# Proj1：向LiteOS中添加一个系统调用

该实验需要完成两部分内容，分别在用户态和内核态。

1. 在用户态下要为新添加的系统调用增加相应的库函数作为接口。
2. 在内核态下要添加与接口函数对应的系统调用。
3. 编写一个应用程序，该应用程序通过调用第一步中添加的库函数接口，进而触发新添加的系统调用，从而验证新添加系统调用的正确性。

参考链接：[OpenHarmony LiteOS-A内核文档之学习--系统调用-开源基础软件社区-51CTO.COM](https://ost.51cto.com/posts/8876)

# Proj2：向LiteOS中添加一个简单的基于线程运行时间的短作业优先调度策略

短作业优先调度策略为优化LiteOS的吞吐量。该策略在将线程TCB插入就绪队列时，按照线程执行的时间长短进行排序。运行时间短的线程先运行，运行时间长的线程后运行。该Proj要完成以下内容：

1. 修改LiteOS内核的PriQueInsert函数，在其中添加相关代码，实现按照任务运行时间的长短将任务的TCB插入就绪队列。
2. 在短作业优先调度策略中，采用LiteOS中定义的SchedParam结构体中的timeSlice成员作为该作业的运行时间。

typedef struct {

UINT16 policy;

UINT16 basePrio;

UINT16 priority;

UINT32 timeSlice;

} SchedParam;

1. 在用户层生成至少3个线程，通过pthread\_attr\_getschedparam和pthread\_attr\_setschedparam两个函数修改所生成线程的调度策略为“短作业优先“调度策略，并对timeSlice进行赋值，表示该线程的运行时间。验证你所编写的调度策略是正确的。线程内部可以用while语句循环timeSlice规定的时间，然后打印自己的TID后退出。

例如，对这3个线程的timeSlice分别赋予1、5、9。那么这3个线程也按照这个顺序完成，即第一个线程1秒后退出，第二个线程5秒后退出，第三个线程9秒后退出。

用户层采用pthread库实现，相关代码参考[Linux线程调度策略以及优先级实验（图文）-CSDN博客](https://blog.csdn.net/qq_33182418/article/details/121135914)

# Proj3：LiteOS同步实验

利用Pthread库，实现生产者-消费者问题。

使用的函数包括：pthread\_create、pthread\_join、pthread\_exit、sem\_init、sem\_post等。

参考程序：[Linux下用pthread实现“生产者---消费者”的同步与互斥-CSDN博客](https://blog.csdn.net/pingxiaozhao/article/details/122224969)

# Proj4：改进LiteOS中物理内存分配算法

LiteOS中的物理内存分配采用了TLSF算法，该算法较好地解决了最坏情况执行时间不确定(not bounded)或者复杂度过高(bounded with a too important bound")，以及碎片化问题(fragmentation)两个问题。

TLSF算法仍存在优化空间，Best-fit策略最主要的问题还在于第三步，仍然需要检索对应范围的那一条空闲块链表，存在潜在的时间复杂度。Good-fit思路与Best-fit不同之处在于，Good-fit并不保证找到满足需求的最小空闲块，而是尽可能接近要分配的大小。

以搜索大小为69字节的空闲块为例，Good-fit并不是找到[68 ~ 70]这一范围，而是比这个范围稍微大一点儿的范围(例如[71 ~ 73])。这样设计的好处就是[71 ~ 73]对应的空闲块链中每一块都能满足需求，不需要检索空闲块链表找到最小的，而是直接取空闲块链中第一块即可。整体上还不会造成太多碎片。

Good-fit分配策略将动态内存的分配与回收时间复杂度都降到了O(1)时间复杂度，并且保证系统运行时不会产生过多碎片。

相关理论参考[LiteOS内存管理：TLSF算法 - 简书 (jianshu.com)](https://www.jianshu.com/p/01743e834432)

也可以向LiteOS中添加一个内存分配算法，参考[三种内存分配算法总结及代码实现\_内存分配算法的实现-CSDN博客](https://blog.csdn.net/xp731574722/article/details/80341659)