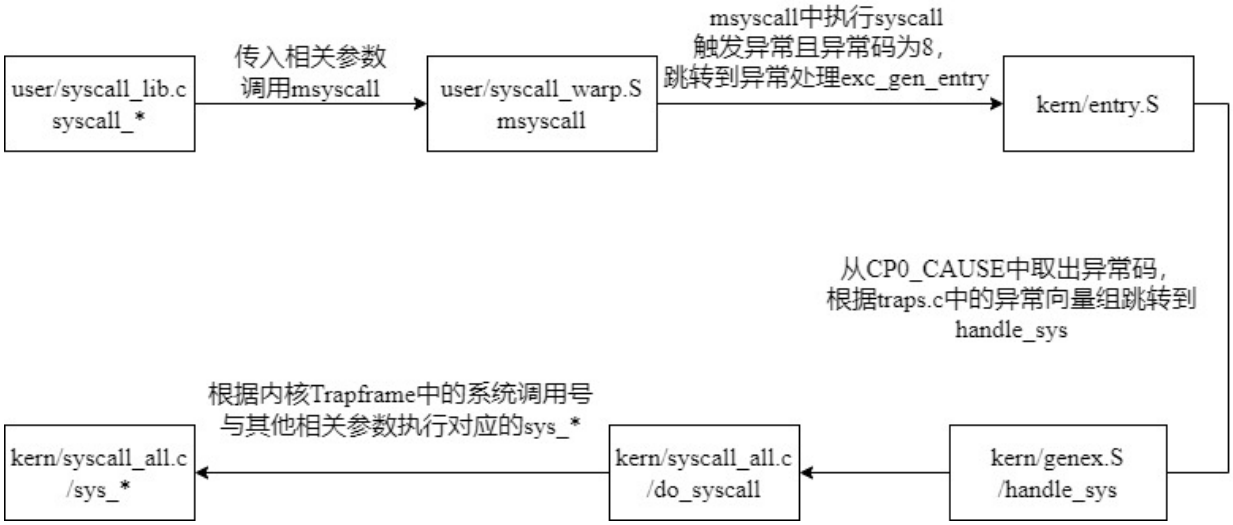
系统调用与IPC

系统调用的流程

实现一个系统调用需要如下完成如下的流程:

1. 在用户的进程中，部分函数需要利用syscall完成相应的功能，调用syscall_*
2. syscall_*将自己的参数传入msyscall，并附带系统调用号
3. msyscall执行syscall,触发8号异常，系统进入内核态
4. 在异常分发过程中，根据异常号结合异常向量组，跳转到handle_sys
5. handle_sys执行对应的解决程序do_syscall.
6. do_syscall根据系统调用号调用对应的sys_*函数完成对应操作并返回



因此。如果需要新增系统调用，则需要修改以下部分：

1. 在user/syscall_lib.c 中新增对应的syscall_*函数，将参数传入msyscall中
2. 在include/syscall.h中添加对应的enum
3. 在syscall_all.c 的syscall_table指定对应的sys_*函数
4. 在syscall_all.c 中实现sys_*函数。

IPC执行的流程

1. 接收方执行syscall_ipc_recv，最终跳转到内核中执行sys_ipc_recv。在sys_ipc_recv中需要完成：

- 检测接收地址合法性
- 设置env_ipc_recving为1，表明自己进入等待接受的状态
- 设置env_ipc_dstva为目标地址
- 将进程控制块的状态调整至ENV_NOT_RUNNABLE，并移出调度队列
- 使用调度函数切换至其他进程，放弃CPU

2. 发送方执行syscall_ipc_try_send，最终跳转到内核中执行sys_ipc_try_send。在sys_ipc_try_send中需要完成：

- 检测发送地址的合法性（如果需要建立映射的话，即srcva不为0）
- 根据env_id获取接收方的进程控制块e，检测e的接受状态(env_ipc_recving)是否为1。
- 设置目标进程控制块的相关信息（value,from,perm）等，并将env_ipc_recving置为0，表示传输结束。
- 设置目标进程控制块状态位RUNNABLE，将其重新放回调度队列中。
- 如果需要建立映射（通过物理页传递数据），则利用page_insert建立映射。