_thread- 线程、互斥锁、信号量

版本：1.0

日期：2026-02-04

状态：受控/临时文件

文件保密级别：（请勾选 ■ ）		
绝密 □	保密 □	公开 ■

文件管控表

文件变更记录			
修订日期	最新版本	修订内容	编制
2026-02-04	1	首次发行	Wells Li

上海移远通信技术股份有限公司（以下简称“移远通信”）始终以为客户提供最及时、最全面的服务为宗旨。如需任何帮助，请随时联系我司上海总部，联系方式如下：

上海移远通信技术股份有限公司
上海市闵行区田林路 1016 号科技绿洲 3 期（B 区）5 号楼　　邮编：200233
电话：+86 21 5108 6236　　邮箱：info@quectel.com

或联系我司当地办事处，详情请登录：http://www.quectel.com/cn/support/sales.htm。

如需技术支持或反馈我司技术文档中的问题，请随时登录网址：
http://www.quectel.com/cn/support/technical.htm 或发送邮件至：support@quectel.com。

前言

移远通信提供该文档内容以支持客户的产品设计。客户须按照文档中提供的规范、参数来设计产品。同时，您理解并同意，移远通信提供的参考设计仅作为示例。您同意在设计您目标产品时使用您独立的分析、评估和判断。在使用本文档所指导的任何硬软件或服务之前，请仔细阅读本声明。您在此承认并同意，尽管移远通信采取了商业范围内的合理努力来提供尽可能好的体验，但本文档和其所涉及服务是在“可用”基础上提供给您的。移远通信可在未事先通知的情况下，自行决定随时增加、修改或重述本文档。

使用和披露限制

许可协议

除非移远通信特别授权，否则我司所提供硬软件、材料和文档的接收方须对接收的内容保密，不得将其用于除本项目的实施与开展以外的任何其他目的。

版权声明

移远通信产品和本协议项下的第三方产品可能包含受移远通信或第三方材料、硬软件和文档版权保护的相关资料。除非事先得到书面同意，否则您不得获取、使用、向第三方披露我司所提供的文档和信息，或对此类受版权保护的资料进行复制、转载、抄袭、出版、展示、翻译、分发、合并、修改，或创造其衍生作品。移远通信或第三方对受版权保护的资料拥有专有权，不授予或转让任何专利、版权、商标或服务商标权的许可。为避免歧义，除了正常的非独家、免版税的产品使用许可，任何形式的购买都不可被视为授予许可。对于任何违反保密义务、未经授权使用或以其他非法形式恶意使用所述文档和信息的违法侵权行为，移远通信有权追究法律责任。

商标

除另行规定，本文档中的任何内容均不授予在广告、宣传或其他方面使用移远通信或第三方的任何商标、商号及名称，或其缩略语，或其仿冒品的权利。

第三方权利

您理解本文档可能涉及一个或多个属于第三方的硬软件和文档（“第三方材料”）。您对此类第三方材料的使用应受本文档的所有限制和义务约束。

移远通信针对第三方材料不做任何明示或暗示的保证或陈述，包括但不限于任何暗示或法定的适销性或特定用途的适用性、平静受益权、系统集成、信息准确性以及与许可技术或被许可人使用许可技术相关的不侵犯任何第三方知识产权的保证。本协议中的任何内容都不构成移远通信对任何移远通信产品或任何其他硬软件、设备、工具、信息或产品的开发、增强、修改、分销、营销、销售、提供销售或以其他方式维持生产的陈述或保证。此外，移远通信免除因交易过程、使用或贸易而产生的任何和所有保证。

隐私声明

为实现移远通信产品功能，特定设备数据将会上传至移远通信或第三方服务器（包括运营商、芯片供应商或您指定的服务器）。移远通信严格遵守相关法律法规，仅为实现产品功能之目的或在适用法律允许的情况下保留、使用、披露或以其他方式处理相关数据。当您与第三方进行数据交互前，请自行了解其隐私保护和数据安全政策。

免责声明

1) 移远通信不承担任何因未能遵守有关操作或设计规范而造成损害的责任。
2) 移远通信不承担因本文档中的任何因不准确、遗漏、或使用本文档中的信息而产生的任何责任。
3) 移远通信尽力确保开发中功能的完整性、准确性、及时性，但不排除上述功能错误或遗漏的可能。除非另有协议规定，否则移远通信对开发中功能的使用不做任何暗示或法定的保证。在适用法律允许的最大范围内，移远通信不对任何因使用开发中功能而遭受的损害承担责任，无论此类损害是否可以预见。
4) 移远通信对第三方网站及第三方资源的信息、内容、广告、商业报价、产品、服务和材料的可访问性、安全性、准确性、可用性、合法性和完整性不承担任何法律责任。

目录

1 模块概述

_thread 模块提供了基本的线程操作功能，包括线程创建、互斥锁和信号量等同步原语，用于实现多线程编程。

2 线程操作

2.1. start_new_thread

2.1.1. 接口概述

启动一个新线程。

2.1.2. 函数原型

thread_id = _thread.start_new_thread(function, args[, kwargs])

2.1.3. 参数描述

参数名	是否必填	数据类型	默认值	说明
function	是	function	-	要在线程中执行的函数
args	否	tuple	-	传递给函数的参数元组
kwargs	否	dict	{}	传递给函数的关键字参数字典

2.1.4. 返回值描述

返回线程标识符。

2.1.5. 示例

```
import _thread
import utime

def worker(thread_id, count):
    for i in range(count):
        print(f"线程{thread_id}: {i}")
        utime.sleep_ms(100)

# 启动线程
thread_id = _thread.start_new_thread(worker, (1, 5))
```

2.2. get_ident

2.2.1. 接口概述

获取当前线程的标识符。

2.2.2. 函数原型

```
ident = _thread.get_ident()
```

2.2.3. 参数描述

无。

2.2.4. 返回值描述

返回当前线程的标识符。

2.2.5. 示例

```
import _thread

def show_id():
    print(f"线程 ID: {_thread.get_ident()}")

_thread.start_new_thread(show_id, ())
```

2.3. statck_size

2.3.1. 接口概述

获取或设置新线程的栈大小。

2.3.2. 函数原型

```
# 获取当前栈大小
size = _thread.stack_size()

# 设置新栈大小
old_size = _thread.stack_size(new_size)
```

2.3.3. 参数描述

参数名	是否必填	数据类型	默认值	说明
new_size	否	int	-	新线程的栈大小（字节），0 表示使用默认值，最小 2048 字节

2.3.4. 返回值描述

返回之前的栈大小（字节）。

2.3.5. 示例

```
import _thread
import utime

# 1. 查看当前栈大小
print(f"默认栈: {_thread.stack_size()} 字节")

# 2. 设置新栈大小
old_size = _thread.stack_size(8192)

# 3. 创建线程
def worker(data):
    print(f"处理: {data}")
    utime.sleep(0.3)

_thread.start_new_thread(worker, ("数据 A",))

# 4. 恢复栈大小
_thread.stack_size(old_size)
```

3 互斥锁

3.1. alloc_lock

3.1.1. 接口概述

创建一个新的互斥锁对象。

3.1.2. 函数原型

lock = _thread.allocate_lock()

3.1.3. 参数描述

无。

3.1.4. 返回值描述

返回一个新的锁对象。

3.2. lock.acquire

3.2.1. 接口概述

获取锁。

3.2.2. 函数原型

success = lock.acquire([waitflag])

3.2.3. 参数描述

参数名	是否必填	数据类型	默认值	说明
waitflag	否	bool	True	等待标志：True/1 表示阻塞等待，False/0 表示不等待立即返回

3.2.4. 返回值描述

成功获取返回 True，失败返回 False。

3.2.5. 示例

```
import _thread
```

```
lock = _thread.allocate_lock()

# 阻塞获取
if lock.acquire():
    print("获取成功")

# 非阻塞获取
if lock.acquire(False):
    print("立即获取成功")
else:
    print("锁已被占用")
```

3.3. lock.release

3.3.1. 接口概述

释放锁。

3.3.2. 函数原型

```
lock.release()
```

3.3.3. 参数描述

无。

3.3.4. 返回值描述

无返回值。如果锁未被获取，会抛出 RuntimeError。

3.3.5. 示例

```
import _thread
import utime

# 创建锁
lock = _thread.allocate_lock()
lock.acquire()
print("操作共享资源...")
lock.release()
```

3.4. lock.locked

3.4.1. 接口概述

检查锁是否已被获取。

3.4.2. 函数原型

is_locked = lock.locked()。

3.4.3. 参数描述

无。

3.4.4. 返回值描述

如果锁已被获取返回 True，否则返回 False。

3.4.5. 示例

```
import _thread
import utime

# 创建锁
lock = _thread.allocate_lock()

print("1. 初始状态:")
print(f"    lock.locked() = {lock.locked()}")   # False

print("\n2. 获取锁后:")
lock.acquire()
print(f"    lock.locked() = {lock.locked()}")   # True

print("\n3. 释放锁后:")
lock.release()
print(f"    lock.locked() = {lock.locked()}")   # False
```

4 信号量

4.1. allocate_semaphore

4.1.1. 接口概述

创建一个新的信号量对象。

4.1.2. 函数原型

sem = _thread.allocate_semaphore([initcount[, maxcount]])

4.1.3. 参数描述

参数名	是否必填	数据类型	默认值	说明
initcount	否	int	0	初始信号量计数
maxcount	否	int	10	最大信号量计数

4.1.4. 返回值描述

返回一个新的信号量对象。

4.2. sem.acquire

4.2.1. 接口概述

获取信号量。

4.2.2. 函数原型

success = sem.acquire([wait_ms])

4.2.3. 参数描述

参数名	是否必填	数据类型	默认值	说明
wait_ms	否	int	-1	等待时间（毫秒）： -1 表示无限等待 0 表示不等待 >0 表示等待指定毫秒数

4.2.4. 返回值描述

成功获取返回 True，失败返回 False，超时抛出 OSError。

4.2.5. 示例

```
import _thread
import utime

# 创建信号量（只有一个）
sem = _thread.allocate_semaphore(1)

# 先获取
sem.acquire()
print("主线程拿到了信号量")

# 尝试获取（会超时）
try:
    print("等 100ms 尝试获取...")
    got = sem.acquire(100)  # 等 100 毫秒
    print(f"拿到了: {got}")
except Exception as e:
    print(f"超时了！ 错误: {e}")

# 释放
sem.release()
print("释放信号量")

# 再获取（能成功）
try:
    print("再尝试获取...")
    got = sem.acquire(100)
    print(f"拿到了: {got}")
except Exception as e:
    print(f"错误: {e}")
```

4.3. sem.release

4.3.1. 接口概述

释放信号量。

4.3.2. 函数原型

sem.release()

4.3.3. 参数描述

无。

4.3.4. 返回值描述

无返回值。

4.3.5. 示例

```
import _thread
sem = _thread.allocate_semaphore()
sem.release()
```

4.4. sem.getCnt()

4.4.1. 接口概述

获取信号量计数。

4.4.2. 函数原型

(maxcnt, curcnt) = sem.getCnt()

4.4.3. 参数描述

无。

4.4.4. 返回值描述

返回元组 (maxcnt, curcnt)，其中：

- maxcnt：最大信号量计数
- curcnt：当前信号量计数。

4.4.5. 示例

```
import _thread

# 创建信号量，最大 3 个，初始 0 个
sem = _thread.allocate_semaphore()

# 获取信号量信息
maxcnt, curcnt = sem.getCnt()
```

```
print(f"最大数量: {maxcnt}, 当前数量: {curcnt}")

# 释放 1 个
sem.release()
maxcnt, curcnt = sem.getCnt()
print(f"释放 1 个后: 最大={maxcnt}, 当前={curcnt}")

# 再释放 1 个
sem.release()
maxcnt, curcnt = sem.getCnt()
print(f"释放 2 个后: 最大={maxcnt}, 当前={curcnt}")
```