# Project1 添加系统调用

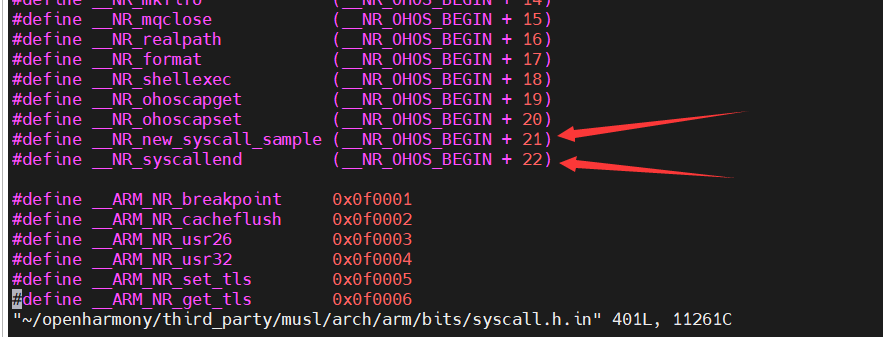
## 姓名：任宇 学号：33920212204567

1. **实验目的**

向鸿蒙Liteos中加入一个自定义的系统调用。

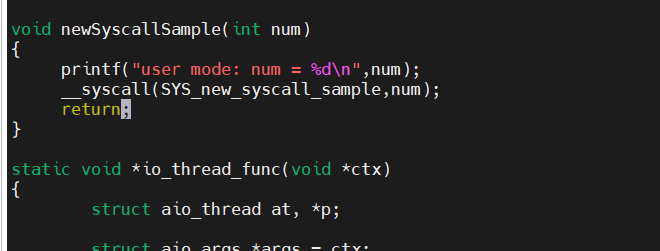
1. **实验环境**
   * 操作系统：
     + - 主机：Windows 10
       - 虚拟机：Ubuntu 18.04
   * 开发板：IMAX6ULL MIN
   * 文件传输工具：FileZilla
   * 终端工具：MobaXterm
2. **实验内容**
3. **在LibC库中确定并添加新增的系统调用号**：

编辑文件openharmony/third\_party/musl/arch/arm/bits/syscall.h.in，如下所示，\_\_NR\_new\_syscall\_sample为新增系统调用号。需要注意同时更新系统调用的编号。



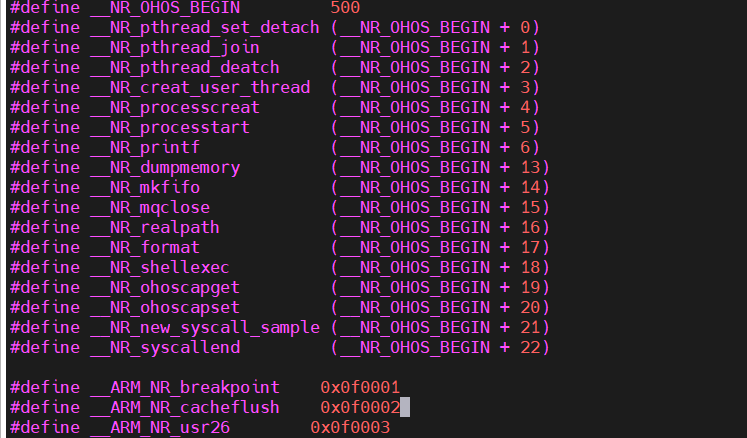
1. **在LibC库中新增用户态的函数接口声明及实现：**

在现成的一个源文件里增加函数实现代码，如在/openharmony/third\_party/musl/src/aio/aio.c文件中增加，如下图所示：

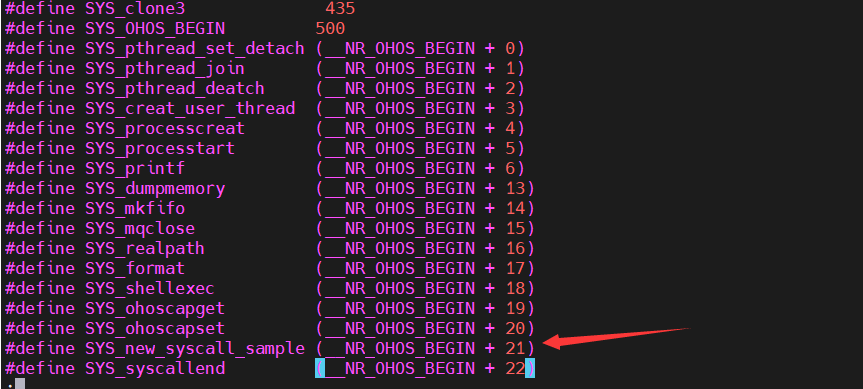


1. **在内核系统调用头文件中新增系统调用号：**

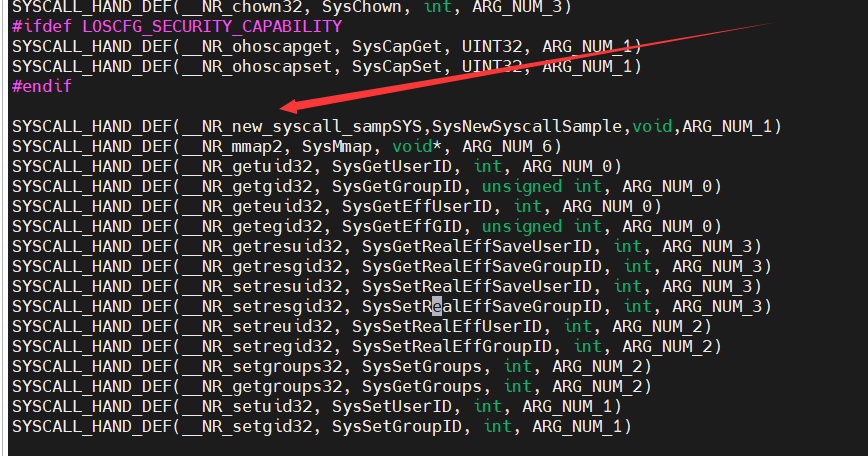
如下所示，在/openharmony/third\_party/musl/kernel/obj/include/bits/syscall.h文件中，\_\_NR\_new\_syscall\_sample为新增系统调用号。用户态代码和内核态代码增加系统调用号方式相同，编号相同。



在/openharmony/prebuilts/lite/sysroot/usr/include/ arm-liteos/bits/syscall.h中做同样修改，如下：

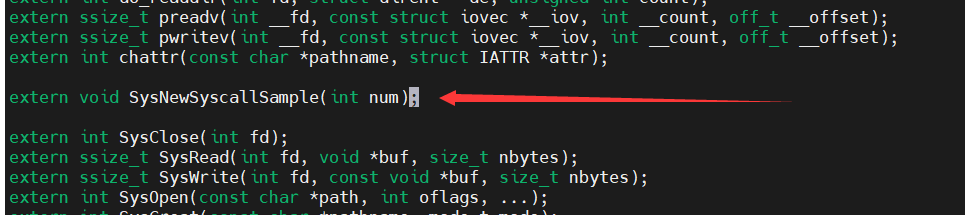


在/openharmony/kernel/liteos\_a/syscall/syscall\_lookup.h中，增加一行，如下。

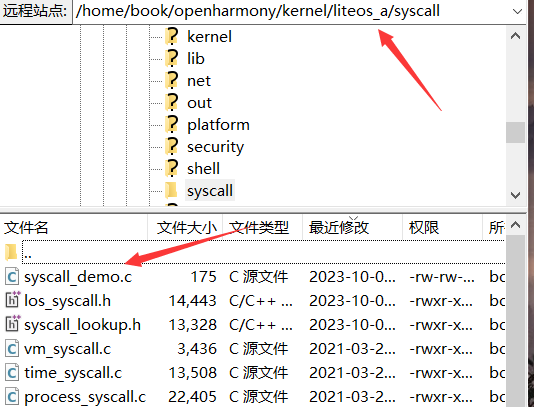


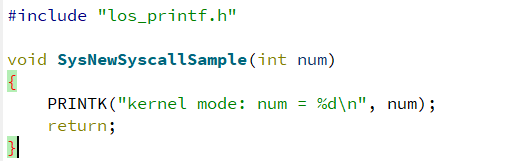
1. **在内核中新增系统调用对应的处理函数：**

需要在内核中新增系统调用函数声明及函数实现，并加入编译构建文件。首先，如下所示，修改/openharmony/kernel/liteos\_a/syscall/los\_syscall.h文件，SysNewSyscallSample为新增系统调用的内核处理函数声明。

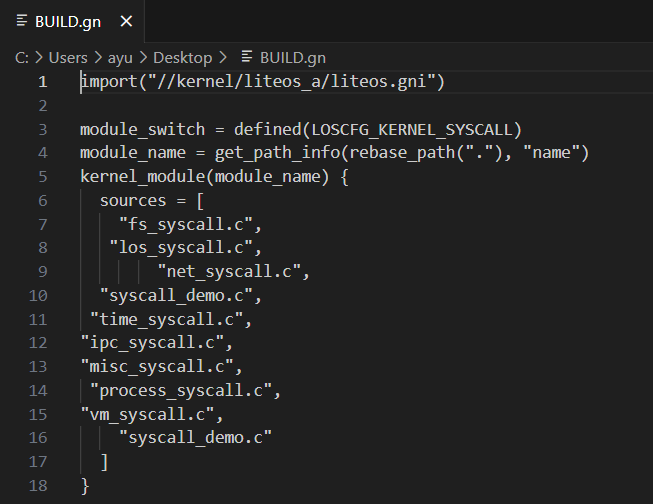


然后在/openharmony/kernel/liteos\_a/syscall目录下新建源文件syscall\_demo.c，新增系统调用的内核处理函数实现。



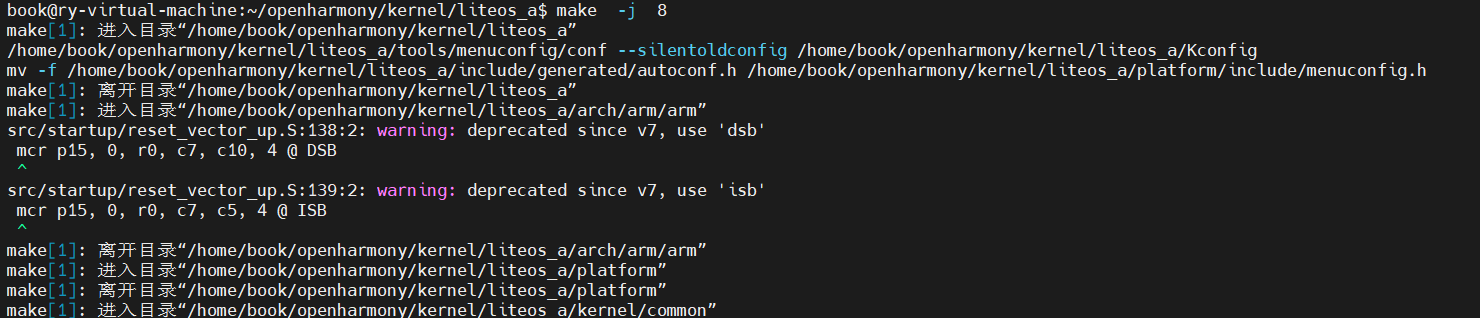


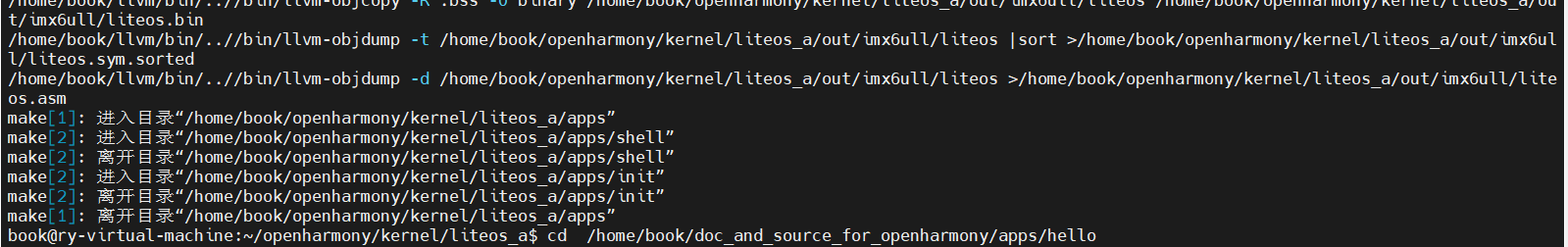
最后，在文件kernel/liteos\_a/syscall/BUILD.gn中增加对syscall\_demo.c源文件的编译管理。



1. **编译内核：**

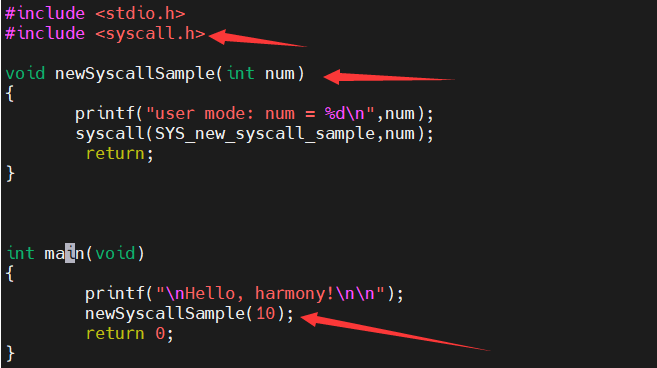
使用make -j 8命令重新编译内核，使新增加的系统调用生效。



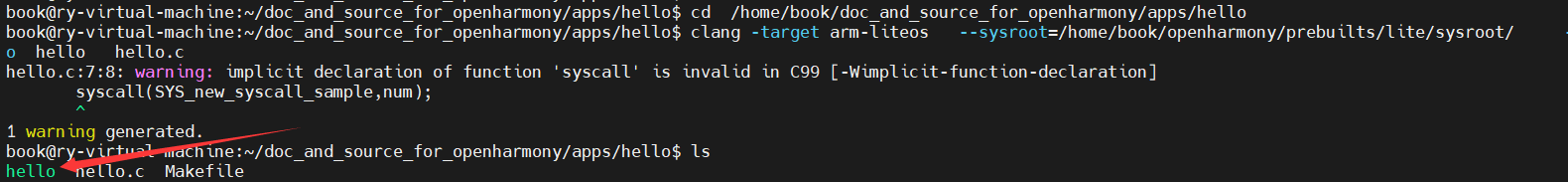


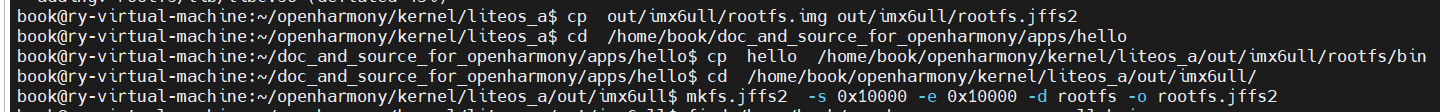
1. **调用并验证：**

对实验二中的hello程序进行修改，使其调用新增的系统调用，如图：

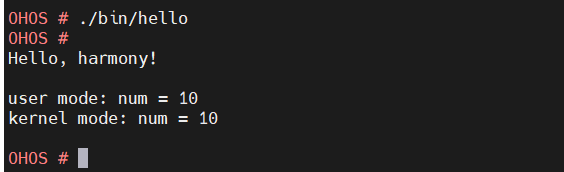


编译hello程序，并将其加载到开发板中：



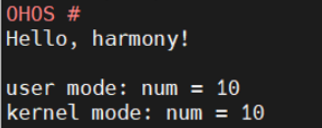


在开发板中运行hello程序，可见程序成功执行系统调用，实验成功：



1. **实验结果**

本实验旨在向鸿蒙Liteos中新增一个系统调用。通过对鸿蒙系统的研究，我确定了在其内核中插入新调用的合适位置。新加入的系统调用的功能是打印一个数字。实验的主要步骤包括设计、编码和测试该调用。在测试环节，新的系统调用成功实现了预期的功能，如下图所示：

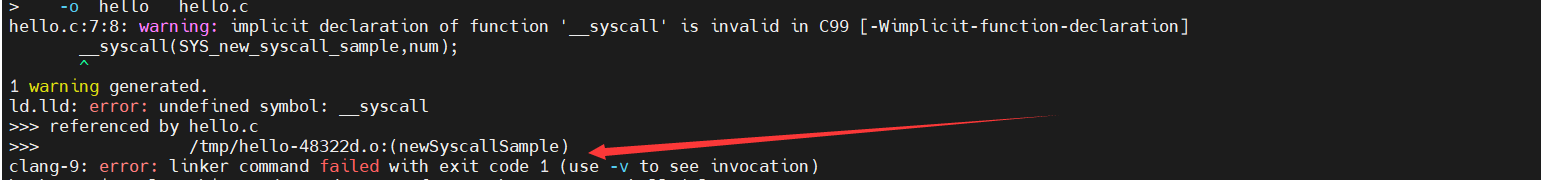


1. **实验分析**

在添加新的系统调用之前，应先了解系统调用， LiteOS\_A内核实现用户态与内核态的区分隔离，用户态程序不能直接访问内核资源，而系统调用则为用户态程序提供了一种访问内核资源、与内核进行交互的通道。

系统调用过程为：调用触发—>上下文切换—>参数传递—>系统调用分发—>执行系统调用—>返回结果—>返回用户空间。因此，需要添加新增的系统调用号，并新增用户态的函数接口声明及实现。接着在内核系统调用头文件中确定并添加新增的系统调用号及对应内核处理函数的声明，并在内核中新增该系统调用对应的内核处理函数。完成以上步骤后，在程序中调用该新增的系统调用，并验证实验是否成功。

在实验过程中，遇到一个问题，即如果在修改内核后不重新编译内核而是直接编译程序，那么就会报错：



解决方法就是先编译内核，再编译程序。

1. **实验总结**

本次实验目标是在鸿蒙操作系统中添加一个新的系统调用。首先，需要在鸿蒙的源代码中定位合适的位置来插入新的系统调用。这一过程需要对Liteos\_a的内部结构有深入的了解，以确保新的系统调用不会影响到其他部分的功能。在代码实现之后，又进行了测试，确保新的系统调用能够正常工作。

总的来说，这次实验增强了我对操作系统工作原理的理解，特别是如何在一个现有的系统中添加新功能。Liteos的灵活性和模块化设计为我学习操作系统提供了宝贵的实践经验。

1. **参考文献**
2. [OpenHarmony LiteOS-A内核文档之学习--系统调用-开源基础软件社区-51CTO.COM](https://ost.51cto.com/posts/8876)
3. **附录**
4. **syscall\_demo.c :**

#include "los\_printf.h"

void **SysNewSyscallSample**(int num)

{

PRINTK("kernel mode: num = %d\n", num);

return;

}

1. **BUILD.gn :**

import("//kernel/liteos\_a/liteos.gni")

module\_switch = defined(LOSCFG\_KERNEL\_SYSCALL)

module\_name = get\_path\_info(rebase\_path("."), "name")

kernel\_module(module\_name) {

sources = [

"fs\_syscall.c",

"los\_syscall.c",

"net\_syscall.c",

"syscall\_demo.c",

"time\_syscall.c",

"ipc\_syscall.c",

"misc\_syscall.c",

"process\_syscall.c",

"vm\_syscall.c",

"syscall\_demo.c"

]

}

1. **hello.c :**

#include <stdio.h>

#include <syscall.h>

void **newSyscallSample**(int num)

{

printf("user mode: num = %d\n",num);

syscall(SYS\_new\_syscall\_sample,num);

return;

}

int **main**(void)

{

printf("\nHello, harmony!\n\n");

newSyscallSample(10);

return 0;

}