LISA_os

LISA_OS_API 参考手册

Version	Data	Description
V1.0	2022.4	First Release

概述

LISA_OS_API 是聆思基于市场主流的 RTOS 和芯片平台，定义的一个介于底层驱动与上层应用的中间层。用于屏蔽不同 OS 的 API 的差别，方便应用层代码开发和移植。包含线程、互斥锁、信号量、队列、定时器及内存单元等相关接口。

API说明

1. Thread

lisa_thread_create

```
lisa_thread_t *lisa_thread_create(const lisa_thread_attr_t *attr,
                                  void (*entry)(void *),
                                  void *arg)
```

参数:
attr: 线程属性

```
typedef struct {
    char *name;
    uint32_t stack_size;    // 栈大小
    uint32_t priority;      // 优先级
} lisa_thread_attr_t;
```

entry: 线程函数

```
void (*entry)(void *)
```

arg: 线程函数的参数
返回值:
lisa_thread_t * 线程指针
失败 返回 NULL
成功 返回非 NULL
说明:
创建线程

lisa_thread_delete

```
lisa_err_t lisa_thread_delete(lisa_thread_t *thread)
```

参数:
thread: 线程指针
返回值:
LISA_OK: 成功
LISA_FAIL: 失败
说明:
销毁线程

lisa_thread_yield

```
1.  lisa_err_t lisa_thread_yield(void)
```

参数:
 void
返回值:
 LISA_OK: 成功
 LISA_FAIL: 失败
说明:
 启动线程调度

lisa_thread_suspend

```
1.  lisa_err_t lisa_thread_suspend(lisa_thread_t *thread)
```

参数:
 thread: 线程指针
返回值:
 LISA_OK: 成功
 LISA_FAIL: 失败
说明:
 挂起线程

lisa_thread_resume

```
1.  lisa_err_t lisa_thread_resume(lisa_thread_t *thread)
```

参数:
 thread: 线程指针
返回值:
 LISA_OK: 成功
 LISA_FAIL: 失败
说明:
 恢复线程

lisa_thread_delay

```
1.  lisa_err_t lisa_thread_delay(uint32_t ticks)
```

参数:
 tick: 延时时间，单位为秒
返回值:
 LISA_OK: 成功
 LISA_FAIL: 失败
说明:
 线程延时，单位为秒

lisa_thread_mdelay

```
1.  lisa_err_t lisa_thread_mdelay(uint32_t ms)
```

参数:
 ms: 单位为ms
返回值:
 LISA_OK: 成功
 LISA_FAIL: 失败
说明:
 线程延时，单位为ms

lisa_thread_getcurrent

```
1.  lisa_thread_t *lisa_thread_getcurrent(void)
```

参数:
 void

返回值:
LISA_OK: 成功
LISA_FAIL: 失败
说明:
获取当前线程指针

2. Queue

lisa_queue_create

```
lisa_queue_t *lisa_queue_create(uint32_t count, uint32_t item_size)
```

参数:
count: 消息队列中消息的最大个数
item_size: 单个消息的大小
返回值:
lisa_queue_t *: 消息队列指针
成功: 返回消息队列指针
失败: 返回 NULL
说明:
创建消息队列

lisa_queue_push

```
lisa_err_t lisa_queue_push(lisa_queue_t *queue,
                           void *item,
                           uint32_t item_size,
                           int32_t wait)
```

参数:
queue: 消息队列指针
item: 发送数据的地址
item_size: 单个消息的大小
wait: 超时时间，单位为ms
返回值:
LISA_OK: 发送成功
LISA_FAIL: 出现错误
说明:
向消息队列里面发送消息

lisa_queue_receive

```
lisa_err_t lisa_queue_receive(lisa_queue_t *queue,
                              void *item,
                              uint32_t item_size,
                              int32_t wait)
```

参数:
queue: 消息队列指针
item: 接收数据的指针
item_size: 单个消息的大小
wait: 超时时间，单位为ms
返回值:
LISA_OK: 接收成功
LISA_FAIL: 出现错误
说明:
从消息队列里面接收消息；如果超时时间到了，还没有收到消息，那么返回 LISA_FAIL；在超时时间到达前收到消息，那么返回 LISA_OK；

lisa_queue_delete

```
lisa_err_t lisa_queue_delete(lisa_queue_t *queue)
```

参数:
queue: 消息队列指针
返回值:

LISA_OK： 成功
LISA_FAIL： 失败
说明：
销毁消息队列

lisa_queue_available

```
1.  uint32_t lisa_queue_available(lisa_queue_t *queue)
```

参数：
queue：消息队列指针
返回值：
消息队列中剩余可用的消息的个数
说明：
查询消息队列中空闲消息的个数

lisa_queue_waiting

```
1.  uint32_t lisa_queue_waiting(lisa_queue_t *queue)
```

参数：
queue：消息队列指针
返回值：
消息队列中已有消息的个数
说明：
查询消息队列中正在等待的消息的个数

lisa_queue_clear

```
1.  lisa_err_t lisa_queue_clear(lisa_queue_t *queue)
```

参数：
queue：消息队列指针
返回值：
LISA_OK： 成功
LISA_FAIL： 失败
说明：
复位消息队列，此函数会清空消息队列

3. Mutex

lisa_mutex_create

```
1.  lisa_mutex_t *lisa_mutex_create(void)
```

参数：
void
返回值：
lisa_mutex_t*：互斥量的指针
成功： 返回互斥量指针
失败： 返回NULL
说明：
创建互斥量

lisa_mutex_lock

```
1.  lisa_err_t lisa_mutex_lock(lisa_mutex_t *mutex, int32_t block_time)
```

参数：
mutex： 互斥量指针
block_time：超时时间，单位为ms
返回值：
LISA_OK： 成功
LISA_FAIL： 失败
说明：

互斥量上锁，如果超时时间到了，还无法获得锁，那么返回 LISA_FAIL；在超时时间到达前获得锁，那么返回 LISA_OK；

lisa_mutex_unlock

```
1.  lisa_err_t lisa_mutex_unlock(lisa_mutex_t *mutex)
```

参数:
mutex： 互斥量指针
返回值:
LISA_OK： 成功
LISA_FAIL： 失败
说明:
互斥量解锁

lisa_mutex_delete

```
1.  lisa_err_t lisa_mutex_delete(lisa_mutex_t *mutex)
```

参数:
mutex： 互斥量指针
返回值:
LISA_OK： 成功
LISA_FAIL： 失败
说明:
销毁互斥量

4 .Semaphore

lisa_semaphore_create

```
1.  lisa_semaphore_t *lisa_semaphore_create(uint32_t count)
```

参数:
count： 信号量最大个数
返回值:
lisa_semaphore_t *： 信号量指针
成功： 返回信号量指针
失败： 返回NULL
说明:
创建信号量

lisa_semaphore_take

```
1.  lisa_err_t lisa_semaphore_take(lisa_semaphore_t *sem, int32_t block_time)
```

参数:
sem： 信号量指针
block_time：超时时间
返回值:
LISA_OK： 成功
LISA_FAIL： 失败
说明:
获取信号量，如果超时时间到了，还无法获得信号量，那么返回LISA_FAIL；在超时时间到达前获得信号量，那么返回LISA_OK；

lisa_semaphore_give

```
1.  lisa_err_t lisa_semaphore_give(lisa_semaphore_t *sem)
```

入参	注释
mutex	信号量指针

返回值:
LISA_OK： 成功
LISA_FAIL： 失败

说明:
释放信号量

lisa_semaphore_delete

```
lisa_err_t lisa_semaphore_delete(lisa_semaphore_t *sem)
```

参数:
mutex: 信号量指针
返回值:
LISA_OK: 成功
LISA_FAIL: 失败
说明:
销毁信号量

lisa_semaphore_getcount

```
uint32_t lisa_semaphore_getcount(lisa_semaphore_t *sem)
```

参数:
mutex: 信号量指针
返回值:
可用的信号量个数
说明:
获取可用的信号量个数

5. Timer

lisa_timer_create

```
lisa_timer_t *lisa_timer_create(lisa_timertype type,
                                lisa_timercb_t cb,
                                void *arg,
                                uint32_t period_ms)
```

参数:
type: 定时器类型

```
typedef enum {
    OS_TIMER_ONCE = 0,     // 单次定时器
    OS_TIMER_PERIODIC = 1, // 周期定时器
} lisa_timertype;
```

cb: 定时器回调函数

```
typedef void (*lisa_timercb_t)(void *arg)
```

arg: 定时器回调函数的参数
period_ms: 定时器的周期，单位ms
返回值:
lisa_timer_t *: 定时器指针
成功: 返回定时器指针
失败: 返回NULL
说明:
创建软件定时器

lisa_timer_delete

```
lisa_err_t lisa_timer_delete(lisa_timer_t *timer)
```

参数:
timer: 定时器指针
返回值:
LISA_OK: 成功
LISA_FAIL: 失败

说明:
销毁定时器

lisa_timer_start

```
1.  lisa_err_t lisa_timer_start(lisa_timer_t *timer)
```

参数:
timer: 定时器指针
返回值:
LISA_OK: 成功
LISA_FAIL: 失败
说明:
启动定时器

lisa_timer_stop

```
1.  lisa_err_t lisa_timer_stop(lisa_timer_t *timer)
```

参数:
timer: 定时器指针
返回值:
LISA_OK: 成功
LISA_FAIL: 失败
说明:
停止定时器

lisa_timer_change_period

```
1.  lisa_err_t lisa_timer_change_period(lisa_timer_t *timer, uint32_t period_ms)
```

参数:
timer: 定时器指针
period: 定时器周期，单位ms
返回值:
LISA_OK: 成功
LISA_FAIL: 失败
说明:
修改定时器周期

6 .Time

lisa_os_get_ticks

```
1.  uint32_t lisa_os_get_ticks(void)
```

参数:
void
返回值:
返回从 OS 启动后到现在为止的时间间隔，单位为时间片，时间片由具体 OS 实现决定
说明:
获得 OS 启动到现在的时间间隔，单位为时间片

lisa_os_get_time

```
1.  uint32_t lisa_os_get_time(void)
```

参数:
void
返回值:
返回从 OS 启动后到现在为止的时间，单位为秒
说明:
获得 OS 启动到现在的时间，单位为秒

lisa_rand32

```
uint32_t lisa_rand32(void)
```

参数:
void
返回值:
返回随机值（直接返回os的当前tick值）
说明:
获得32位随机值

7.Memory

> 基于Zephyr操作系统，在使用lisa memory相关接口之前，首先需要使用CONFIG_LISA_HEAP_ARENA_SIZE配置heap区大小。

lisa_mem_alloc

```
void *lisa_mem_alloc(uint32_t size);
```

参数:
size：大小为byte
返回值:
void*
说明:
申请内存空间

lisa_mem_realloc

```
void *lisa_mem_realloc(void *ptr, uint32_t size)
```

参数:
ptr：内存空间地址
size：申请内存大小，单位为byte
返回值:
void*
成功：返回非NULL指针
失败：返回NULL
说明:
重新申请内存空间

lisa_mem_calloc

```
void *lisa_mem_calloc(uint32_t count, uint32_t size)
```

参数:
count：内存块个数
size：单个内存块大小，单位为byte
返回值:
void*
成功：返回非NULL指针
失败：返回NULL
说明:
申请count*size的内存空间

lisa_mem_free

```
void lisa_mem_free(void *ptr)
```

参数:
ptr：内存空间指针
返回值:
void
说明:
释放内存空间

lisa_strdup

```
1.  char *lisa_strdup(const char *s)
```

参数:
s：字符串
返回值:
char*
成功：返回非NULL指针
失败：返回NULL
说明:
字符串拷贝函数