### 7.18

今天主要完成了404号系统功能调用的流程以及所涉及的系统资源分析。

之前完成了namespace和cgroups实现的隔离性的学习与掌握，今天开始对系统调用进行隔离性分析。我的任务中第一个系统调用clock\_settime函数，系统调用号为404。在对系统调用的流程分析之前，需要掌握该函数相关的数据结构，包括结构体k\_clock、clock\_posix\_dynamic、timespec等等。此外，在该系统调用中也涉及到其他许多函数，包括clockid\_to\_kclock、get\_timespec64等等。最后大致了解该系统调用所涉及到的数据结构、函数后分析系统调用源码得到实现流程。分析完该函数调用后可知该系统功能调用主要实现设置系统时间秒数与纳秒数。参数分别为：时间类型、timespec类型的时间。时间类型根据定义有：CLOCK\_REALTIME、CLOCK\_MONOTONIC、CLOCK\_PROCESS\_CPUTIME\_ID、CLOCK\_THREAD\_CPUTIME\_ID四种。时间结构内包括秒以及纳秒信息。所以该系统调用主要影响系统的时间，后续需要判断是否已经实现隔离。

### 7.19

今天主要完成了404号系统调用时间信息是否在容器中完成了隔离的尝试性测试。以及部分405号系统调用的流程分析。

测试时首先完成了docker的安装，以及使用方法的初步学习。然后在容器中安装必要的包后，编写测试程序。必要的包有：vim、gcc等。但是在运行测试程序时发现，系统调用404号执行失败，之后经过查询资料发现是设置的时间大小问题，尝试修改后，可以执行成功。该系统调用docker没有实现隔离，此外，在修改时间时还存在锁的竞争。

此外，今天也大致查找到了405号系统调用的源码以及涉及的数据结构及其函数的分析。涉及的数据结构与404号大致相同，函数包括copy\_from\_user：将相应内容拷贝到内核空间；copy\_to\_user：将对应内容拷贝到用户空间；do\_clock\_adjtime：主要函数，目前尚未分析。

### 7.22

今天主要完成了405号系统调用隔离性分析、406号系统调用功能流程和隔离性分析、407号系统调用功能流程分析。

经过分析405号系统调用与404号系统调用涉及的系统资源相似，容器下的隔离性也相似，时间上容器没有实现隔离，同时也存在对timekeeper的锁竞争。

406号系统调用涉及的数据结构与前面大致相同，该函数主要获取对应时钟类型的时间精度，只是对相关宏的读取，所以不存在隔离性问题。

407号系统调用比较复杂，参数也比较多。该系统调用主要提供设置进程纳秒级的休眠时间。由于该函数涉及到的函数较多，隔离性分析还未完成。

7.23

今天主要完成了407号系统调用的隔离性分析，408、409号系统调用的功能、流程、隔离性分析。

在昨天的基础上，今天继续结合源码分许出407号主要设计进程描述符task\_struct结构以及高精度定时器的使用。由于只跟进程相关，所以docker容器间也不存在隔离性问题。

408、409号都是定时器系列的系统调用，408号用于获取定时器剩余时间，409号用于设置定时器时间。这两个系统调用主要涉及定时器的读取与设置，这些内容都存储在一个哈希表中，而存储位置与本进程和定时器ID有关，所以不同容器之间不会发生冲突，这两个系统调用在容器环境下都不存在隔离性问题。