厦門大學

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

1 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

2 实验环境

主机： Windows 11

虚拟机： Ubuntu 18.04

开发板： IMAX6ULL MINI

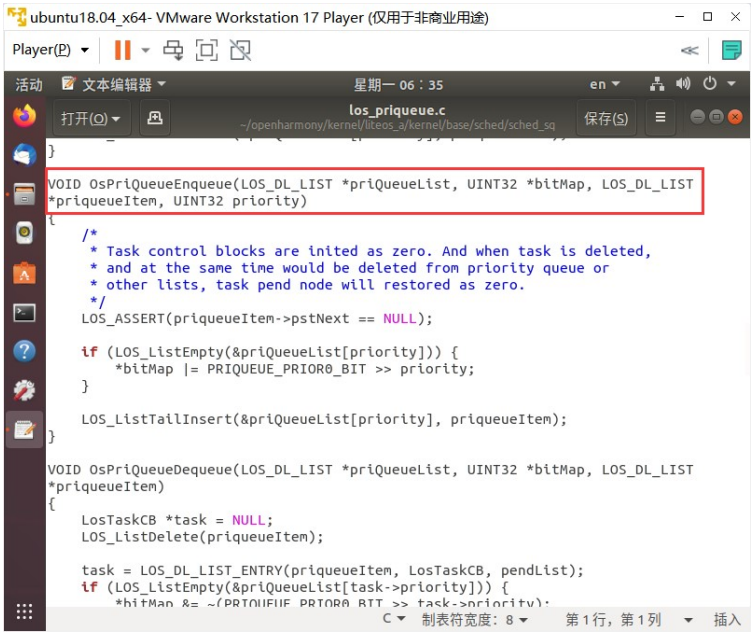
终端： MobaXterm

3 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

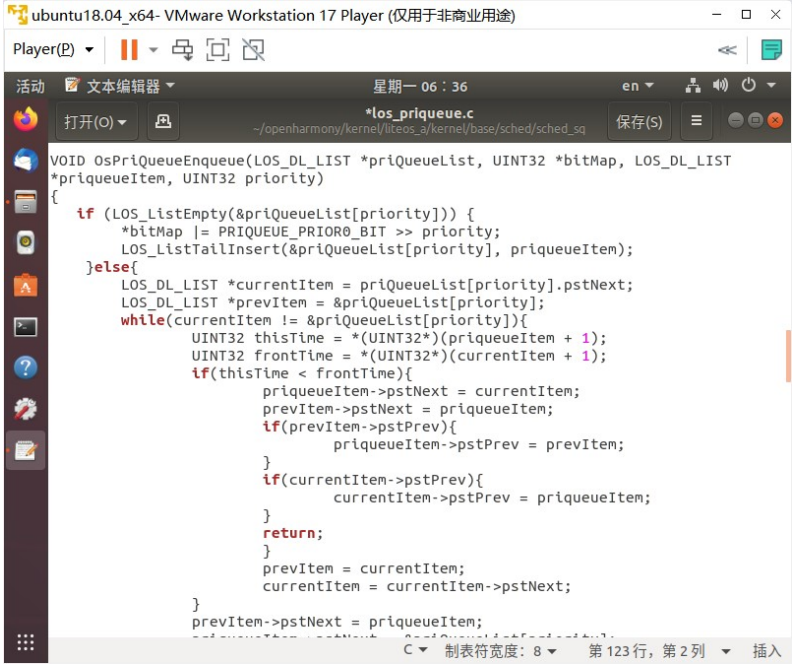


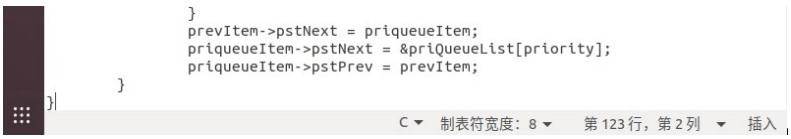
该方法就是优先级队列函数

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

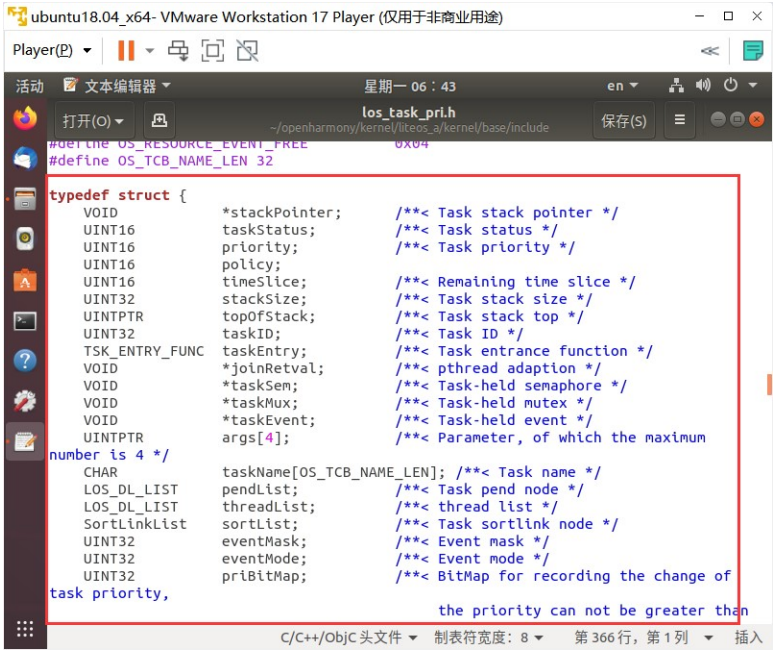




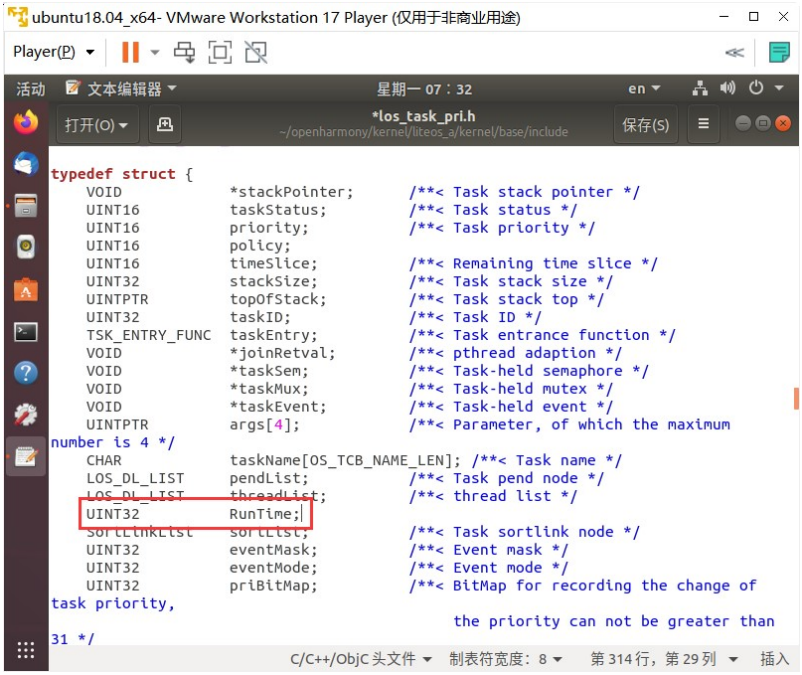
2. 修改 LosTaskCB 结构体

打开 openharmony/kernel/liteos_a/kernel/base/include/los_task_pri.h

找到 LosTaskCB 结构体



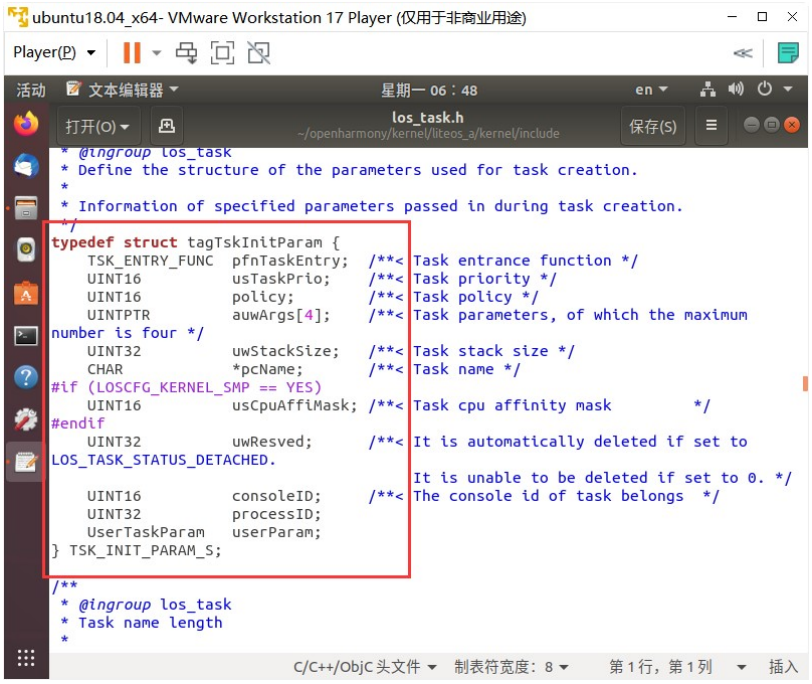
在 pendList 变量下添加一条 UINT32　　RunTime



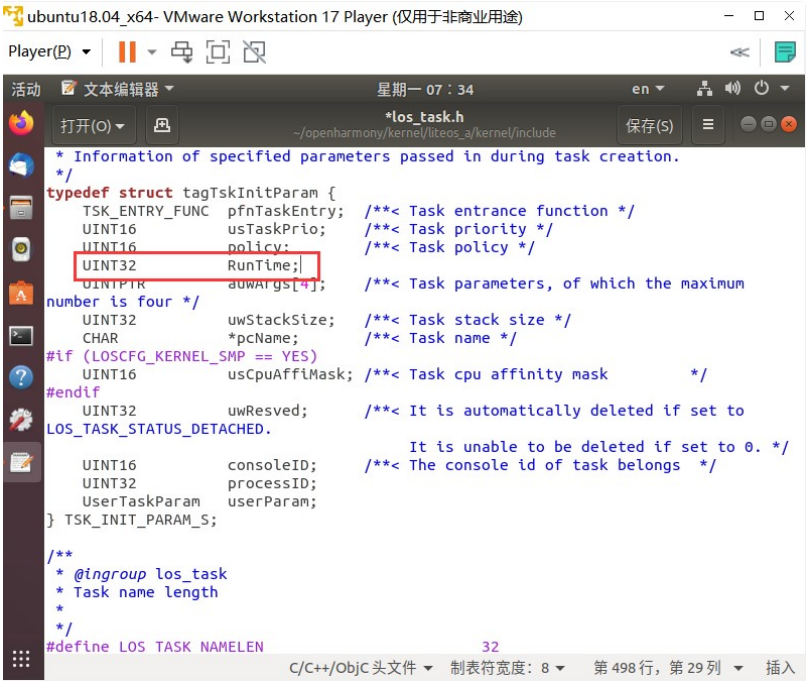
3. 修改 TSK_INIT_PARAM_S 结构体

打开 openharmony/kernel/liteos_a/kernel/include/los_task.h

找到 TSK_INIT_PARAM_S 结构体



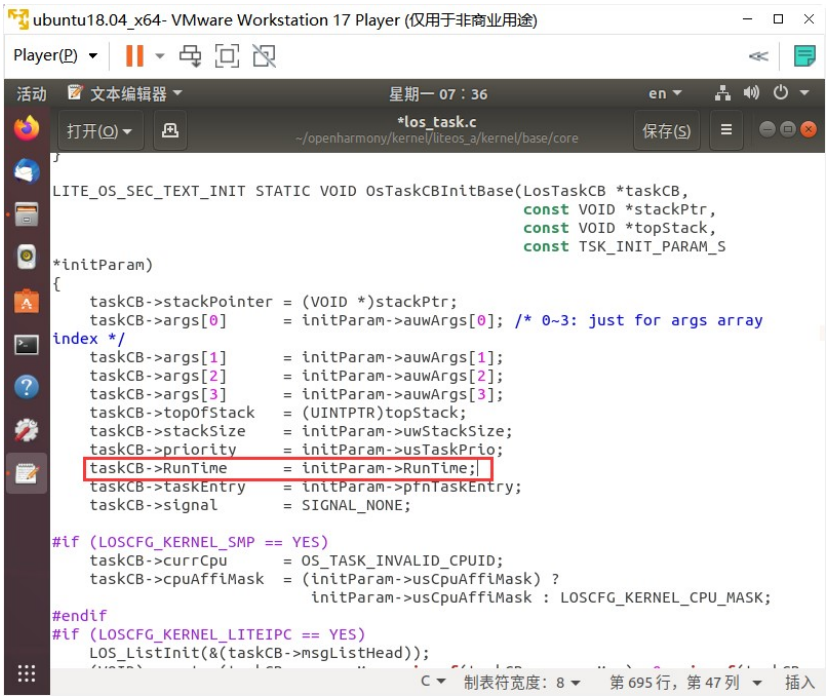
在 policy 下添加一条 UINT32 RunTime



4. 修改任务赋值（初始化）函数

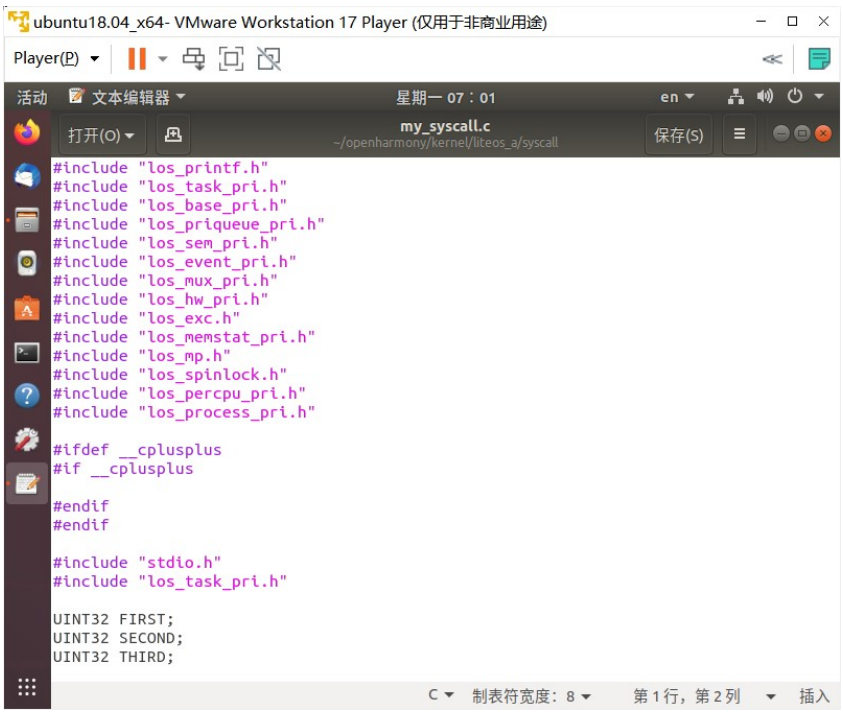
打开 openharmony/kernel/liteos_a/kernel/base/core/los_task.c

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



5. 编写验证测试程序

更改 Project1 中制作的 openharmony/kernel/liteos_a/syscall/my_syscall.c



ubuntu18.04_x64- VMware Workstation 17 Player (仅用于非商业用途)

Player(P)

活动 文本编辑器 星期一 08：15 en

打开(O) my_syscall.c ~/openharmony/kernel/liteos_a/syscall 保存(S)

```c
#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;
```

C 制表符宽度：8 第 43 行，第 28 列 插入

ubuntu18.04_x64- VMware Workstation 17 Player (仅用于非商业用途)

Player(P)

活动 文本编辑器 星期一 07：02 en

打开(O) my_syscall.c ~/openharmony/kernel/liteos_a/syscall 保存(S)

```c
        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");
```

C 制表符宽度：8 第 1 行，第 2 列 插入

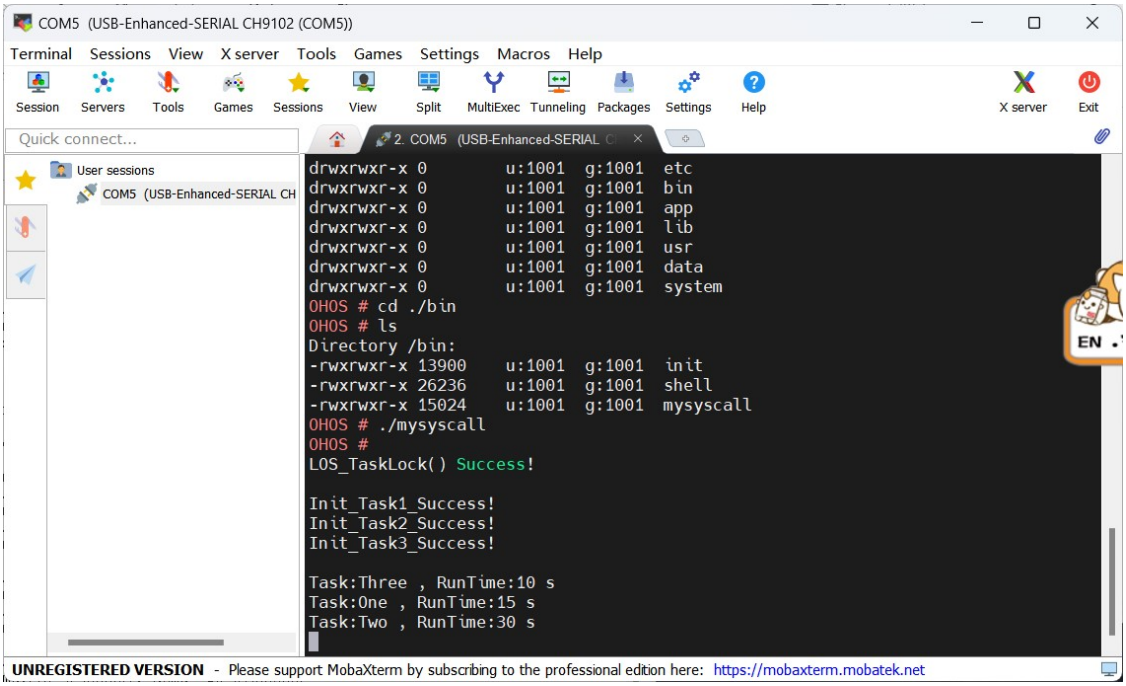
```
        initParam.pfnTaskEntry = (TSK_ENTRY_FUNC)PRIOR_THIRD_TASK;
        initParam.usTaskPrio = PRIOR;
        initParam.pcName = "PRIOR_THIRD_TASK";
        initParam.uwStackSize = OS_TASK_RESOURCE_STATCI_SIZE;
        initParam.RunTime = 10;
        initParam.uwResved = LOS_TASK_STATUS_DETACHED;
        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;
}
```

C ▾ 制表符宽度：8 ▾ 第 72 行，第 48 列 ▾ 插入

6. 开发板测试

测试结果如下文

4 实验结果



5 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

6 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。

7 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）》

8 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;
    }
```

}

2. 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;
}
```