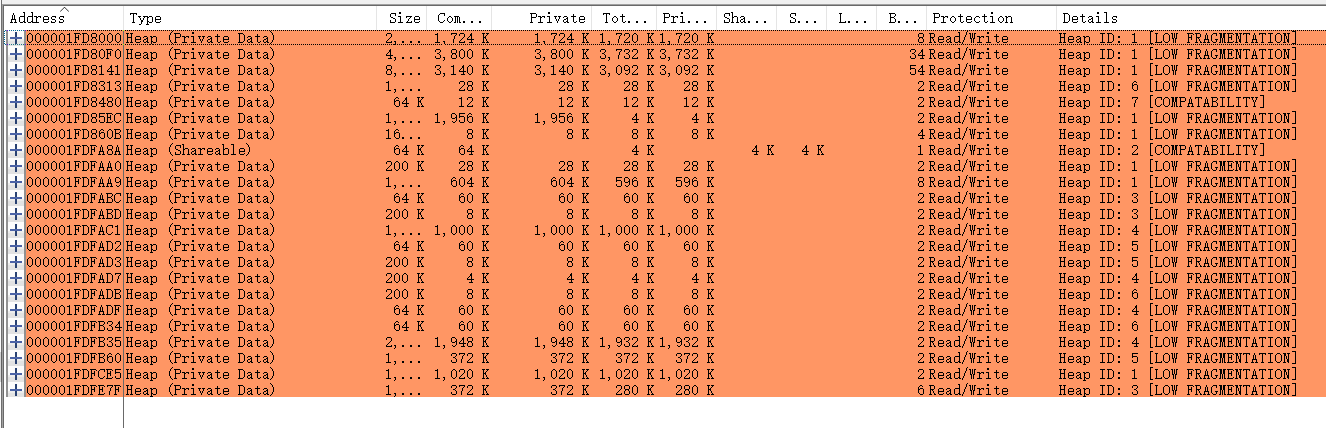


图1.10 VMMap中进程堆区空间状态

**二、项目内存泄露点**

 Visual Leak Detector，它是一款用于 Visual C++免费的内存检测工具。VLD可以得到内存泄漏点的调用堆栈、泄露内存的完整数据，还可以设置内存泄露报告的级别。在VS项目中，加上<vld.h>头文件后运行一次程序，即可得到内存泄露分析结果。可以**检测出疑似内存泄露**的具体行号，new生成的对象使用完毕后未及时释放，因此发生内存泄露。

**内存检测结果如下：**



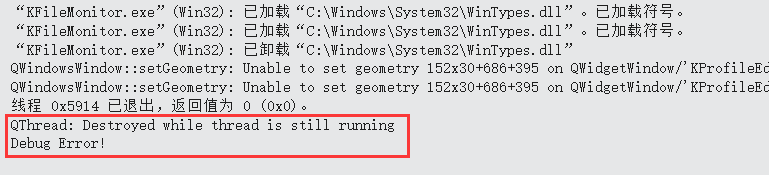
**分析过程：**

找到泄露点位置如下：



可知是new消息队列的时候忘记传入父指针了，导致未能加入析构树

同时在内存测试时发现了一个程序异常，在某次关闭程序时出现了异常如下所示，该异常很难复现



子线程还在运行的时候，将线程对象给析构了

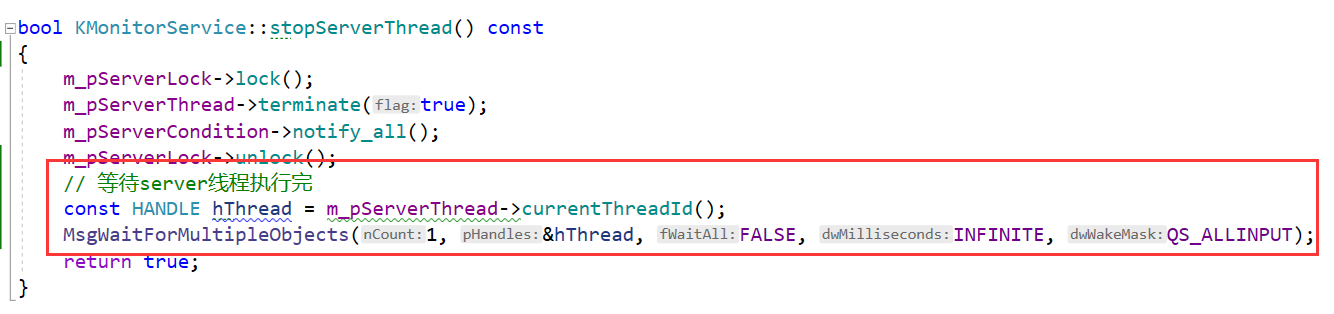
于是我找到对应位置如下



经过分析，该处的问题在于析构的时候，主线程唤醒了阻塞的子线程，但此时线程会同时执行，于是需要在此处让主线程等待子线程执行完成

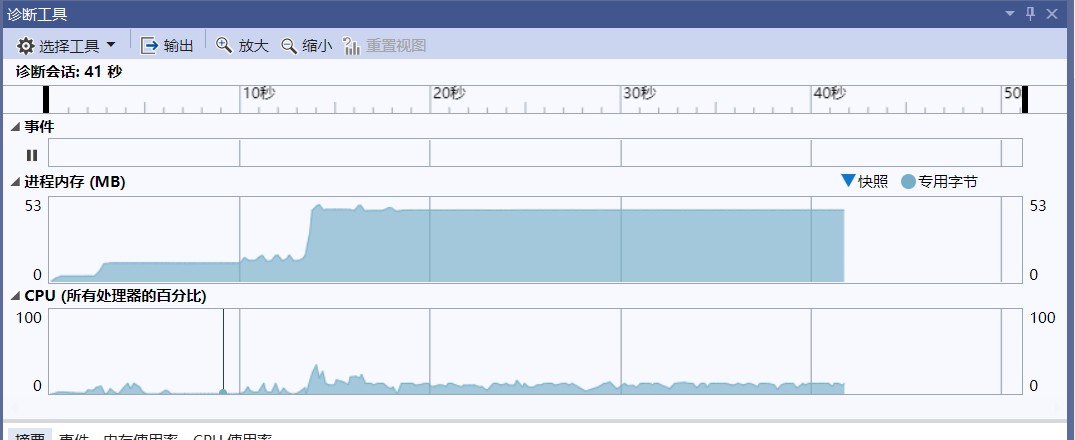
由于线程执行的不可确定性所以问题很难复现

经过查询修改如下将问题解决



**三、VS诊断工具的使用与项目内存分析**

在VS中，可启用自带的诊断工具对**KFileMonitor程序**项目内存占用情况进行监控与分析。



VS诊断工具中项目内存分析

还可以通过内存快照，来截取当前时刻的内存使用情况，并对多个时刻的内存使用情况进行对比与分析。



图4.2 VS诊断工具中项目内存快照

发现当打开文件浏览器时，内存会急剧上升

**四、项目内存优化**

我们可以通过优化查找算法、使用数据压缩、使用内存池等方式对内存进行优化。要注意手动创建的对象内存使用完毕后是否释放，创建的Qt控件是否指定了父窗口，一些临时变量使用完毕后是否可以及时释放，在使用结构体等复杂的数据类型时要考虑字节对齐、字节序问题，通过调整变量声明顺序来优化内存结构。频繁的new和delete很容易导致内存碎片化，内存管理显得很重要。

想法：

尽量使用qt的对象树来进行对象的析构删除

传参时要注意对象的拷贝，必要时可以使用引用或者右值传递，避免开销

使用锁和条件变量管理多线程时要注意锁的释放