凌智電子
LOCKZHINER Electronic

OpenHarmony内核开发
事件

凌智電子
LOCKZHINER Electronic

目 录

CONTENTS

凌智電子
LOCKZHINER Electronic

01 什么是事件

- OpenHarmony内核中，事件（Event）是一种任务间通信的机制，可用于任务间的同步。
- 多任务环境下，任务之间往往需要同步操作，一个等待即是一个同步。事件可以提供一对多、多对多的同步操作。
- 一对多同步模型：一个任务等待多个事件的触发。可以是任意一个事件发生时唤醒任务处理事件，也可以是几个事件都发生后才唤醒任务处理事件。
- 多对多同步模型：多个任务等待多个事件的触发。

01 什么是事件

- LiteOS提供的事件具有如下特点：
- 任务通过创建事件控制块来触发事件或等待事件。
- 事件间相互独立，内部实现为一个32位无符号整型，每一位标识一种事件类型。第25位不可用，因此最多可支持31种事件类型。
- 事件仅用于任务间的同步，不提供数据传输功能。
- 多次向事件控制块写入同一事件类型，在被清零前等效于只写入一次。
- 多个任务可以对同一事件进行读写操作。
- 支持事件读写超时机制。

02 事件机制

- 当任务调用LOS_EventRead接口读事件时，可以根据eventMask读取单个或者多个事件类型。
- 当任务调用LOS_EventWrite接口写事件时，可以一次同时写多个事件类型。
- 当任务调用LOS_EventClear接口清除事件时，可以对多个事件对应位进行清0操作。

03 事件的接口

事件接口的头文件

/kernel/liteos_m/kernel/include/los_event.h

OpenHarmony内核开发中，事件接口主要分为几大类：

（1）初始化事件；

（2）读/写事件；

（3）清除事件；

（4）校验事件掩码；

（5）销毁事件。

03 事件的接口

功能分类	接口名	功能描述
初始化事件	LOS_EventInit	初始化一个事件控制块。
读/写事件	LOS_EventRead	读取指定事件类型，超时时间为相对时间：单位为Tick。
	LOS_EventWrite	写指定的事件类型。
清除事件	LOS_EventClear	清除指定的事件类型。
校验事件掩码	LOS_EventPoll	根据用户传入的事件ID、事件掩码及读取模式，返回用户传入的事件是否符合预期。
销毁事件	LOS_EventDestroy	销毁指定的事件控制块。

凌智電子
LOCKZHINER Electronic

03 事件的接口

```
UINT32 LOS_EventInit(PEVENT_CB_S eventCB);
```

该函数主要功能是初始化一个事件控制块。

其中，参数eventCB为初始化的事件控制块指针。

执行函数返回LOS_OK为成功，其余为失败。

03 事件的接口

UINT32 LOS_EventDestroy(PEVENT_CB_S eventCB);

该函数主要功能是销毁指定的事件控制块。

其中，参数eventCB为需要销毁的事件控制块指针。

接口执行完毕后，返回LOS_OK为成功，其余为失败。

03 事件的接口

```
UINT32 LOS_EventRead(PEVENT_CB_S eventCB,
                     UINT32 eventMask,
                     UINT32 mode,
                     UINT32 timeOut);
```

该函数主要功能是读取指定事件类型，超时时间为相对时间。单位为tick。

其中，参数eventCB为需要读取的事件控制块指针；

参数eventMask为事件掩码；

参数mode为事件读取的模式；

参数timeout为超时时间。

接口执行完毕后，返回LOS_OK为成功，其余为失败。

03 事件的接口

```
UINT32 LOS_EventWrite(PEVENT_CB_S eventCB, UINT32 events);
```

该函数主要功能是数据写入指定事件类型。其中，

参数eventCB为需要写入的事件控制块指针；

参数events为需要写入的事件掩码。

接口执行完毕后，返回LOS_OK为成功，其余为失败。

03 事件的接口

UINT32 LOS_EventClear(PEVENT_CB_S eventCB, UINT32 eventMask);

该函数主要功能是数据事件事件掩码，清除事件控制块中的事件。其中，

参数eventCB为需要写入的事件控制块指针；

参数eventMask为需要清除的事件掩码。

接口执行完毕后，返回LOS_OK为成功，其余为失败。

04 如何使用事件

1、打开sdk下面路径的文件

vendor/lockzhiner/rk2206/samples/a6_kernal_event/kernel_event_example.c

2、创建队列

在event_example函数中，通过LOS_EventInit函数初始化事件，并通过LOS_TaskCreate创建event_master_thread和event_slave_thread两个任务。

```
void event_example()
{
 unsigned int ret = LOS_OK;

    ret = LOS_EventInit(&m_event);
    if (ret != LOS_OK)
    {
        printf("Falied to create EventFlags\n");
        return;
    }
}
```

如何使用事件

```
task1.pfnTaskEntry = (TSK_ENTRY_FUNC)event_master_thread;
    task1.uwStackSize = 2048;
    task1.pcName = "event_master_thread";
    task1.usTaskPrio = 5;
    ret = LOS_TaskCreate(&thread_id1, &task1);
    if (ret != LOS_OK)
    {
        printf("Falied to create event_master_thread ret:0x%x\n", ret);
        return;
    }
```

```
task2.pfnTaskEntry = (TSK_ENTRY_FUNC)event_slave_thread;
    task2.uwStackSize = 2048;
    task2.pcName = "event_slave_thread";
    task2.usTaskPrio = 5;
    ret = LOS_TaskCreate(&thread_id2, &task2);
    if (ret != LOS_OK)
    {
        printf("Falied to create event_slave_thread ret:0x%x\n", ret);
        return;
    }
```

如何使用事件

```
void event_master_thread()
{
    unsigned int ret = LOS_OK;

    LOS_Msleep(1000);

    while (1)
    {
        printf("%s write event:0x%x\n", __func__, EVENT_WAIT);
        ret = LOS_EventWrite(&m_event, EVENT_WAIT);
        if (ret != LOS  OK) {
            printf("%s write event failed ret:0x%x\n", __func__, ret);
        }

        /*delay 1s*/
        LOS  Msleep(2000);
        LOS_EventClear(&m_event, ~m_event.uwEventID);
    }
}
```

```
void event_slave_thread()
{
    unsigned int event;

    while (1)
    {
        event = LOS_EventRead(&m_event, EVENT_WAIT,
LOS_WAITMODE_AND, LOS_WAIT_FOREVER);
        printf("%s read event:0x%x\n", __func__, event);
        LOS_Msleep(1000);
    }
}
```

如何使用事件

4.修改编译脚本

修改 vendor/lockzhiner/rk2206/sample 路径下 BUILD.gn 文件，指定 a6_kernal_event 参与编译。

```
 “./a6_kernal_event:kernal_event",
```

修改 device/lockzhiner/rk2206/sdk_liteos 路径下 Makefile 文件，添加 –lkernal_event 参与编译。

```
hardware_LIBS = -lhal_iothardware -lhardware -lkernal_event
```

5.编译固件

```
hb set -root .
hb set
hb build -f
```

如何使用事件

6.烧写固件

7.通过串口查看结果

运行结果

```
event_master_thread write event:0x1
event_slave_thread read event:0x1
event_slave_thread read event:0x1
event_master_thread write event:0x1
event_slave_thread read event:0x1
event_slave_thread read event:0x1
......
```

凌智電子
LOCKZHINER Electronic

谢谢聆听

单击此处添加副标题内容