项目 1 向鸿蒙 *Liteos* 中加入一个自定义的系统调用

22920212204392 黄勖

1 实验目的

理解系统调用的原理，并向 *LiteOS* 中增加一个自定义的系统调用

2 实验环境

宿主机操作系统：`Windows 10`
虚拟机操作系统：`Ubuntu 18.04.6`
开发板：`IMAX6ULL MINI`
终端工具：`MobaXterm`

3 实验思路

要增加一个自定义的系统调用，首先我们需要新增系统调用号，接下来需要对调用函数进行声明和具体实现，并且新增系统调用号和系统调用函数之间的映射关系，最后对我们编写的内容进行测试。

4 实验内容

4.1 新增系统调用号

第一步需要新增系统调用号，针对的文件有两个：

$/home/book/openharmony/prebuilts/lite/sysroot/usr/include/arm-liteos/bits/syscall.h$

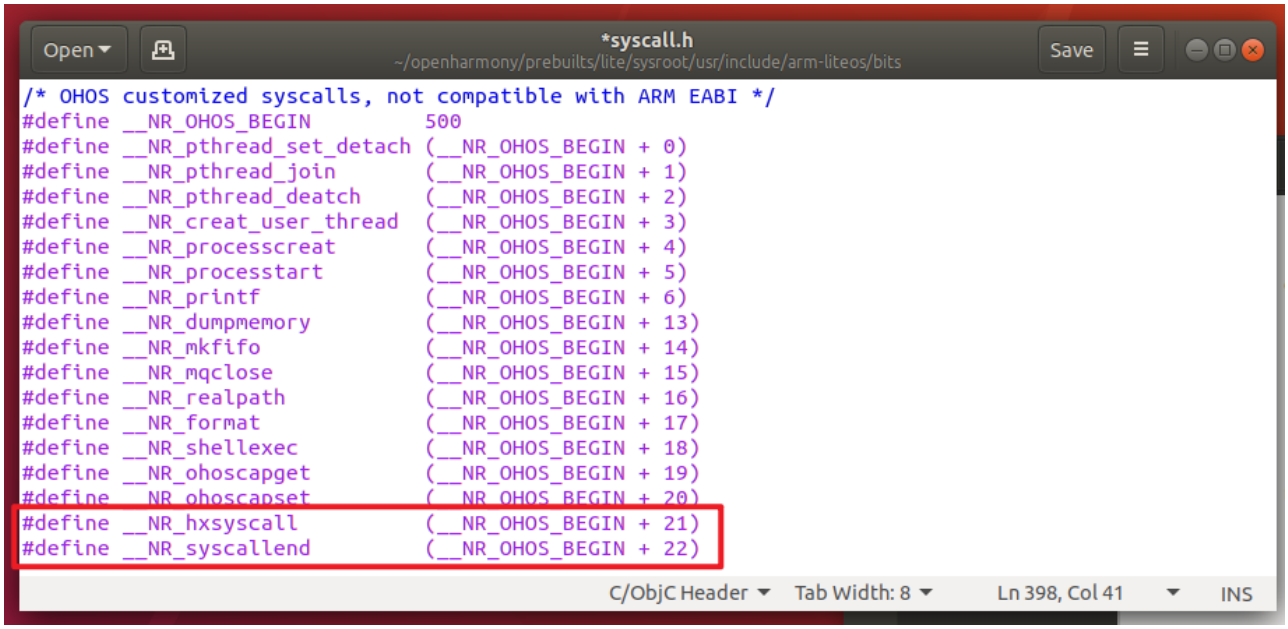
$/home/book/openharmony/third\_party/musl/kernel/obj/include/bits/syscall.h$

以下针对进行修改。

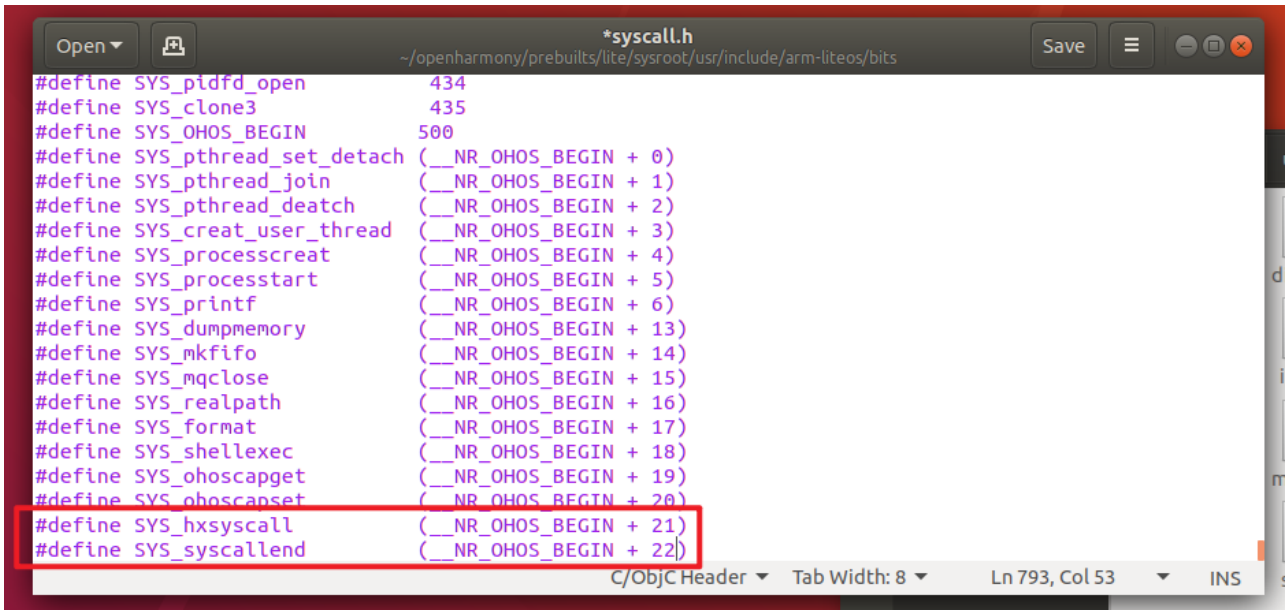
4.1.1 prebuilts/lite/sysroot/usr/include/arm-liteos/bits/syscall.h

在最下方添加用于内核态的系统调用号 `__NR_hxsyscall` 和用于用户态的系统调用号 `SYS_hxsyscall`

同时，需要将 `__NR_syscallend` 数值增加一，用于系统调用号的边界判断



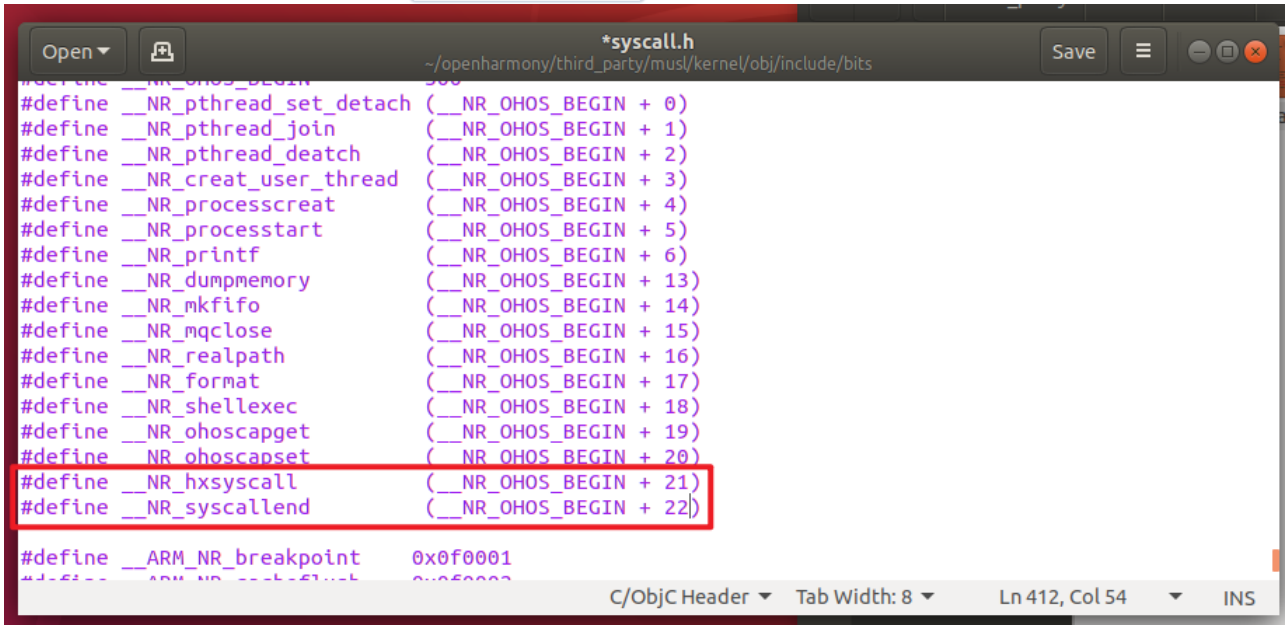
```
/* OHOS customized syscalls, not compatible with ARM EABI */
#define __NR_OHOS_BEGIN          500
#define __NR_pthread_set_detach (__NR_OHOS_BEGIN + 0)
#define __NR_pthread_join       (__NR_OHOS_BEGIN + 1)
#define __NR_pthread_deatch     (__NR_OHOS_BEGIN + 2)
#define __NR_creat_user_thread  (__NR_OHOS_BEGIN + 3)
#define __NR_processcreat       (__NR_OHOS_BEGIN + 4)
#define __NR_processtart        (__NR_OHOS_BEGIN + 5)
#define __NR_printf             (__NR_OHOS_BEGIN + 6)
#define __NR_dumpmemory         (__NR_OHOS_BEGIN + 13)
#define __NR_mkfifo             (__NR_OHOS_BEGIN + 14)
#define __NR_mqclose            (__NR_OHOS_BEGIN + 15)
#define __NR_realpath           (__NR_OHOS_BEGIN + 16)
#define __NR_format             (__NR_OHOS_BEGIN + 17)
#define __NR_shellexec          (__NR_OHOS_BEGIN + 18)
#define __NR_ohoscapget         (__NR_OHOS_BEGIN + 19)
#define __NR_ohoscapset         (__NR_OHOS_BEGIN + 20)
#define __NR_hxsyscall          (__NR_OHOS_BEGIN + 21)
#define __NR_syscallend         (__NR_OHOS_BEGIN + 22)
```



```
#define SYS_pidfd_open          434
#define SYS_clone3              435
#define SYS_OHOS_BEGIN          500
#define SYS_pthread_set_detach (__NR_OHOS_BEGIN + 0)
#define SYS_pthread_join       (__NR_OHOS_BEGIN + 1)
#define SYS_pthread_deatch     (__NR_OHOS_BEGIN + 2)
#define SYS_creat_user_thread  (__NR_OHOS_BEGIN + 3)
#define SYS_processcreat       (__NR_OHOS_BEGIN + 4)
#define SYS_processtart        (__NR_OHOS_BEGIN + 5)
#define SYS_printf             (__NR_OHOS_BEGIN + 6)
#define SYS_dumpmemory         (__NR_OHOS_BEGIN + 13)
#define SYS_mkfifo             (__NR_OHOS_BEGIN + 14)
#define SYS_mqclose            (__NR_OHOS_BEGIN + 15)
#define SYS_realpath           (__NR_OHOS_BEGIN + 16)
#define SYS_format             (__NR_OHOS_BEGIN + 17)
#define SYS_shellexec          (__NR_OHOS_BEGIN + 18)
#define SYS_ohoscapget         (__NR_OHOS_BEGIN + 19)
#define SYS_ohoscapset         (__NR_OHOS_BEGIN + 20)
#define SYS_hxsyscall          (__NR_OHOS_BEGIN + 21)
#define SYS_syscallend         (__NR_OHOS_BEGIN + 22)
```

4.1.2 third_party/musl/kernel/obj/include/bits/syscall.h

添加用于内核态的系统调用号 `__NR_hxsyscall`

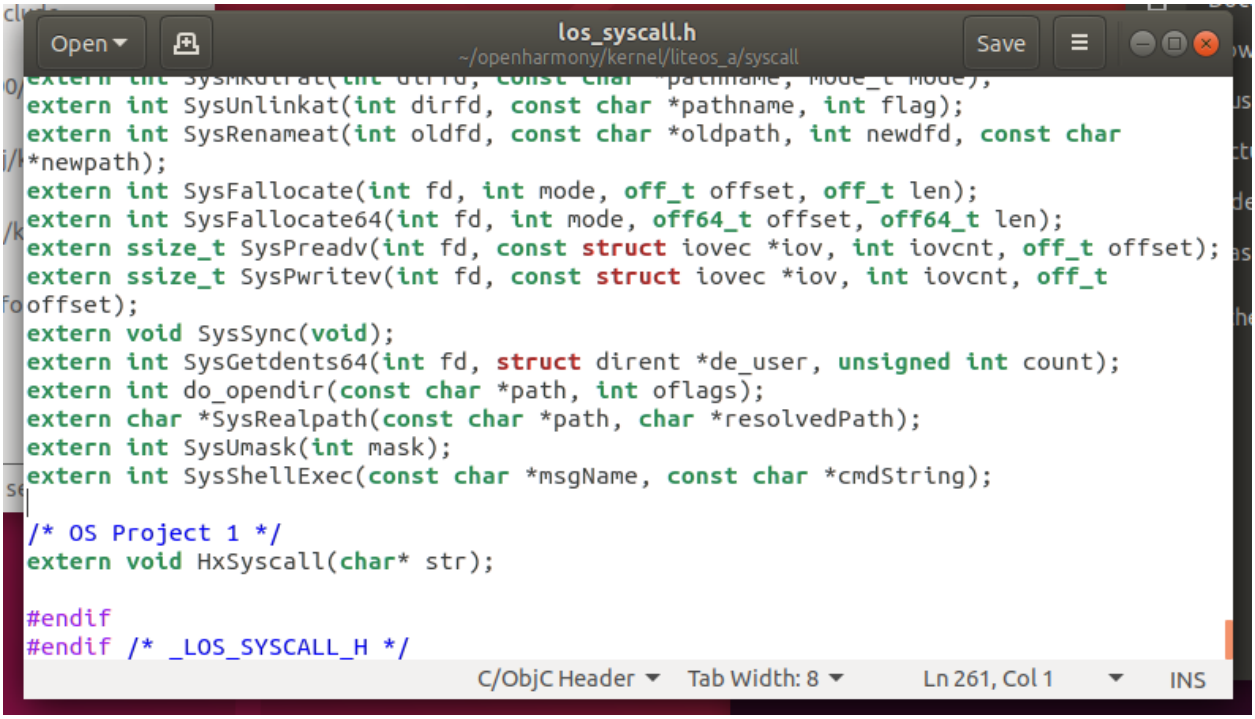


4.2 新增系统调用的函数声明

函数声明文件在:

/home/book/openharmony/kernel/liteos_a/syscall/los_syscall.h

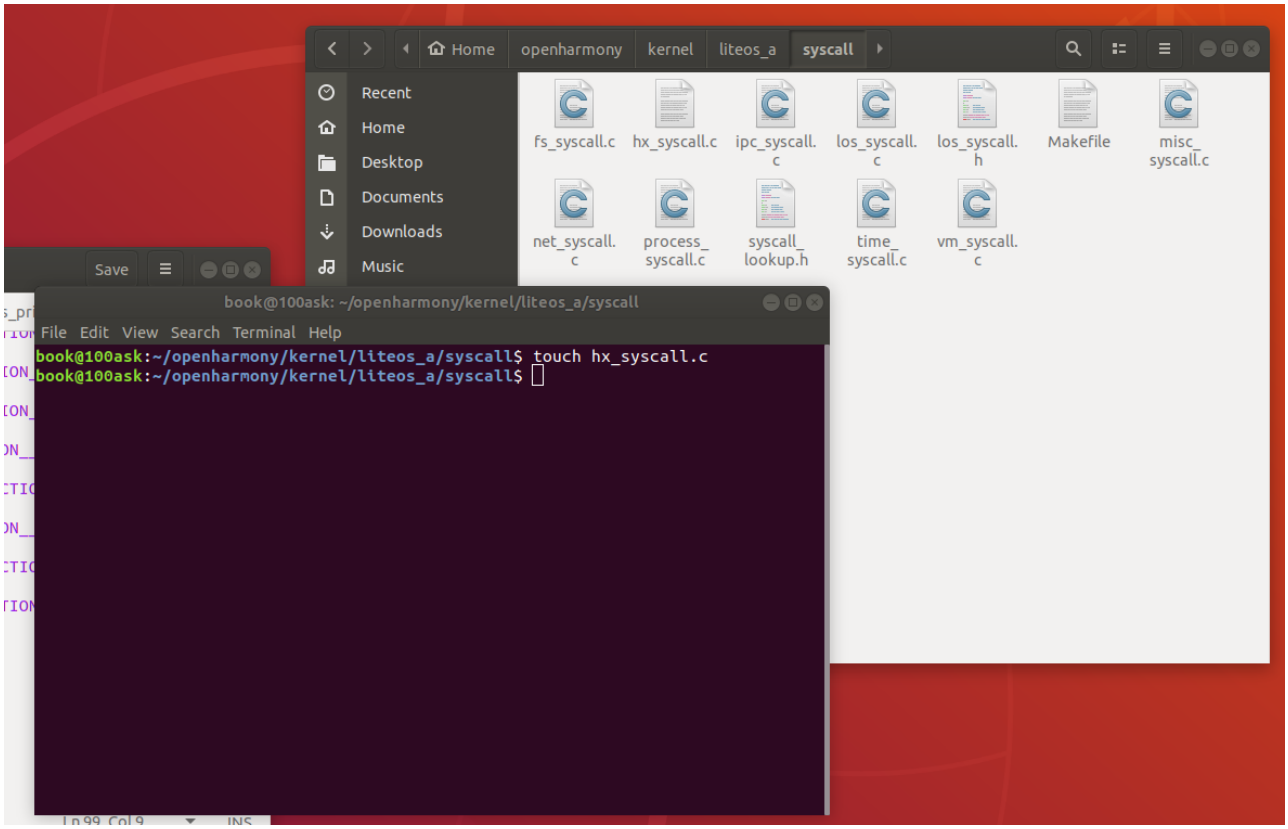
新增系统调用函数 `HxSyscall` 声明。当发生对应的系统调用时，该函数将被执行



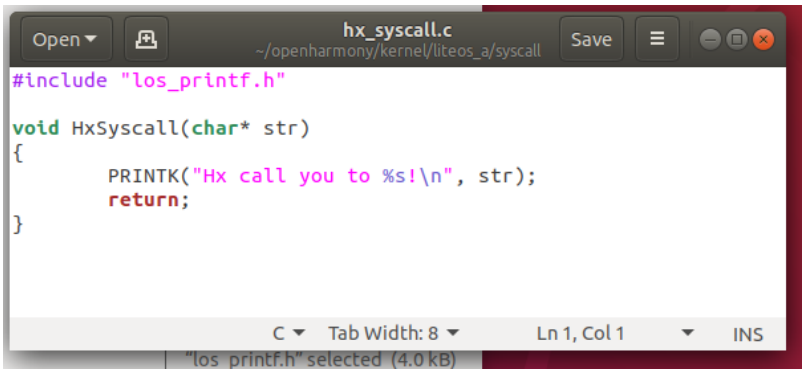
4.3 新增系统调用的函数实现

不难发现，所有在 `los_suscall.h` 文件中使用 `extern` 修饰的函数声明，其实现都定义在同级目录下的 `.c` 文件中

因此，可以在`los_syscall.c`文件的同级目录下，新建一个 `hx_syscall.c` 文件，并在其中定义 `HxSyscall` 函数的实现



引入 `los_printf.h` 头文件，调用其中的 `PRINTK` 函数来实现字符串的输出



```
Open ▾   los_printf.h   ~/openharmony/kernel/liteos_a/kernel/include   Save
*hx_syscall.c   los_printf.h
#define PRINT_DEBUG(fmt, args...)      LOS_LkPrint(LOS_DEBUG_LEVEL, __FUNCTION__, __LINE__, fmt,
##args)
#define PRINT_INFO(fmt, args...)       LOS_LkPrint(LOS_INFO_LEVEL, __FUNCTION__, __LINE__, fmt,
##args)
#define PRINT_WARN(fmt, args...)       LOS_LkPrint(LOS_WARN_LEVEL, __FUNCTION__, __LINE__, fmt,
##args)
#define PRINT_ERR(fmt, args...)        LOS_LkPrint(LOS_ERR_LEVEL, __FUNCTION__, __LINE__, fmt,
##args)
#define PRINTK(fmt, args...)           LOS_LkPrint(LOS_COMMON_LEVEL, __FUNCTION__, __LINE__, fmt,
##args)
#define PRINT_EMG(fmt, args...)        LOS_LkPrint(LOS_EMG_LEVEL, __FUNCTION__, __LINE__, fmt,
##args)
#define PRINT_RELEASE(fmt, args...)    LOS_LkPrint(LOS_COMMON_LEVEL, __FUNCTION__, __LINE__, fmt,
##args)
#define PRINT_TRACE(fmt, args...)      LOS_LkPrint(LOS_TRACE_LEVEL, __FUNCTION__, __LINE__, fmt,
##args)

typedef enum {
    NO_OUTPUT = 0,
    UART_OUTPUT = 1,
    CONSOLE_OUTPUT = 2,
    EXC_OUTPUT = 3
} OutputType;

extern VOID OsVprintf(const CHAR *fmt, va_list ap, OutputType type);
C/ObjC Header ▾   Tab Width: 8 ▾   Ln 99, Col 9 ▾   INS
```

4.4 新增系统调用号和系统调用函数的映射关系

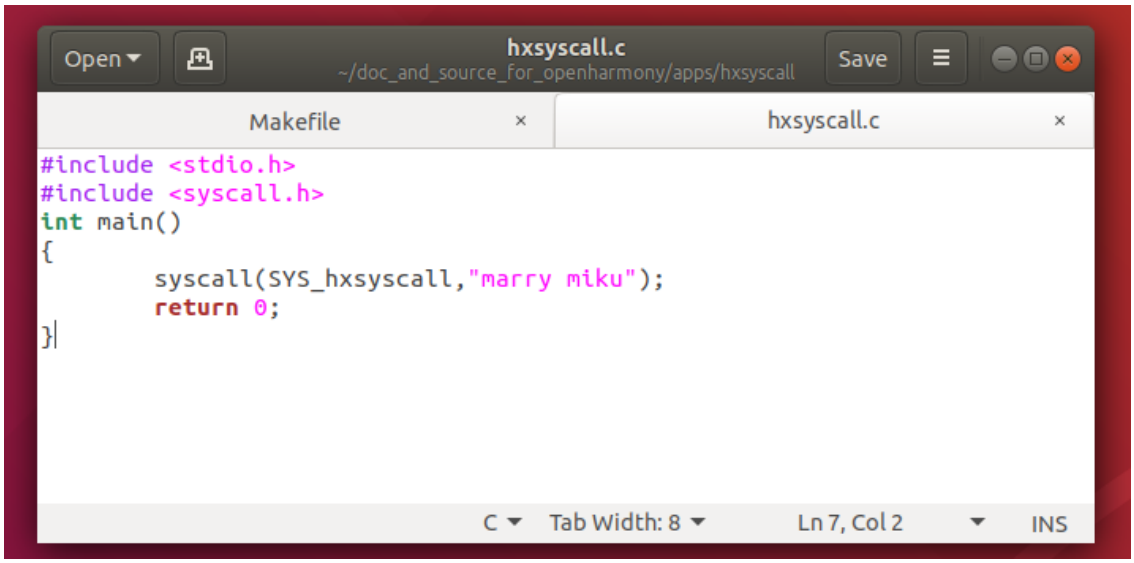
针对文件：*/home/book/openharmony/kernel/liteos_a/syscall/syscall_lookup.h*

新增系统调用号 `__NR_hxsyscall` 和系统调用函数 `HxSyscall` 之间的映射关系，返回值类型为 `void`，参数个数为 `1`

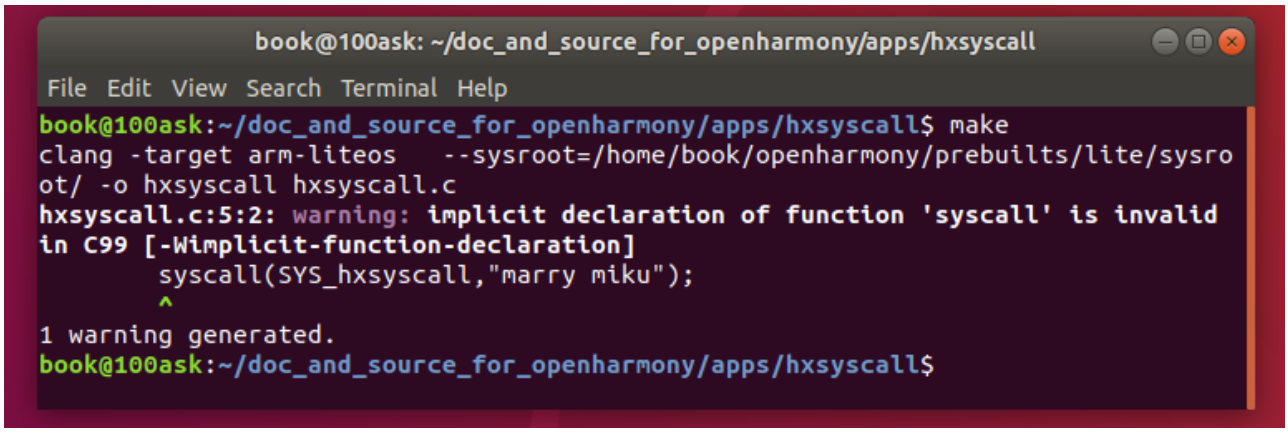
```
Open ▾   *syscall_lookup.h   ~/openharmony/kernel/liteos_a/syscall   Save
SYSCALL_HAND_DEF(__NR_recvfrom, SysRecvFrom, ssize_t, ARG_NUM_6)
SYSCALL_HAND_DEF(__NR_shutdown, SysShutdown, int, ARG_NUM_2)
SYSCALL_HAND_DEF(__NR_setsockopt, SysSetSockOpt, int, ARG_NUM_5)
SYSCALL_HAND_DEF(__NR_getsockopt, SysGetSockOpt, int, ARG_NUM_5)
SYSCALL_HAND_DEF(__NR_sendmsg, SysSendMsg, ssize_t, ARG_NUM_3)
SYSCALL_HAND_DEF(__NR_recvmsg, SysRecvMsg, ssize_t, ARG_NUM_3)
#endif

SYSCALL_HAND_DEF(__NR_shmat, SysShmAt, void *, ARG_NUM_3)
SYSCALL_HAND_DEF(__NR_shmdt, SysShmDt, int, ARG_NUM_1)
SYSCALL_HAND_DEF(__NR_shmget, SysShmGet, int, ARG_NUM_3)
SYSCALL_HAND_DEF(__NR_shmctl, SysShmCtl, int, ARG_NUM_3)

SYSCALL_HAND_DEF(__NR_statx, SysStatx, int, ARG_NUM_5)
SYSCALL_HAND_DEF(403, SysClockGettime64, int, ARG_NUM_2)
SYSCALL_HAND_DEF(404, SysClockSettime64, int, ARG_NUM_2)
SYSCALL_HAND_DEF(406, SysClockGetres64, int, ARG_NUM_2)
SYSCALL_HAND_DEF(407, SysClockNanoSleep64, int, ARG_NUM_4)
SYSCALL_HAND_DEF(408, SysTimerGettime64, int, ARG_NUM_2)
SYSCALL_HAND_DEF(409, SysTimerSettime64, int, ARG_NUM_4)

/* LiteOS customized syscalls, not compatible with ARM EABI */
SYSCALL_HAND_DEF(__NR_pthread_set_detach, SysUserThreadSetDeatch, int, ARG_NUM_1)
SYSCALL_HAND_DEF(__NR_pthread_join, SysThreadJoin, int, ARG_NUM_1)
SYSCALL_HAND_DEF(__NR_pthread_deatch, SysUserThreadDetach, int, ARG_NUM_1)
SYSCALL_HAND_DEF(__NR_creat_user_thread, SysCreateUserThread, unsigned int, ARG_NUM_3)
SYSCALL_HAND_DEF(__NR_hxsyscall, HxSyscall, void, ARG_NUM_1)
C/ObjC Header ▾   Tab Width: 8 ▾   Ln 240, Col 61 ▾   INS
```

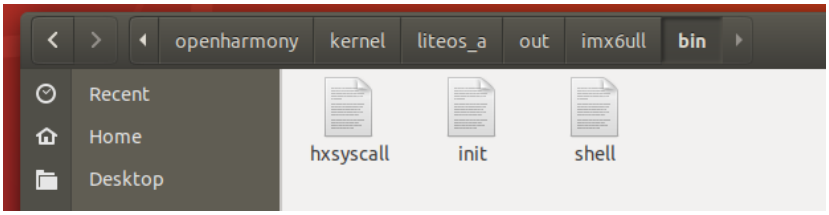
4.5 编译内核与编写测试

编写 `hxsyscall.c` 文件，直接在 `main` 函数中执行调用号为 `SYS_hxsyscall` 的系统调用



对 `hxsyscall.c` 文件进行编译，然后复制到 `rootfs` 目录下的 `bin` 目录中，重新制作 `rootfs.jffs2`





重新编译和烧录

```
book@100ask: ~/openharmony/kernel/liteos_a
File  Edit  View  Search  Terminal  Help
book@100ask:~/openharmony/kernel/liteos_a$ make -j 8
make[1]: Entering directory '/home/book/openharmony/kernel/liteos_a/arch/arm/arm
'
make[1]: Nothing to be done for 'all'.
make[1]: Leaving directory '/home/book/openharmony/kernel/liteos_a/arch/arm/arm'
make[1]: Entering directory '/home/book/openharmony/kernel/liteos_a/platform'
make[1]: Leaving directory '/home/book/openharmony/kernel/liteos_a/platform'
make[1]: Entering directory '/home/book/openharmony/kernel/liteos_a/kernel/commo
n'
make[1]: Nothing to be done for 'all'.
make[1]: Leaving directory '/home/book/openharmony/kernel/liteos_a/kernel/common
```

```
book@100ask:~/openharmony/kernel/liteos_a/out/imx6ull$ mkfs.jffs2  -s 0x10000 -e
 0x10000 -d rootfs -o rootfs.jffs2
book@100ask:~/openharmony/kernel/liteos_a/out/imx6ull$
```

5 实验结果

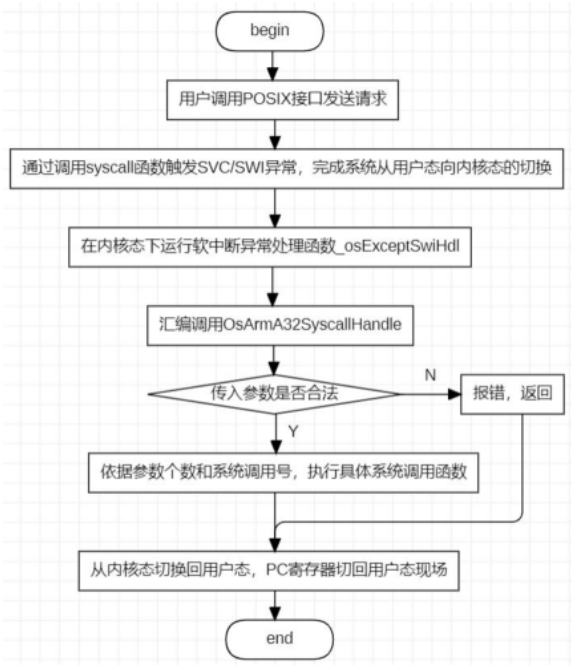
在开发板中执行 `hxsyscall` 程序，成功打印 `Hx call you to marry miku!` ，测试成功

```
COM5  (USB-Enhanced-SERIAL CH9102 (COM5)) (1)
Terminal  Sessions  View  X server  Tools  Games  Settings  Macros  Help
Session  Servers  Tools  Games  Sessions  View  Split  MultiExec  Tunneling  Packages  Settings  Help  X server  Exit
Quick connect...
sessions
COM3  (蓝牙链接上的标准串行 (COM3))
COM4  (蓝牙链接上的标准串行 (COM4))
COM5  (USB-Enhanced-SERIAL CH9102 (COM
Serial (COM)
2. COM5  (USB-Enhanced-SERIAL CH
DeviceManagerStart end ...
[ERR]No console dev used.
[ERR]No console dev used.
OHOS # ls
Directory /:
drwxr-xr-x 0        u:0     g:0     dev
dr-xr-xr-x 0        u:0     g:0     proc
drwxrwxr-x 0        u:1001  g:1001  etc
drwxrwxr-x 0        u:1001  g:1001  bin
drwxrwxr-x 0        u:1001  g:1001  app
drwxrwxr-x 0        u:1001  g:1001  lib
drwxrwxr-x 0        u:1001  g:1001  usr
drwxrwxr-x 0        u:1001  g:1001  data
drwxrwxr-x 0        u:1001  g:1001  system
OHOS # cd ./bin
OHOS # ls
Directory /bin:
-rwxrwxr-x 13900    u:1001  g:1001  init
-rwxrwxr-x 26236    u:1001  g:1001  shell
-rwxrwxr-x 15088    u:1001  g:1001  hxsyscall
OHOS # hxsyscall
hxsyscall:command not found
OHOS # ./hxsyscall
OHOS # Hx call you to marry miku!
UNREGISTERED VERSION  -  Please support MobaXterm by subscribing to the professional edition here:  https://mobaxterm.mobatek.net
```

6 实验分析

本次实验实现了新增一个系统调用，大概分为新增系统调用号、新增系统调用的函数声明、新增系统调用的函数实现和新增系统调用号和系统调用函数的映射关系四个步骤，最后我们编译测试。

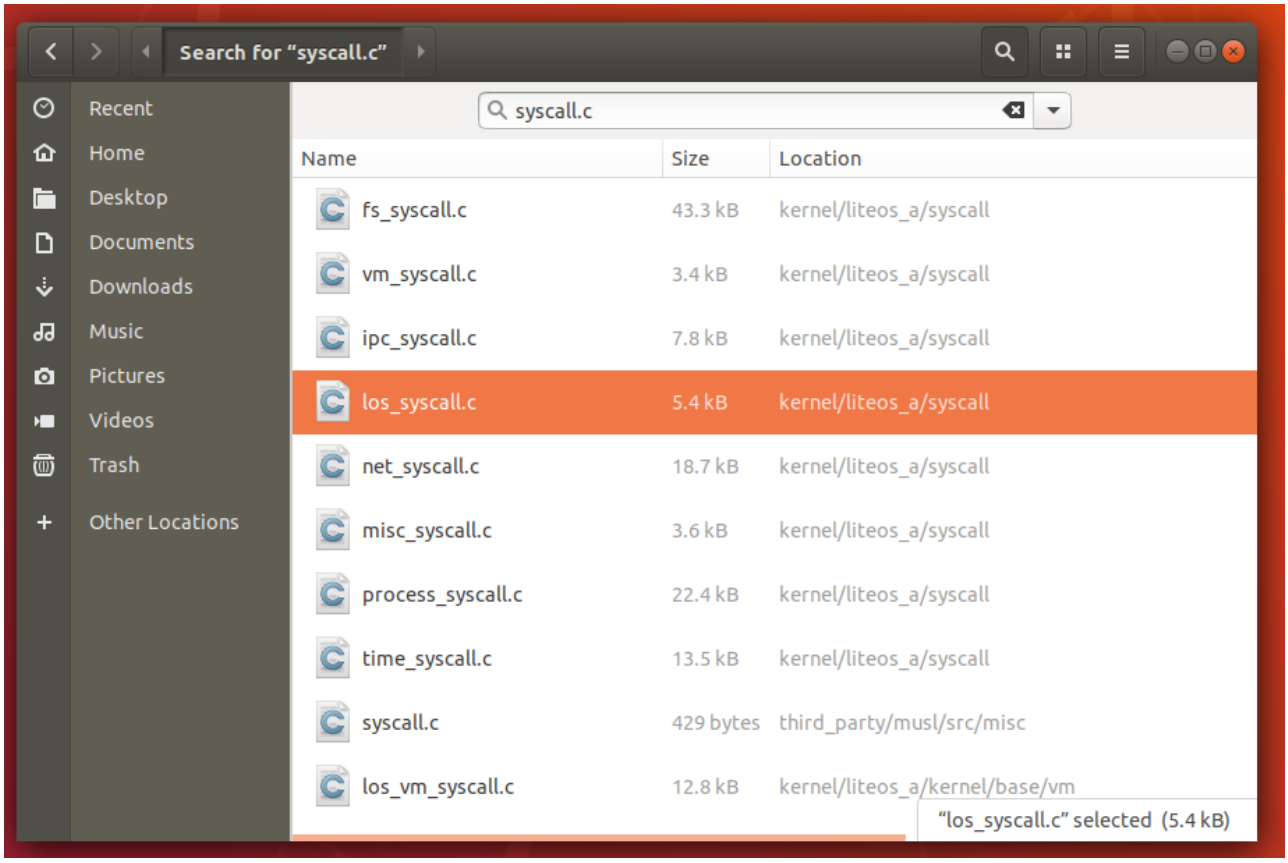
系统调用的一般流程为：用户态 `main` 函数中调用用户态函数、用户态函数调用 `syscall`，将相关信息寄存在用户栈中、触发 SVC 异常切换换到内核态、内核态中首先执行软中断异常处理函数 `_osExceptSwiHdl` 、 调 用 `OsArmA32SyscallHandle` 完成调用、恢复现场回到用户态。



系统启动时，将syscall_lookup.h文件中定义的映射关系注册到全局变量中进行维护，发生系统调用时，直接使用系统调用号访问`g_syscallHandle`查找执行函数.

6.1 验证 `__NR_syscallend` 边界判断的作用

在 `los_syscall.c` 文件中可以看到。




```
……
#define SYS_CALL_NUM    (__NR_syscallend + 1) //总调用数
……
/* The SYSCALL ID is in R7 on entry.  Parameters follow in R0..R6 */
LITE_OS_SEC_TEXT UINT32 *OsArmA32SyscallHandle(UINT32 *regs)
{
    ……

    if (cmd >= SYS_CALL_NUM) {
        // 判断边界
        PRINT_ERR("Syscall ID: error %d !!!\n", cmd);
        return regs;
    }
    ……
}
```

6.2 找不到 `BUILD.gn`

在系统调用处理函数时 `los_syscall.h` 中使用了 `extern` 修饰这些函数，在链接的时候由链接器来实现函数关联；由于我采用自己新写了 `.c` 文件，如果没有 `BUILD.gn` 则无法添加到对应的编译链中。在这里为了简便我采用了在 `hx_syscall.c` 中加入 `#include "los_syscall.h"` 来解决，还可以通过添加该文件来解决。

7 实验总结

本次实验中，我学会了如何在LiteOS中新增自定义系统调用，进一步了解了系统调用的执行过程和实现原理，加深了对操作系统的理解

8 参考文献

1. OpenHarmony LiteOS-A内核文档之学习--系统调用
2. 鸿蒙OS的系统调用是如何实现的？

9 附录

```c
/* hxsyscall.c in 'kernel/liteos_a/syscall' */
#include "los_printf.h"
#include "los_syscall.h"
void HxSyscall(char* str)
{
    PRINTK("Hx call you to %s!\n", str);
    return;
}


/* hxsyscall.c in apps */
#include <stdio.h>
#include <syscall.h>
int main()
{
    syscall(SYS_hxsyscall,"marry miku");
    return 0;
}
```