厦門大學

软件学院

《实用操作系统》实验报告

题　　目 鸿蒙 LiteOS-a 内核移植——系统时钟移植

姓　　名 陈澄

学　　号 32420212202930

班　　级 软工三班

实验时间 2023/11/22

2023　年　11　月　22　日

1 实验目的

鸿蒙 LiteOS-a 内核移植——系统时钟移植

2 实验步骤

2.1 GenericTimer 源码分析

打开 kernel\liteos_a\platform\hw\arm\timer\arm_generic\arm_generic_timer.c

2.1.1 初始化

该函数的主要任务是初始化系统时钟，并创建定时器中断处理程序。具体而言，它执行以下步骤：

1.调用 HalClockFreqRead()函数获取系统时钟 SystemCounter 频率，并将其存储在全局变量 g_sysClock 中。

2. 调用 LOS_HwiCreate() 函数创建定时器中断处理程序，并将其注册为 OS_TICK_INT_NUM 号中断。函数的第二个参数 MIN_INTERRUPT_PRIORITY 指定了中断的优先级。当中断处理程序被触发时，将会跳转到 OsTickEntry()函数。

3.如果创建中断处理程序失败，则会在终端打印一条错误消息。

```
LITE_OS_SEC_TEXT_INIT VOID HalClockInit(VOID)
{
    UINT32 ret;

    g_sysClock = HalClockFreqRead();
    ret = LOS_HwiCreate(OS_TICK_INT_NUM, MIN_INTERRUPT_PRIORITY, 0, OsTickEntry,
0);
    if (ret != LOS_OK) {
        PRINT_ERR("%s, %d create tick irq failed, ret:0x%x\n", __FUNCTION__,
__LINE__, ret);
    }
}
```

2.1.2 启动 Timer

这段代码的主要功能是启动系统时钟和定时器中断，以便系统可以开始运行。

具体而言，该函数执行以下步骤：

1.调用 HalIrqUnmask()函数解除 OS_TICK_INT_NUM 号中断的屏蔽，并允许中断处理程序被触发。这样，当定时器中断到来时，处理程序就可以被调用。

2.调用 TimerCtlWrite()函数将定时器控制寄存器设置为 0，清除定时器中断标志位。这是为了确保第一个定时器中断不会立即发生，可以等待一段时间。

3.调用 TimerTvalWrite()函数将定时器超时值寄存器设置为 OS_CYCLE_PER_TICK，即每个滴答周期的时长。这将告诉定时器在多长时间后触发中断。

4.调用 TimerCtlWrite()函数将定时器控制寄存器设置为 1，开启定时器并开始计数。

```
LITE_OS_SEC_TEXT_INIT VOID HalClockStart(VOID)
{
    HalIrqUnmask(OS_TICK_INT_NUM);

    /* triggle the first tick */
    TimerCtlWrite(0);
    TimerTvalWrite(OS_CYCLE_PER_TICK);
    TimerCtlWrite(1);
}
```

2.1.3 中断处理

这段代码是用于处理定时器中断的入口函数 OsTickEntry。当系统定时器产生中断时，将会跳转执行该函数中的代码。

具体而言，该函数执行以下步骤：

1.调用 TimerCtlWrite(0)函数将定时器控制寄存器设置为 0，即清除定时器中断标志位。

2.调用 OsTickHandler()函数进行系统的滴答处理。这可能包括更新系统定时器、任务调度等操作。

3.根据注释的说明，使用上一个比较值（last cval）来生成下一个滴答的时序，以确保时序的绝对和精确。这里强调不要使用 tval 来驱动通用时间，否则滴答将会变慢。

4.调用 TimerCvalWrite()函数将下一个滴答的时刻写入比较值寄存器，这里使用当前的比较值加上 OS_CYCLE_PER_TICK，表示在当前比较值的基础上增加一个滴答周期的时间。

5.最后调用 TimerCtlWrite(1)函数将定时器控制寄存器设置为 1，重新开启定时器并开始计数，准备下一次的定时器中断。

```
LITE_OS_SEC_TEXT VOID OsTickEntry(VOID)
{
    TimerCtlWrite(0);

    OsTickHandler();

    /*
     * use last cval to generate the next tick's timing is
     * absolute and accurate. DO NOT use tval to drive the
     * generic time in which case tick will be slower.
     */
    TimerCvalWrite(TimerCvalRead() + OS_CYCLE_PER_TICK);
    TimerCtlWrite(1);
}
```

3 实验遇到的问题及其解决方法

无

4 我的体会

在本次实验中，我们对鸿蒙系统中的时钟模块 GenericTimer 进行了深入的源码分析。通过对相关源码的逐行解读和理解，我们深入了解了系统时钟的工作原理以及与定时器中断相关的关键代码。

首先，我们分析了 LITE_OS_SEC_TEXT_INIT 函数中的代码，这段代码负责启动系统时钟和定时器中断。通过调用 HalIrqUnmask()函数解除中断屏蔽，设置定时器控制寄存器，并配置定时器超时值，系统可以开始正常运行并响应定时器中断。

接着，我们研究了 OsTickEntry 函数，这是处理定时器中断的入口函数。在这个函数中，我们清除定时器中断标志位、执行系统滴答处理、更新下一个滴答的时刻，并重新开启定时器。这些操作保证了系统时钟的正常运行和时序的精确性。

通过这次实验，我们深入了解了鸿蒙系统时钟 GenericTimer 的实现原理，以及系统时钟与定时器中断相关的重要代码。这些知识对于理解系统的时间管理、任务调度等核心功能具有重要意义。同时，通过源码分析，我们也提升了对嵌入式系统底层代码的理解能力和分析能力，为今后的系统开发和调试工作打下了坚实的基础。