

实验案例三：内核子系统—启动流程与系统调用(指导文档)

实验案例三：内核子系统—启动流程与系统调用(指导文档)

- 一、实验简介
- 二、实验内容步骤
 - 1.内核启动
 - 2. 添加系统调用

一、实验简介

OpenHarmony 采用多内核设计，目前支持 **Linux**、**LiteOS-m** 与 **LiteOS-a** 内核。开发者可根据设备需求和资源情况选择不同内核实现产品。所有内核均使用 **KAL (Kernel Abstraction Layer)** 模块，通过屏蔽内核差异向上提供统一接口，从而降低开发难度。

LiteOS 是华为面向物联网（IoT）领域开发的轻量级操作系统，主要应用于智能家居、个人穿戴、车联网、城市公共服务及制造业等场景。LiteOS 具有高实时性、高稳定性和低功耗特点，基础内核可裁剪至不到 10 KB，适合部署在小型设备中。因此，LiteOS 成为 OpenHarmony 在轻量 and 小型系统中的主要内核选择。

本节实验旨在通过对内核进行基础修改，增加对 OpenHarmony 支持的内核有初步了解。考虑到 Linux 内核相关资料较为完备，本次实验仅聚焦 **LiteOS** 内核。LiteOS-a 与 LiteOS-m 系统结构相似，仅在面向不同应用层级时有少量差异。本实验及后续实验均选用 **LiteOS-a** 内核，其功能更完善，并且支持 MMU 等特性，更贴近现代操作系统功能。

二、实验内容步骤

本次实验需要依次完成下列实验要求，并提交在qemu中运行的结果截图。

1.内核启动

本节实验需要在OpenHarmony的内核子系统的LiteOS-A内核启动框架中注册新模块，并在新模块中打印信息。具体来说，需要分别在 `LOS_INIT_LEVEL_EARLIEST`、`LOS_INIT_LEVEL_KMOD_PREVM`、`LOS_INIT_LEVEL_KMOD_EXTENDED` 中注册新模块，并打印简单的信息。最终运行 `qemu_run` 运行结果如下：

```
Enter to start qemu[y/n]:y
Waiting VNC connection on: 5920 ...(Ctrl-C exit)
初始化: LOS_INIT_LEVEL_EARLIEST

*****welcome*****

Processor   : Cortex-A7
Run Mode    : UP
GIC Rev     : GICv2
build time  : Feb 24 2024 20:42:22
Kernel      : Huawei LiteOS 2.0.0.37/debug

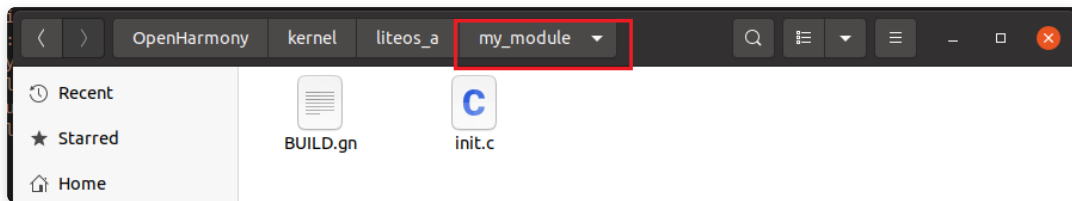
*****
```

```
main core booting up...
初始化: LOS_INIT_LEVEL_KMOD_PREVM
初始化: LOS_INIT_LEVEL_KMOD_EXTENDED
cpu 0 entering scheduler
mem dev init ...
DevMmzRegister...
Date:Feb 24 2024.
Time:20:42:21.
```

流程

本节任务主要需要为修改LiteOS-a内核，在内核的启动框架之中进行模块的注册，本文档选择的实现方法是直接在liteos内核中添加新的模块，具体流程如下：

1. 在 LiteOS-A 内核目录下创建新的文件夹 `my_module`，用于存放即将在内核中新增的功能模块。



其中init.c主要使用 `LOS_MODULE_INIT` 宏在内核中启动流程中注册新模块，打印相关信息,其内容如下：

```
//init.c
/* 内核启动框架头文件 */
#include "los_init.h"
#include "los_printf.h"

/* 新增模块的初始化函数 */
void OsEarliestModInit(void)
{
    PRINTK("初始化: LOS_INIT_LEVEL_EARLIEST\n");
}

void OsKmodPrevModInit(void)
{
    PRINTK("初始化: LOS_INIT_LEVEL_KMOD_PREVM\n");
}

void OsKmodExtendedModInit(void)
{
    PRINTK("初始化: LOS_INIT_LEVEL_KMOD_EXTENDED\n");
}

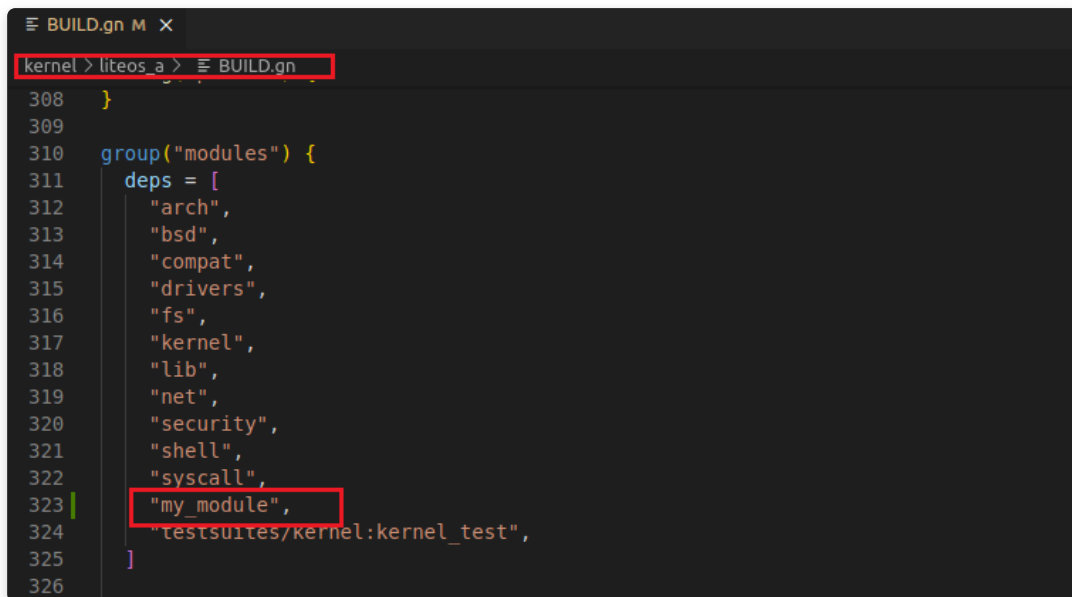
/* 在启动框架的目标层级中注册新增模块 */
LOS_MODULE_INIT(OsEarliestModInit, LOS_INIT_LEVEL_EARLIEST);
LOS_MODULE_INIT(OsKmodPrevModInit, LOS_INIT_LEVEL_KMOD_PREVM);
LOS_MODULE_INIT(OsKmodExtendedModInit, LOS_INIT_LEVEL_KMOD_EXTENDED);
```

而BUILD.gn文件中则简单的配置c文件的编译选项等信息，内容如下：

```
import("//kernel/liteos_a/liteos.gni")

module_name = get_path_info(rebase_path("."), "name")
kernel_module(module_name) {
    sources = [
        "init.c",
    ]
}
```

即使在 LiteOS 内核中新增加了上述文件，OpenHarmony 仍不会自动将其纳入内核编译流程。因此，需要修改 `liteos_a` 文件夹下的 `BUILD.gn` 文件，将新增加的模块加入编译流程。具体操作为，在 `//kernel/liteos_a/BUILD.gn` 文件中，向目标 `group("modules")` 添加新建的文件夹模块。



完成上述操作后，重新编译与构建系统即完成了任务的要求。

2. 添加系统调用

本实验要求在 OpenHarmony 中开发系统调用，为 **LiteOS-a** 内核添加新的系统调用接口。通过在 **musl** 库中添加用户态函数接口，使后续应用程序能够调用该接口，从而使用新增的系统调用功能。具体要求如下：

1. LiteOS-a内核中新增系统调用接口A,在其中打印信息“Kernel mode:: SYS_print_taskinfo”
2. musl库中新增函数接口B，在其中打印信息“User mode:: SYS_print_taskinfo”，并调用系统接口B
3. 在实验二中创建的sysu子系统下新建新的组件和可执行程序模块C，使得运行C时其会调用函数接口B
4. 最终在OHOS系统中运行程序C的结果应当如下所示：

```
OHOS:/$ ./bin/print_taskinfo
User mode:: SYS_print_taskinfo
Kernel mode: SYS_print_taskinfo
```

流程

本任务需为 LiteOS-A 内核添加新的系统调用，并通过 musl 库间接调用该系统调用接口。因此，需要对 musl 库及 LiteOS-A 内核进行相应修改。大致流程如下：

1.需要在musl库，即 `third_party/musl/porting/liteos_a/user/arch/arm/bits/syscall.h.in` 下添加新系统调用声明。需要注意，新建的系统调用号需要在 `__NR_syscallend` 之前，因为内核系统调用的注册就以该标志为截至位。

```
C syscall.h.in M X
third_party > musl > porting > liteos_a > user > arch > arm > bits > C syscall.h.in
402 #define __NR_ohoscapset      ( __NR_OHOS_BEGIN + 20 )
403 #define __NR_sysconf         ( __NR_OHOS_BEGIN + 21 )
404
405 #define __NR_print_taskinfo   ( __NR_OHOS_BEGIN + 22 )           // 新增系统调用
406 #define __NR_syscallend      ( __NR_OHOS_BEGIN + 23 )
407
```

在 `third_party/musl/porting/liteos_a/user/include` 新的头文件声明系统调用函数。

```
// my_sys.h

#include "syscall.h"

void print_taskinfo();
```

在 `third_party/musl/porting/liteos_a/user/src` 新建函数定义文件，要注意其中的系统调用号应该是以 **SYS** 开头而不是原先的 **_NR**，这是因为经过预处理之后，原先的 **_NR** 调用号最后会被修改为 **SYS**。

```
// File: print_taskinfo.c

#include <mysys.h>
#include <stdio.h>
#include "syscall.h"

void print_taskinfo() {
    printf("User mode:: SYS_print_taskinfo\n");
    __syscall(SYS_print_taskinfo);
}
```

此部分无需手动编写 `BUILD.gn` 编译构建文件，musl 库已自动完成该步骤，会遍历文件夹下的所有内容并将其纳入编译流程。

3.在 `third_party/musl/porting/liteos_a/kernel/include/bits/syscall.h` 中添加系统调用号声明，该文件将被 LiteOS-A 内核引用。至此，musl 库中系统调用号的添加步骤已完成，接下来需在内核中实现对应的系统调用处理函数。

```
C syscall.h M X
third_party > musl > porting > liteos_a > kernel > include > bits > C syscall.h > __NR_syscallend
406 #define __NR_realpath      ( __NR_OHOS_BEGIN + 16 )
407 #define __NR_format         ( __NR_OHOS_BEGIN + 17 )
408 #define __NR_shellexec      ( __NR_OHOS_BEGIN + 18 )
409 #define __NR_ohoscapget     ( __NR_OHOS_BEGIN + 19 )
410 #define __NR_ohoscapset     ( __NR_OHOS_BEGIN + 20 )
411 #define __NR_sysconf        ( __NR_OHOS_BEGIN + 21 )
412 #define __NR_print_taskinfo ( __NR_OHOS_BEGIN + 22 )           // 新增系统调用
413 #define __NR_syscallend     ( __NR_OHOS_BEGIN + 23 )
414
```

4.从这一步开始的工作就是在内核子系统中进行，首先是需要要在 `kernel/liteos_a/syscall/los_syscall.h` 添加系统调用处理函数声明。

```

C los_syscall.h 6, M X
kernel > liteos_a > syscall > C los_syscall.h
316 extern int SysShellExec(const char *msgName, const char *cmdString);
317 extern int SysReboot(int magic, int magic2, int type);
318 extern int SysGetrusage(int what, struct rusage *ru);
319 extern long SysSysconf(int name);
320 extern int SysUgetrlimit(int resource, unsigned long long k_rlim[2]);
321 extern int SysSetrlimit(int resource, unsigned long long k_rlim[2]);
322 #ifdef LOSCFG_CHROOT
323 extern int SysChroot(const char *path);
324 #endif
325 #endif
326
327 /* 新增系统调用函数*/
328 extern void SysPrintTaskinfo(void);
329
330 #endif /* _LOS_SYSCALL_H */
331

```

接着需要在 kernel/liteos_a/syscall/syscall_lookup.h 添加系统调用号的注册，将新建的系统调用号与相应的处理函数联系到一起。

```

C syscall_lookup.h 1, M X
kernel > liteos_a > syscall > C syscall_lookup.h
263 SYSCALL_HAND_DEF(__NR_shmget, SysShmGet, int, ARG_NUM_3)
264 SYSCALL_HAND_DEF(__NR_shmctl, SysShmCtl, int, ARG_NUM_3)
265 #endif
266
267 SYSCALL_HAND_DEF(__NR_statx, SysStatx, int, ARG_NUM_5)
268
269 #ifdef LOSCFG_CHROOT
270 SYSCALL_HAND_DEF(__NR_chroot, SysChroot, int, ARG_NUM_1)
271 #endif
272
273 /* LiteOS customized syscalls, not compatible with ARM EABI */
274 SYSCALL_HAND_DEF(__NR_pthread_set_detach, SysUserThreadSetDetach, int, ARG_NUM_1)
275 SYSCALL_HAND_DEF(__NR_pthread_join, SysThreadJoin, int, ARG_NUM_1)
276 SYSCALL_HAND_DEF(__NR_pthread_deatch, SysUserThreadDetach, int, ARG_NUM_1)
277 SYSCALL_HAND_DEF(__NR_create_user_thread, SysCreateUserThread, unsigned int, ARG_NUM_3)
278 SYSCALL_HAND_DEF(__NR_getrusage, SysGetrusage, int, ARG_NUM_2)
279 SYSCALL_HAND_DEF(__NR_sysconf, SysSysconf, long, ARG_NUM_1)
280 SYSCALL_HAND_DEF(__NR_ugetrlimit, SysUgetrlimit, int, ARG_NUM_2)
281 SYSCALL_HAND_DEF(__NR_setrlimit, SysSetrlimit, int, ARG_NUM_2)
282
283
284 /* 新增系统调用处理 */
285 SYSCALL_HAND_DEF(__NR_print_taskinfo, SysPrintTaskinfo, void, ARG_NUM_0)
286

```

在 kernel/liteos_a/syscall 下新建文件，实现系统调用处理函数的定义，其中需打印 "Kernel mode:: SYS_print_taskinfo"。

```

// my_syscall.c

#include "los_printf.h"

void SysPrintTaskinfo(void) {
    PRINTK("Kernel mode: SYS_print_taskinfo\n");
    return;
}

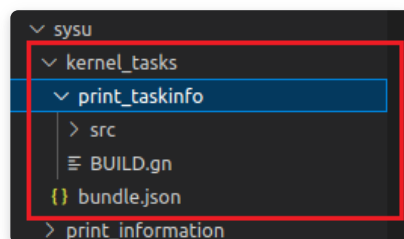
```

最后需要在 kernel/liteos_a/syscall/BUILD.gn 中 sources 添加新建的文件,将其加入到文件编译流程之中去

```
≡ BUILD.gn M X
kernel > liteos_a > syscall > ≡ BUILD.gn
30 import("//kernel/liteos_a/liteos.gni")
31
32 module_switch = defined(LOSCFG_KERNEL_SYSCALL)
33 module_name = get_path_info(rebase_path("."), "name")
34 kernel_module(module_name) {
35     sources = [
36         "fs_syscall.c",
37         "ipc_syscall.c",
38         "los_syscall.c",
39         "misc_syscall.c",
40         "net_syscall.c",
41         "process_syscall.c",
42         "syscall_pub.c",
43         "time_syscall.c",
44         "vm_syscall.c",
45         "my_syscall.c",
46     ]
47 }
48
```

至此，已在 OpenHarmony 的 LiteOS-A 内核中新增了一个系统调用。接下来的工作是在新编写的程序中调用该系统调用函数进行验证。

3.在上一次实验中已为 OpenHarmony 创建了 `sysu` 子系统，本次任务需在该子系统中新增一个组件，用于调用本次任务中在 `musl` 库新增的接口，从而实现对系统调用的调用。



新增组件的流程和内容与之前的一样。其中各个文件内容如下：

- `bundle.json`

```
{
  "name": "@ohos/kernel_tasks",
  "description": "kernel_tasks.",
  "version": "1.0",
  "license": "Apache License 2.0",
  "publishAs": "code-segment",
  "segment": {
    "destPath": ""
  },
  "dirs": {},
  "scripts": {},
  "component": {
    "name": "kernel_tasks",
    "subsystem": "sysu",
  }
}
```

```

        "syscap": [],
        "features": [],
        "adapted_system_type": [
            "small"
        ],
        "rom": "",
        "ram": "",
        "deps": {
            "components": [],
            "third_party": []
        },
        "build": {
            "sub_component": [
                "//sysu/kernel_tasks/print_taskinfo"
            ],
            "inner_kits": [],
            "test": []
        }
    }
}

```

- BUILD.gn

```

import("//build/ohos.gni")

ohos_executable("print_taskinfo") {
    sources=[
        "src/main.c"
    ]
    include_dirs=[
        "//kernel/liteos_a/kernel/include"
    ]

    subsystem_name="sysu"
    part_name="kernel_tasks"
}

```

- main.c

```

//main.c

#include <stdio.h>
#include <mysys.h>

int main() {
    print_taskinfo();
    return 0;
}

```

最后重新编译OpenHarmony并进入OHOS系统运行新增组件程序，即可打印任务目标信息。