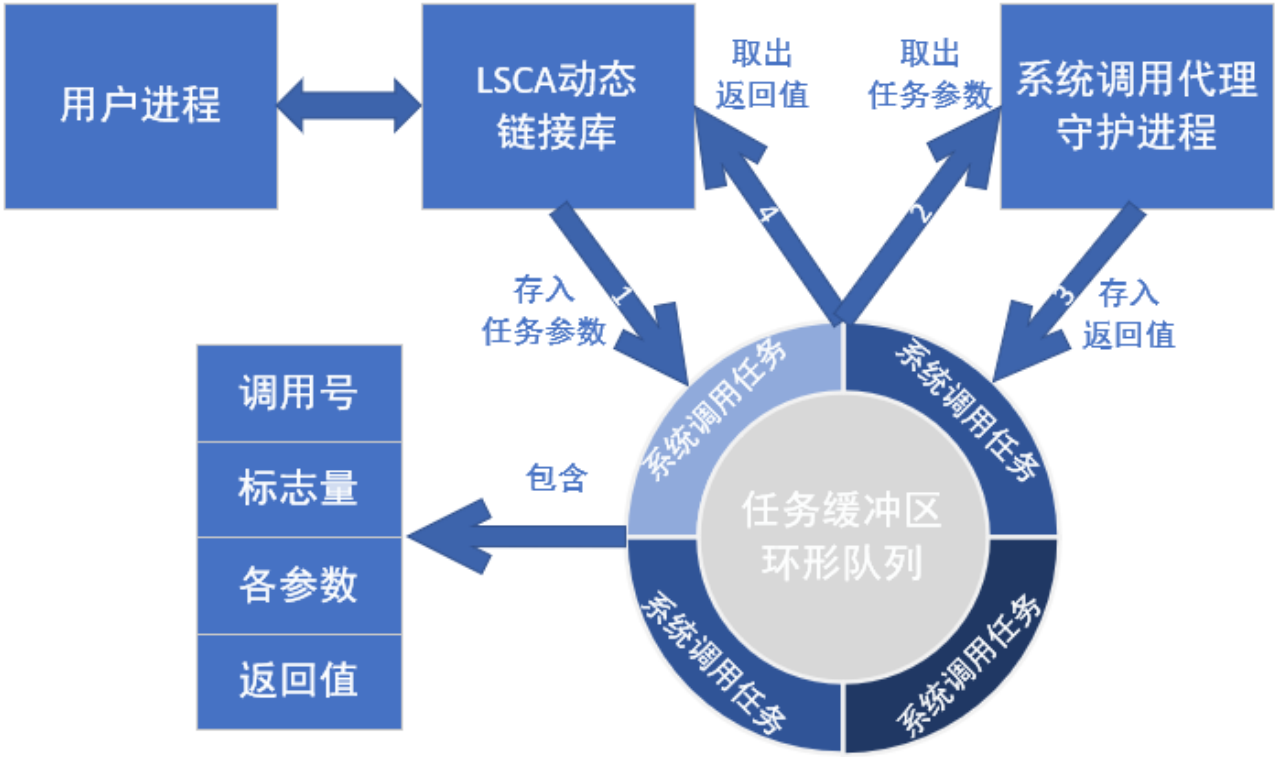
中期报告

项目信息

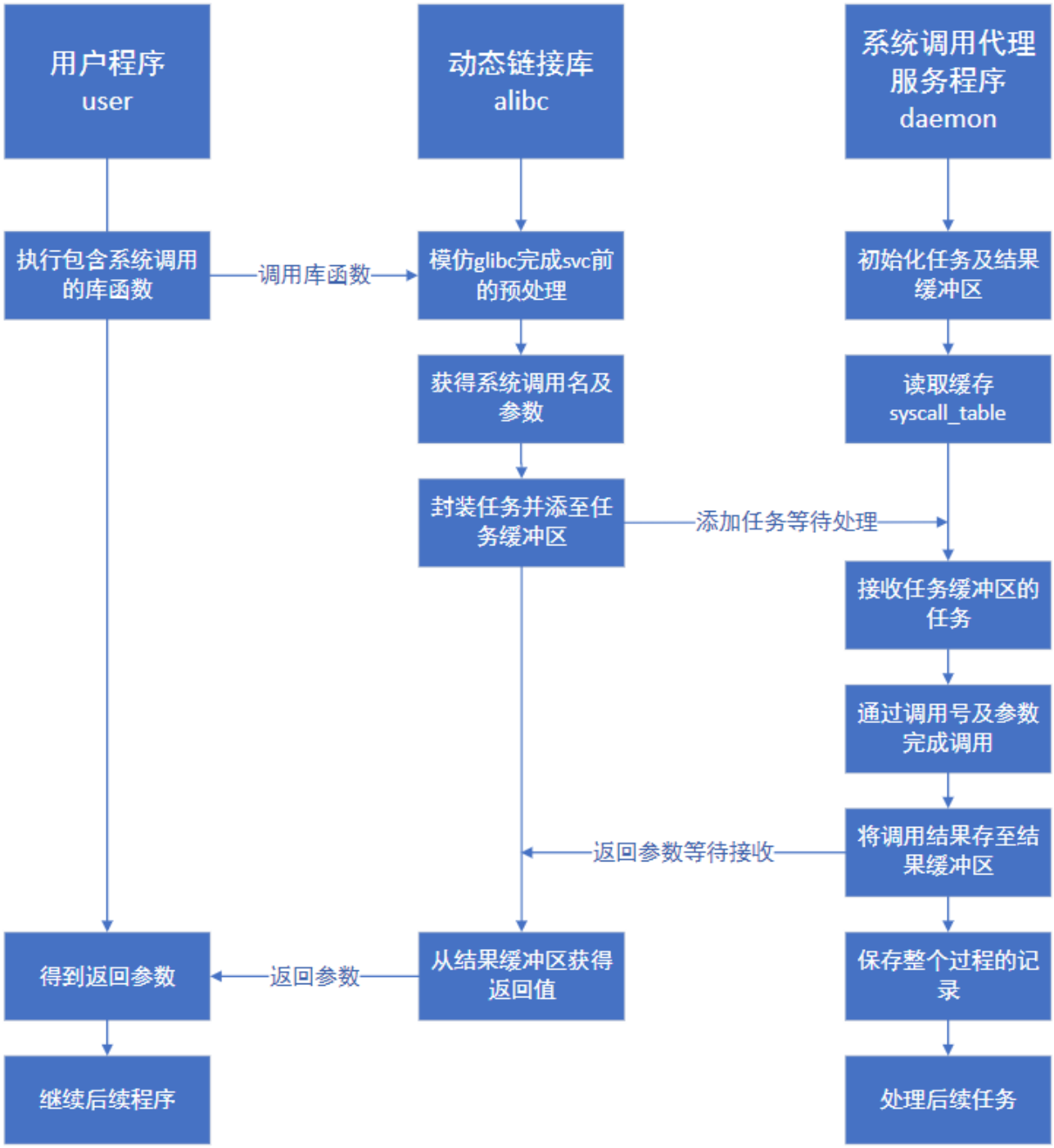
- **项目名称**：LSCA - Linux 系统调用代理
- **方案描述**：方案主要包含的角色有用户程序（user）、lsca动态链接库（agent_lib.so）和lsca系统调用代理服务程序（daemon）。方案执行流程大体如下：

  i. 通过LD_PRELOAD引导用户程序使用LSCA动态链接库替代glibc标准库，使得用户程序将系统调用交由LSCA框架代理执行系统调用，避免陷入内核。
  ii. 动态链接库根据用户程序调用的库函数将参数包装后发往指定的共享内存区（RingBuffer,环形队列缓冲区），等待被daemon代理执行。
  iii. daemon检测到任务，调用相应的服务执行并将结果返回至缓冲区，供用户程序读取。
  iv. 在agent_lib.so帮助下user从缓冲区得到返回值，调用代理流程结束。

结构示意图



流程示意图


用户程序
user
动态链接库
alibc
系统调用代理
服务程序
daemon
执行包含系统调用的库函数
调用库函数
模仿glibc完成svc前的预处理
初始化任务及结果缓冲区
获得系统调用名及参数
读取缓存
syscall_table
封装任务并添至任务缓冲区
添加任务等待处理
接收任务缓冲区的任务
通过调用号及参数完成调用
将调用结果存至结果缓冲区
返回参数等待接收
得到返回参数
返回参数
从结果缓冲区获得返回值
保存整个过程的记录
继续后续程序
处理后续任务

- **时间规划**：时间计划如下表所示。

时间规划表				
序号	具体事项	时间段	进度	备注
第一阶段（demo 验证）				
1	收集相关资料，确认任务细节	7.1-7.7	100%	
2	编写简易代理动态链接库	7.8-7.14	100%	
3	编写简易守护进程	7.15-7.21	100%	
4	调试 App 与 daemon 进程通信	7.22-7.31	100%	
第二阶段（v0.1 版编写）				
5	收集项目所需调用	8.1-8.7	100%	
6	编写完整代理链接库	8.8-8.14	50%	计划改变，先完成整体框架再进行迭代。
7	编写正式 LSCA daemon	8.15-8.21	50%	
8	调试 App 与 daemon 互联	8.22-8.31	50%	
第三阶段（完善）				
9	编写项目要求测试样例	9.1-9.7	50%	
10	根据测试结果调试优化	9.8-9.14		
11	编写使用手册	9.15-9.21		
12	准备结题材料	9.22-9.30		

项目进度

工作安排

整体项目原定计划和进度如上表。原计划编写正式代码时先做完链接库再做代理程序，但是由于两者关系密切，如果分开实现可能隐藏较多问题，导致后期难以调试。因此计划变为先各做一部分，做出一个能够联动的demo，后续不断迭代完善。

工作成果

工作基本按照进度进行，主要分为验证阶段和实现阶段，如下是工作成果的详细描述。

- **验证阶段**
  - 按照计划于鲲鹏服务器&openEuler平台完成了项目所需的关键步骤的验证，包括共享内存通信、socket_server&client等，代码存放在gitlab的test文件夹下。

- **实现阶段**
  - **agent_lib.so**
    编写了简易的agent_lib库，可以将用户程序的系统调用包装并加入到共享内存中并等待返回值。当前只代理write系统调用，后续将连接syscall_wrapper文件夹下的封装函数，支持更多库函数的封装，以代替glibc。
  - **daemon**
    编写了daemon程序，可以完成共享内存的创建以及环形任务队列的维护。参照Linux kernel编写了基于函数指针数组的系统调用代理表。模块化的设计使得daemon可以非常方便地添加系统调用代理服务函数。
  - **Makefile**
    在各个源代码目录分级编写了可以适应文件变化的Makefile，使得开发人员可以方便地增加新的系统调用包装或系统调用服务代码文件。增加功能后无需修改Makefile文件，只需在顶层目录一键make即可。
  - **联调**
    框架编写完成后用write进行了测试，结果显示LSCA可以正常代理测试函数的系统调用，并且将log存放到指定文件。测试文件以及演示视频分别放在gitlab的examples和video目录下。

遇到的问题及解决方案

- **方案设计**
  - 7月的计划主要是熟悉和验证各种项目需要使用的程序，同时构思方案的设计，学习gvisor和Linux kernel的设计。深入了解后发现若要完成理想中的框架工作量大的几乎不可能完成。在与黎亮老师交流后确定了暑期的主要工作。
- **段错误**
  - 在验证阶段熟悉共享内存的使用时，由于共享内存保存的数据结构非常简单因此过程非常顺利。但是建立正式程序所用的环形缓冲区时频繁遇到段错误。并且时而正常时而出错。
  - 于是乎开始用gdb一步步调试，发现在有其他进程写入一个指向某节点的指针的指针时，daemon读取就会发生错误。后来发现是采用的数据结构嵌套的指针太多导致逻辑混乱地将指针指向了进程私有的内存地址空间，导致daemon在访问时遇到段错误。现在修复了这个问题。
- **参数传递**
  - 由于系统调用的发生非常频繁，因此需要非常注重效率。在验证阶段我编写了有十几个参数的程序编译后在反编译观察汇编源代码，了解鲲鹏&openEuler下的参数传递过程，并参照其进行设计。但是很快发现进程间内存空间的隔离导致不能照搬用户程序与内核的交互方法，指针无法正常传递内容。泛化能力（适应多数量、多类型参数的能力）和效率之间存在冲突。
  - 首先是指针传递的问题。传递字符串是很常规的操作，内核可以访问用户进程的内存空间因此可以通过copy_from_user方法直接获取存在用户进程空间中的字符串。但用户进程没有内核的能力，因此在agent_lib.so对用户进程传递的参数进行包装时，增加将指针指向的内存复制到共享空间可以解决这个问题。另外就是用同一个队列节点的数据结构适应各种调用的问题，在于黎亮老师交流后，确定使用序列化和反序列化的方式解决这一问题。具体c语言的序列化、反序列化实现会在后续实现。
- **模块化及自动化编译**
  - 由于LSCA框架需要适应各种库函数和系统调用的代理，因此后期会有不断增加包装和代理服务函数的需求。如何模块化地分割框架，使得在加入新的服务函数时最省事成立就成了一个值得解决的问题。
  - 这里首先参照Linux kernel添加一条系统调用的方式，以系统调用号作为链接daemon和服务函数的桥梁。在daemon中构建一个函数指针数组，并将调用号作为数组下标，连接相应的服务函数。保存各种程序的文件夹也进行了分类。

- 第二个问题就是编译自动化，为了避免用户添加新服务函数时修改makefile文件，因此花时间学习了一下多级makefile和脚本自动处理的功能。使得make时脚本能够适应新加入的源码文件，编译进主程序。