QuecPython 多线程用户指导

LTE Standard 模块系列

版本：1.0.0

日期：2020-11-10

状态：临时文件

上海移远通信技术股份有限公司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 51086236　　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，可随时登陆如下网址：
http://www.quectel.com/cn/support/technical.htm 或发送邮件至：support@quectel.com。

前言

上海移远通信技术股份有限公司提供该文档内容用以支持其客户的产品设计。客户须按照文档中提供的规范、参数来设计其产品。因未能遵守有关操作或设计规范而造成的损害，上海移远通信技术股份有限公司不承担任何责任。在未声明前，上海移远通信技术股份有限公司有权对该文档进行更新。

免责声明

上海移远通信技术股份有限公司尽力确保开发中功能的完整性、准确性、及时性或效用，但不排除上述功能错误或遗漏的可能。除非其他有效协议另有规定，否则上海移远通信技术股份有限公司对开发中功能的使用不做任何暗示或明示的保证。在适用法律允许的最大范围内，上海移远通信技术股份有限公司不对任何因使用开发中功能而遭受的损失或损害承担责任，无论此类损失或损害是否可以预见。

保密义务

除非上海移远通信技术股份有限公司特别授权，否则我司所提供文档和信息的接收方须对接收的文档和信息保密，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。未经上海移远通信技术股份有限公司书面同意，不得获取、使用或向第三方泄露我司所提供的文档和信息。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，上海移远通信技术股份有限公司有权追究法律责任。

版权申明

本文档版权属于上海移远通信技术股份有限公司，任何人未经我司允许而复制转载该文档将承担法律责任。

文档历史

修订记录

版本	日期	作者	变更表述
-	2020-11-10	Kingka/Kenney	文档创建
1.0.0	2020-11-10	Kingka/Kenney	临时版本

目录

图片索引

1 引言

本文档主要以 EC100Y 为例介绍如何使用 QuecPython 类库方法快速开始多线程项目的开发。

适用模块：

- EC100Y-CN（本文以该模块为例进行详细介绍）
- EC600S-CN

2 多线程介绍

2.1. 多线程/进程的基本概念

Python 运行在 Python 虚拟机中，用户创建的多线程只是在 Python 虚拟机中的虚拟线程，而非在操作系统中的真正的线程。也就是说，Python 中的多线程是由 Python 虚拟机来进行轮询调度，而不是操作系统。

多线程可以使同一程序同时执行多个任务。线程在执行过程中与进程存在区别，在每个独立的线程中，都分别存在程序运行的入口、顺序执行序列以及程序的出口。并且线程必须依附在某个程序中，由程序来控制多个线程的运行。

2.2. 线程的基本操作

线程具有 5 种状态，状态转换的过程如下：

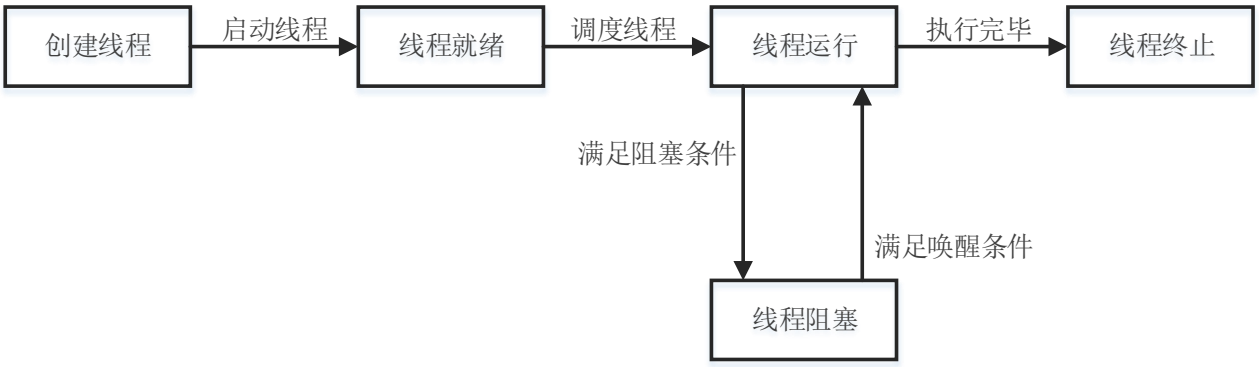

图 1：线程的 5 种状态

2.3. 线程和进程的主要区别

线程和进程都是操作系统控制程序运行的基本单位，系统可以利用这两个特性对程序实现高并发。而线程和进程的主要区别如下：

1. 一个程序至少有一个进程；一个进程中至少包含一个线程。
2. 进程在内存中拥有独立的存储空间，而多个线程则共享它所依赖的进程的存储空间。
3. 进程和线程对操作系统的资源管理的方式不同。

3 多线程 API 详解

3.1. _thread.allocate_lock

该函数用于创建一个互斥锁对象。

- **函数原型**

```
_thread.allocate_lock()
```

- **参数**

无

返回值

返回互斥锁对象。互斥锁对象的函数详见***第 3.1.1 章***。

3.1.1. 互斥锁对象函数

3.1.1.1. lock.acquire

该方法用于获取锁。

- **函数原型**

```
lock.acquire()
```

- **参数**

无

- **返回值**

True 成功
False 失败

3.1.1.2. lock.release

该方法用于释放锁。

- **函数原型**

```
lock.release()
```

- **参数**

无

- **返回值**

无

3.1.1.3. lock.locked

该方法用于返回锁的状态。

- **函数原型**

```
lock.locked()
```

- **参数**

无

- **返回值**

True 表示被某个线程获取
False 表示没有被线程获取

3.2. _thread.get_ident

该方法用于获取当前线程号。

- **函数原型**

```
_thread.get_ident()
```

- **参数**

无

- **返回值**

返回当前线程号。

3.3. _thread.stack_size

该方法用于设置创建新线程使用的堆栈大小。单位：字节。

- **函数原型**

```
_thread.stack_size(size)
```

- **参数**

size：
堆栈大小。默认值：8192。

- **返回值**

返回当前堆栈大小。

3.4. _thread.start_new_thread

该方法用于创建一个新线程。

- **函数原型**

```
_thread.start_new_thread(function, args)
```

- **参数**

function：
接收执行函数

args：
被执行函数参数

- **返回值**

无

4 多线程使用示例

步骤1： 将开发板接入电脑。接入后的操作方法详见《Quectel_QuecPython_基础操作说明》。

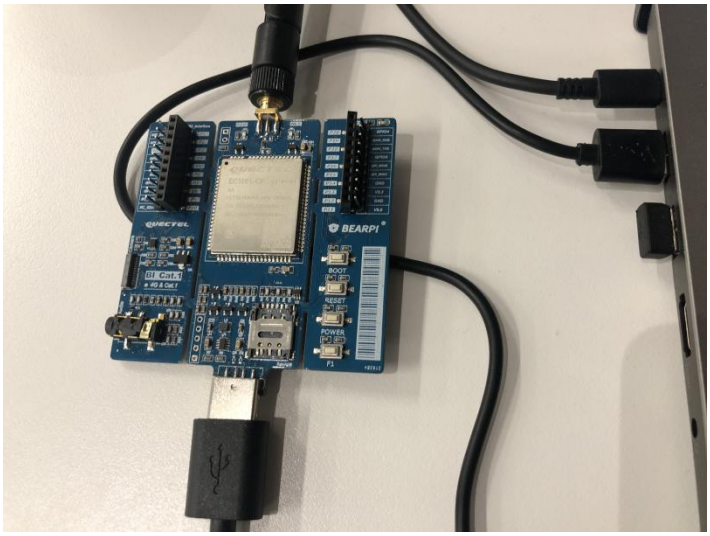

图 2：开发板接入电脑

步骤2： 创建 *test.py* 文件，在文件内导入 QuecPython 中的_thread 模块，编写多线程代码。

```
import _thread

def th_func(thread_id):
      Print("thread id is:%d" % thread_id)

for i in range(5):
    _thread.start_new_thread(th_func,(i+1,))
```

步骤3： 将 test.py 文件上传到开发板，上传方法详见《Quectel_QuecPython_基础操作说明》。

步骤4： 程序运行结果，如图所示：

```
>>>
>>> import example
>>> example.exec('test.py')
>>> thread id is:1
thread id is:2
thread id is:3
thread id is:4
thread id is:5
```

5 附录 A 参考文档

表 1：参考文档

序号	文档名称	备注
[1]	Quectel_QuecPython_基础操作说明	QuecPython 上传下载文件说明