

****

软 件 学 院

《实用操作系统》Project3

**题　　目 向LiteOS中添加一个短作业优先调度策略**

**姓　　名 陈澄**

**学　　号 32420212202930**

**班　　级 软工三班**

**实验时间 2023/9/26**

**2023 年 09 月 26 日**

# 实验目的

向LiteOS中添加一个简单的基于线程运行时的短作业优先调度策略

# 实验环境

主机： Windows 11

虚拟机： Ubuntu 18.04

开发板： IMAX6ULL MINI

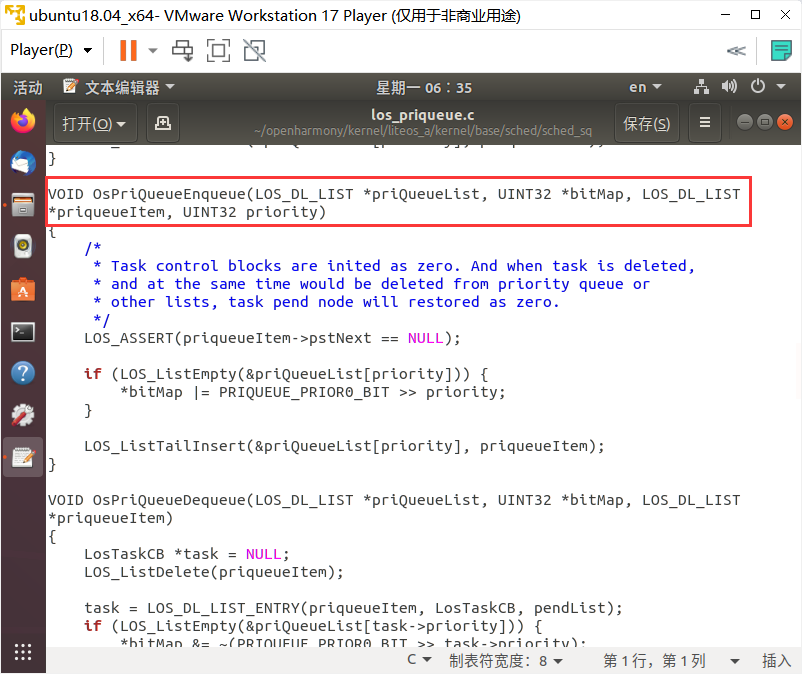
终端： MobaXterm

# 实验内容

1. **修改优先级队列函数**

打开openharmony/kernel/liteos\_a/kernel/base/sched/sched\_sq/los\_priqueue.c

找到OsPriQueueEnqueue方法

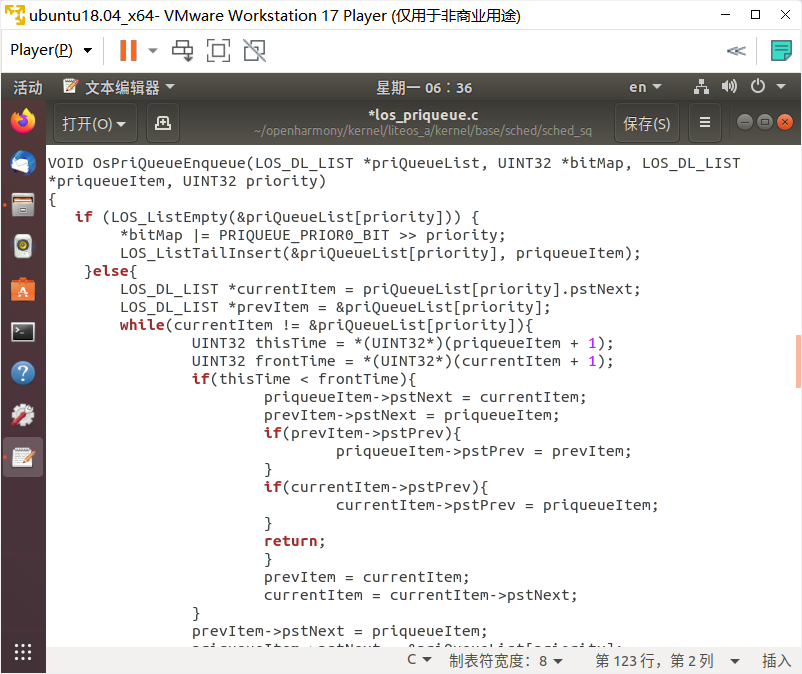


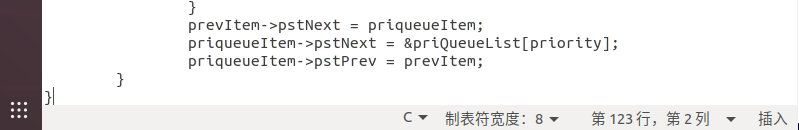
该方法就是优先级队列函数

将其更改即可改变调度策略

本实验将其替换为运行时间短的优先策略

代码如下

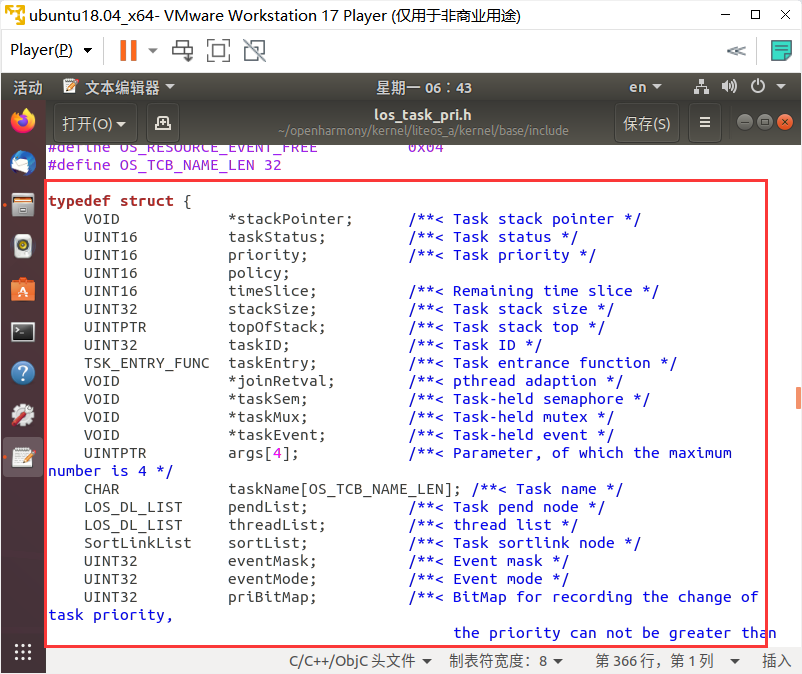




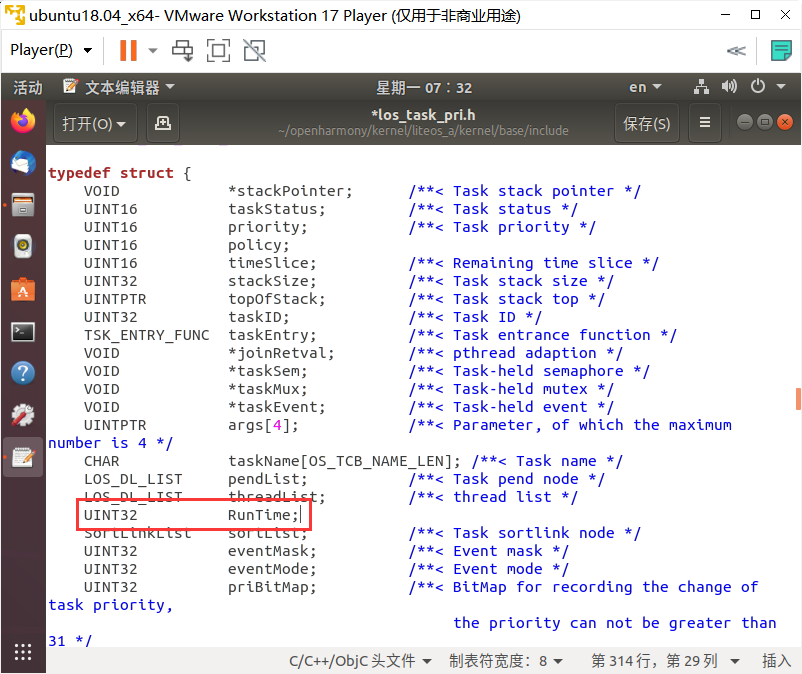
1. **修改LosTaskCB结构体**

打开openharmony/kernel/liteos\_a/kernel/base/include/los\_task\_pri.h

找到LosTaskCB结构体



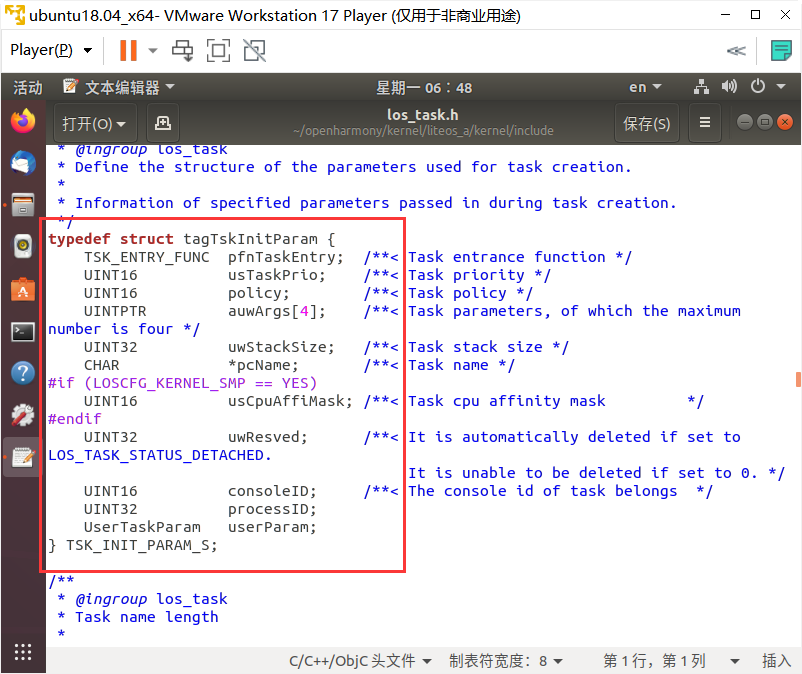
在pendList变量下添加一条UINT32 RunTime



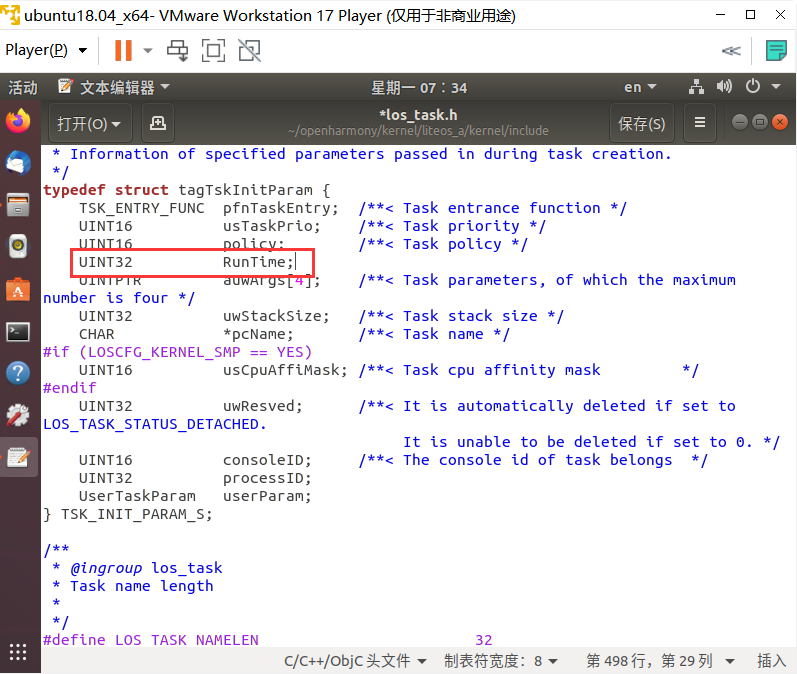
1. **修改TSK\_INIT\_PARAM\_S结构体**

打开openharmony/kernel/liteos\_a/kernel/include/los\_task.h

找到TSK\_INIT\_PARAM\_S结构体



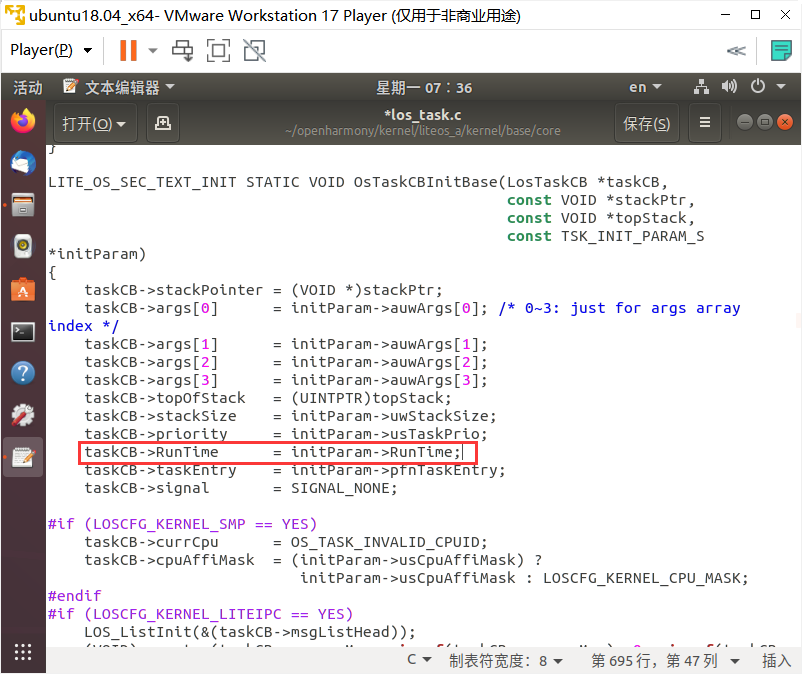
在policy下添加一条UINT32 RunTime



1. **修改任务赋值（初始化）函数**

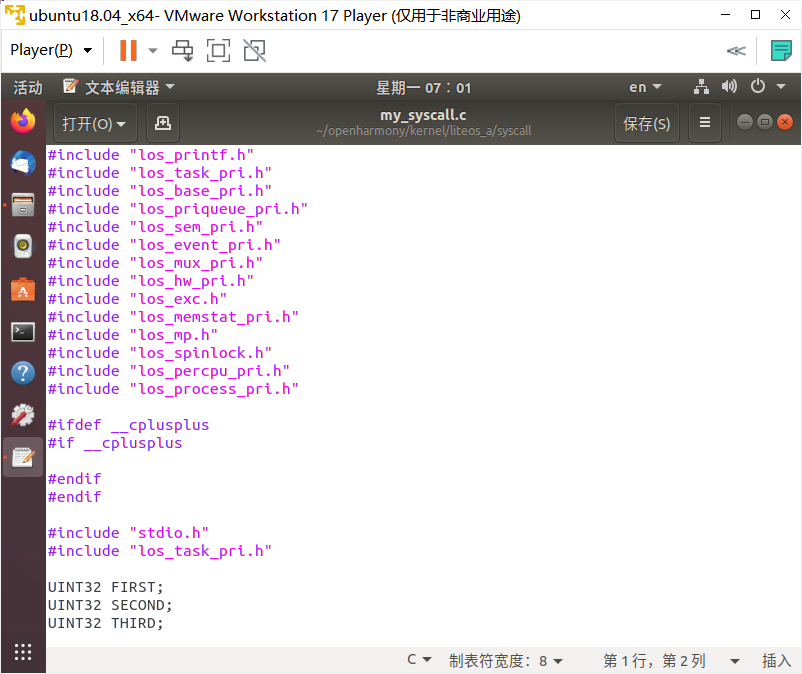
打开openharmony/kernel/liteos\_a/kernel/base/core/los\_task.c

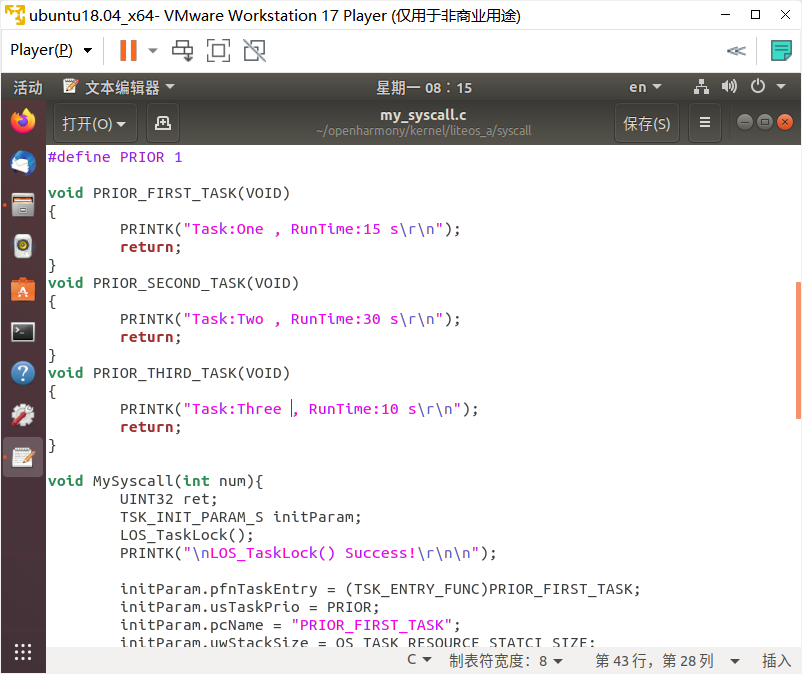
在taskCB->priority下添加一条：taskCB->RunTime = initParam->RunTime



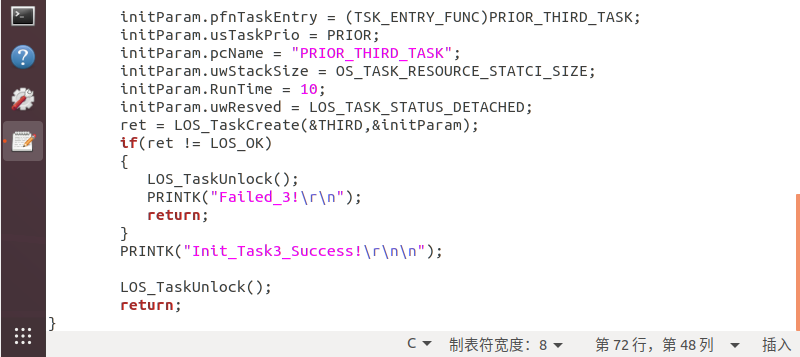
1. **编写验证测试程序**

更改Project1中制作的openharmony/kernel/liteos\_a/syscall/my\_syscall.c





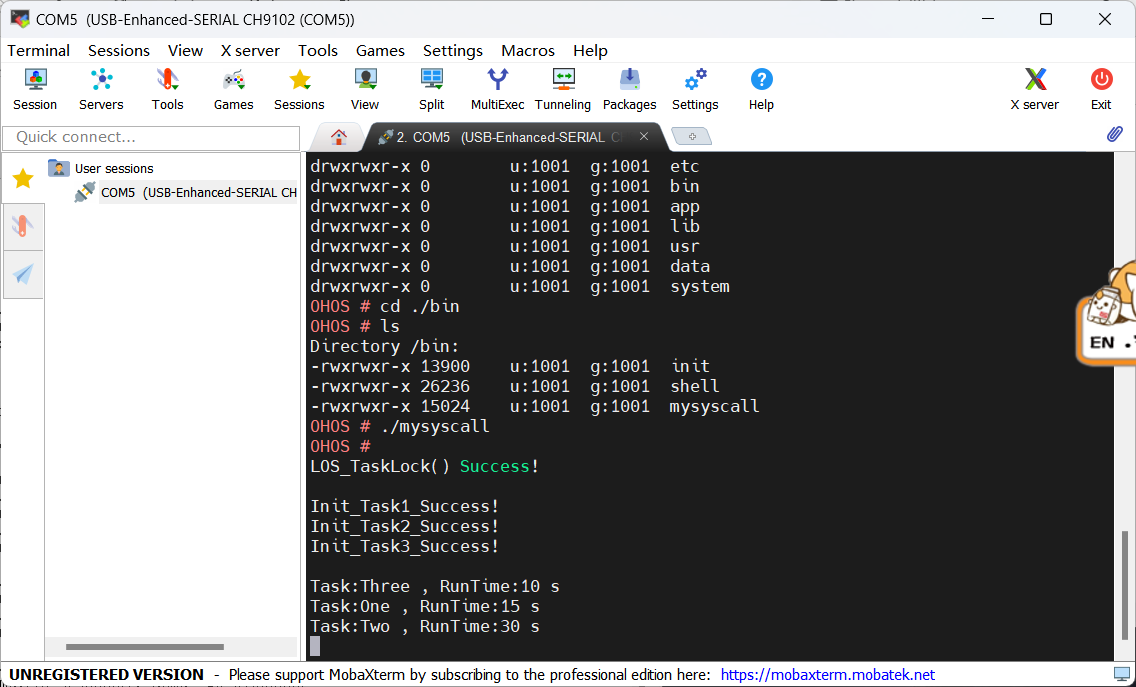




1. **开发板测试**

测试结果如下文

# 实验结果



# 实验分析

主要实现思路如下：

1.在TCB中插入一个RunTime变量用于记录任务的运行时间

2.修改优先级队列函数使得运行时间较低的任务优先

实验结果：

实验结果中的2-4行代表任务的生成顺序

5-8行为任务的执行顺序

可以看到10s运行时间的Task3优先运行，15s的Task1次之，最后是30s的Task2，符合运行时间低优先运行

因此实验成功

# 实验总结

通过本次实验，我深入理解了LiteOS的调度器实现和代码结构，了解了不同的调度策略对同一批任务的处理运行时间的影响以及他们的优劣。该实验不仅加深了我对LiteOS调度器的理解，还使我学到了很多关于系统修改和优化的经验。

# 参考文献

1.[美]William Stallings著，陈向群，陈 渝 等译《操作系统——精髓与原理设计（第八版）》

# 附录

1. OsPriQueueEnqueue方法

VOID OsPriQueueEnqueue(LOS\_DL\_LIST \*priQueueList, UINT32 \*bitMap, LOS\_DL\_LIST \*priqueueItem, UINT32 priority)

{

if (LOS\_ListEmpty(&priQueueList[priority])) {

\*bitMap |= PRIQUEUE\_PRIOR0\_BIT >> priority;

LOS\_ListTailInsert(&priQueueList[priority], priqueueItem);

}else{

LOS\_DL\_LIST \*currentItem = priQueueList[priority].pstNext;

LOS\_DL\_LIST \*prevItem = &priQueueList[priority];

while(currentItem != &priQueueList[priority]){

UINT32 thisTime = \*(UINT32\*)(priqueueItem + 1);

UINT32 frontTime = \*(UINT32\*)(currentItem + 1);

if(thisTime < frontTime){

priqueueItem->pstNext = currentItem;

prevItem->pstNext = priqueueItem;

if(prevItem->pstPrev){

priqueueItem->pstPrev = prevItem;

}

if(currentItem->pstPrev){

currentItem->pstPrev = priqueueItem;

}

return;

}

prevItem = currentItem;

currentItem = currentItem->pstNext;

}

prevItem->pstNext = priqueueItem;

priqueueItem->pstNext = &priQueueList[priority];

priqueueItem->pstPrev = prevItem;

}

}

1. 验证测试程序

#include "los\_printf.h"

#include "los\_task\_pri.h"

#include "los\_base\_pri.h"

#include "los\_priqueue\_pri.h"

#include "los\_sem\_pri.h"

#include "los\_event\_pri.h"

#include "los\_mux\_pri.h"

#include "los\_hw\_pri.h"

#include "los\_exc.h"

#include "los\_memstat\_pri.h"

#include "los\_mp.h"

#include "los\_spinlock.h"

#include "los\_percpu\_pri.h"

#include "los\_process\_pri.h"

#ifdef \_\_cplusplus

#if \_\_cplusplus

#endif

#endif

#include "stdio.h"

#include "los\_task\_pri.h"

UINT32 FIRST;

UINT32 SECOND;

UINT32 THIRD;

#define PRIOR 1

void PRIOR\_FIRST\_TASK(VOID)

{

PRINTK("Task:One , RunTime:15 s\r\n");

return;

}

void PRIOR\_SECOND\_TASK(VOID)

{

PRINTK("Task:Two , RunTime:30 s\r\n");

return;

}

void PRIOR\_THIRD\_TASK(VOID)

{

PRINTK("Task:Three , RunTime:10 s\r\n");

return;

}

void MySyscall(int num){

UINT32 ret;

TSK\_INIT\_PARAM\_S initParam;

LOS\_TaskLock();

PRINTK("\nLOS\_TaskLock() Success!\r\n\n");

initParam.pfnTaskEntry = (TSK\_ENTRY\_FUNC)PRIOR\_FIRST\_TASK;

initParam.usTaskPrio = PRIOR;

initParam.pcName = "PRIOR\_FIRST\_TASK";

initParam.uwStackSize = OS\_TASK\_RESOURCE\_STATCI\_SIZE;

initParam.RunTime = 15;

initParam.uwResved = LOS\_TASK\_STATUS\_DETACHED;

ret = LOS\_TaskCreate(&FIRST,&initParam);

if(ret != LOS\_OK)

{

LOS\_TaskUnlock();

PRINTK("Failed\_1!\r\n");

return;

}

PRINTK("Init\_Task1\_Success!\r\n");

initParam.pfnTaskEntry = (TSK\_ENTRY\_FUNC)PRIOR\_SECOND\_TASK;

initParam.usTaskPrio = PRIOR;

initParam.pcName = "PRIOR\_SECOND\_TASK";

initParam.uwStackSize = OS\_TASK\_RESOURCE\_STATCI\_SIZE;

initParam.RunTime = 30;

initParam.uwResved = LOS\_TASK\_STATUS\_DETACHED;

ret = LOS\_TaskCreate(&SECOND,&initParam);

if(ret != LOS\_OK)

{

LOS\_TaskUnlock();

PRINTK("Failed\_2!\r\n");

return;

}

PRINTK("Init\_Task2\_Success!\r\n");

initParam.pfnTaskEntry = (TSK\_ENTRY\_FUNC)PRIOR\_THIRD\_TASK;

initParam.usTaskPrio = PRIOR;

initParam.pcName = "PRIOR\_THIRD\_TASK";

initParam.uwStackSize = OS\_TASK\_RESOURCE\_STATCI\_SIZE;

initParam.RunTime = 10;

initParam.uwResved = LOS\_TASK\_STATUS\_DETACHED;

ret = LOS\_TaskCreate(&THIRD,&initParam);

if(ret != LOS\_OK)

{

LOS\_TaskUnlock();

PRINTK("Failed\_3!\r\n");

return;

}

PRINTK("Init\_Task3\_Success!\r\n\n");

LOS\_TaskUnlock();

return;

}