# Project2 调度算法

姓名：任宇　　学号：33920212204567

## 一、 实验目的

对鸿蒙 LiteOS 的调度算法进行改进或添加一个新的调度算法。

## 二、 实验环境

- 操作系统：
  - 主机：Windows 10
  - 虚拟机：Ubuntu 18.04
- 开发板：IMAX6ULL MIN
- 文件传输工具：FileZilla
- 终端工具：MobaXterm

## 三、 实验内容

### 1. 修改优先级队列入队函数：

通过阅读Liteos_a的源码可以发现，不管是线程的优先级队列还是任务的优先级队列，其底层都是调用文件openharmony/kernel/liteos_a/kernel/base/include/los_queue_pri.h文件中声明的OsPriQueueEnqueue函数。如图所示：

```
/*VOID OsPriQueueEnqueue(LOS_DL_LIST *priQueueList, UINT32 *bitMap, LOS_DL_LIST *priqueueItem, UINT32 priority)
{
    LOS_ASSERT(priqueueItem->pstNext == NULL);

    if (LOS_ListEmpty(&priQueueList[priority])) {
        *bitMap |= PRIQUEUE_PRIOR0_BIT >> priority;
    }

    LOS_ListTailInsert(&priQueueList[priority], priqueueItem);
}*/
```

这个函数的功能即：当对应优先级的队列为空时将待入队元素作为队首，而当对应优先级队列不为空时则直接将待入队元素尾插进队列。

为了实现结合优先级和短作业优先的调度算法，我们需要对入队函数进行修改，使其能够支持在同等优先级下，短作业优先执行这一特点，即修改openharmony/kernel/liteos_a/kernel/base/sched/sched_sq/los_priqueue.c文件中OsPriQueueEnqueue函数的实现，如图所示：

```
VOID OsPriQueueEnqueue(LOS_DL_LIST *priQueueList, UINT32 *bitMap, LOS_DL_LIST *priqueueItem, UINT32 priority)
{

    LOS_ASSERT(priqueueItem->pstNext == NULL);

    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 estimateRuntime = *(UINT32*)(priqueueItem+1);
              UINT32 curEstimateRuntime = *(UINT32*)(currentItem+1);
              if(estimateRuntime < curEstimateRuntime)
              {
                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;
     }
}
```

需要注意的是，由于没有在函数参数列表中添加参数estimateRunTime，这里的代码是通过直接访问内存实现的，这与结构体定义中变量声明的顺序有密切关系。

## 2. 修改 LosTaskCB 结构体：

通过阅读源码可以发现，在 Liteos_a 中线程和任务的调度算法是非常相似的，因此在本次实验中，以修改任务的调度算法为例子。

LosTaskCB 的结构体定义位于 openharmony/kernel/liteos_a/kernel/base/include/los_task_pri.h，为其添加 estimateRunTime 属性，需要注意的是，由于第一步中入队函数的修改，这里的 estimateRunTime 需要添加在 pendList 下方：

```
typedef struct {
    VOID            *stackPointer;      /**< Task stack pointer */
    UINT16          taskStatus;         /**< Task status */
    UINT16          priority;           /**< Task priority */
    UINT16          policy;
    UINT16          timeSlice;          /**< Remaining time slice */
    UINT32          stackSize;          /**< Task stack size */
    UINTPTR         topOfStack;         /**< Task stack top */
    UINT32          taskID;             /**< Task ID */
    TSK_ENTRY_FUNC  taskEntry;          /**< Task entrance function */
    VOID            *joinRetval;        /**< pthread adaption */
    VOID            *taskSem;           /**< Task-held semaphore */
    VOID            *taskMux;           /**< Task-held mutex */
    VOID            *taskEvent;         /**< Task-held event */
    UINTPTR         args[4];            /**< Parameter, of which the maximum number is 4 */
    CHAR            taskName[OS_TCB_NAME_LEN]; /**< Task name */
    LOS_DL_LIST     pendList;           /**< Task pend node */
    UINT32          estimateRunTime;     /**< Task estimate runtime */
    LOS_DL_LIST     threadList;         /**< thread list */
    SortLinkList    sortList;           /**< Task sortlink node */
    UINT32          eventMask;          /**< Event mask */
    UINT32          eventMode;          /**< Event mode */
    UINT32          priBitMap;          /**< BitMap for recording the change of task priority,
                                             the priority can not be greater than 31 */
    INT32           errorNo;            /**< Error Num */
```

## 3. 修改 TSK_INIT_PARAM_S 结构体：

TSK_INIT_PARAM_S 用于在创建任务时指定任务初始化的参数。位于 openharmony/kernel/liteos_a/kernel/include/los_task.h 文件中，为其增加 usEstimateRunTime 属性，以便在初始化时设定预计运行时间的值，修改如图所示：

```
typedef struct tagTskInitParam {
    TSK_ENTRY_FUNC  pfnTaskEntry;  /**< Task entrance function */
    UINT16          usTaskPrio;    /**< Task priority */
    UINT16          policy;        /**< Task policy */
    UINT32          usEstimateRunTime;
    UINTPTR         auwArgs[4];    /**< Task parameters, of which the maximum number is four */
    UINT32          uwStackSize;   /**< Task stack size */
    CHAR            *pcName;       /**< Task name */
#if (LOSCFG_KERNEL_SMP == YES)
    UINT16          usCpuAffiMask; /**< Task cpu affinity mask        */
#endif
    UINT32          uwResved;      /**< It is automatically deleted if set to LOS_TASK_STATUS_DETACHED.
                                        It is unable to be deleted if set to 0. */
    UINT16          consoleID;     /**< The console id of task belongs  */
    UINT32          processID;
    UserTaskParam   userParam;
} TSK_INIT_PARAM_S;
```

## 4. 修改任务初始化函数：

修改 penharmony/kernel/liteos_a/kernel/base/core/los_task.c 中的 OsTaskCBInitBase 函数，使其支持对预计运行时间的初始化：

```
LITE_OS_SEC_TEXT_INIT STATIC VOID OsTaskCBInitBase(LosTaskCB *taskCB,
                                                   const VOID *stackPtr,
                                                   const VOID *topStack,
                                                   const TSK_INIT_PARAM_S *initParam)
{
    taskCB->stackPointer = (VOID *)stackPtr;
    taskCB->args[0]      = initParam->auwArgs[0]; /* 0~3: just for args array index */
    taskCB->args[1]      = initParam->auwArgs[1];
    taskCB->args[2]      = initParam->auwArgs[2];
    taskCB->args[3]      = initParam->auwArgs[3];
    taskCB->topOfStack   = (UINTPTR)topStack;
    taskCB->stackSize    = initParam->uwStackSize;
    taskCB->priority     = initParam->usTaskPrio;
    taskCB->estimateRunTime     = initParam->usEstimateRunTime;
    taskCB->taskEntry    = initParam->pfnTaskEntry;
    taskCB->signal       = SIGNAL_NONE;
```

## 5. 编写程序实例以验证对调度算法的修改是否成功：

在这一步中，编写一个程序测试修改是否成功。在测试程序中，设置 3 个不同的优先级，其中高优先级任务 1 个、中优先级任务 3 个、低优先级任务 1 个。在 3 个中优先级任务中分别设置 3 个不同的预计运行时间来验证短作业优先。最后程序会按新的调度算法调度 5 个不同的任务，观察程序输出以验证修改是否成功。为了方便调用，将测试程序写在 Project1 新增的系统调用当中。

```
void Example_PRIOR_HI_Task(VOID)
{

    PRINTK("Enter PRIOR_HI_Task Handler.\r\n");
    PRINTK("PRIOR_HI_Task LOS_TaskDelete Success.\r\n");
    return;
}

void Example_PRIOR_LO_Task(VOID)
{

    PRINTK("Enter PRIOR_LO_Task Handler.\r\n");
    PRINTK("PRIOR_LO_Task LOS_TaskDelete Success.\r\n");
    return;
}

void Example_PRIOR_MID_TaskHi(VOID)
{

    PRINTK("Enter PRIOR_MID_TaskHi Handler.\r\n");
    PRINTK("PRIOR_MID_TaskLo LOS_TaskDelete Success.\r\n");
    return;
}

void Example_PRIOR_MID_TaskMid(VOID)
{

    PRINTK("Enter PRIOR_MID_TaskMid Handler.\r\n");
    PRINTK("PRIOR_MID_TaskMid LOS_TaskDelete Success.\r\n");
    return;
}
```

```c
void Example_PRIOR_MID_TaskLo(VOID)
{
    PRINTK("Enter PRIOR_MID_TaskLo Handler.\r\n");
    PRINTK("PRIOR_MID_TaskLo LOS_TaskDelete Success.\r\n");
    return;
}


void SysNewSyscallSample(int num)
{
    UINT32 ret;
    TSK_INIT_PARAM_S initParam;

    /* 锁任务调度，防止新创建的任务比本任务高而发生调度 */
    LOS_TaskLock();

    PRINTK("LOS_TaskLock() Success!\r\n");


    initParam.pfnTaskEntry = (TSK_ENTRY_FUNC)Example_PRIOR_MID_TaskMid;
    initParam.usTaskPrio = TSK_PRIOR_MID;
    initParam.pcName = "PRIOR_MID_TaskMid";
    initParam.uwStackSize = OS_TASK_RESOURCE_STATCI_SIZE;
    initParam.uwResved   = LOS_TASK_STATUS_DETACHED;
    initParam.usEstimateRunTime   = 10;
    ret = LOS_TaskCreate(&g_PRIOR_MID_taskMidId, &initParam);
    if (ret != LOS_OK) {
        LOS_TaskUnlock();

        PRINTK("Example_PRIOR_MID_TaskMid create Failed!\r\n");
        return;
    }
    PRINTK("Example_PRIOR_MID_TaskMid create Success!\r\n");

    initParam.pfnTaskEntry = (TSK_ENTRY_FUNC)Example_PRIOR_HI_Task;
    initParam.usTaskPrio = TSK_PRIOR_HI;
    initParam.pcName = "PRIOR_HI_Task";
    initParam.uwStackSize = OS_TASK_RESOURCE_STATCI_SIZE;
    initParam.uwResved   = LOS_TASK_STATUS_DETACHED;
    initParam.usEstimateRunTime   = 10;
    ret = LOS_TaskCreate(&g_PRIOR_HI_taskId, &initParam);
    if (ret != LOS_OK) {
        LOS_TaskUnlock();

        PRINTK("Example__PRIOR_HI_Task create Failed!\r\n");
        return;
    }
    PRINTK("Example_PRIOR_HI_Task create Success!\r\n");


    initParam.pfnTaskEntry = (TSK_ENTRY_FUNC)Example_PRIOR_LO_Task;
    initParam.usTaskPrio = TSK_PRIOR_LO;
    initParam.pcName = "PRIOR_LO_Task";
    initParam.uwStackSize = OS_TASK_RESOURCE_STATCI_SIZE;
    initParam.uwResved   = LOS_TASK_STATUS_DETACHED;
    initParam.usEstimateRunTime   = 10;
    ret = LOS_TaskCreate(&g_PRIOR_LO_taskId, &initParam);
    if (ret != LOS_OK) {
        LOS_TaskUnlock();

        PRINTK("Example__PRIOR_LO_Task create Failed!\r\n");
        return;
    }
    PRINTK("Example_PRIOR_LO_Task create Success!\r\n");

    initParam.pfnTaskEntry = (TSK_ENTRY_FUNC)Example_PRIOR_MID_TaskHi;
    initParam.usTaskPrio = TSK_PRIOR_MID;
    initParam.pcName = "PRIOR_MID_TaskHi";
    initParam.uwStackSize = OS_TASK_RESOURCE_STATCI_SIZE;
    initParam.uwResved   = LOS_TASK_STATUS_DETACHED;
    initParam.usEstimateRunTime   = 5;
    ret = LOS_TaskCreate(&g_PRIOR_MID_taskHiId, &initParam);
    if (ret != LOS_OK) {
        LOS_TaskUnlock();

        PRINTK("Example_PRIOR_MID_TaskHi create Failed!\r\n");
        return;
    }
    PRINTK("Example_PRIOR_MID_TaskHi create Success!\r\n");
```

```
    initParam.pfnTaskEntry = (TSK_ENTRY_FUNC)Example_PRIOR_MID_TaskLo;
    initParam.usTaskPrio = TSK_PRIOR_MID;
    initParam.pcName = "PRIOR_MID_TaskLo";
    initParam.uwStackSize = OS_TASK_RESOURCE_STATCI_SIZE;
    initParam.uwResved   = LOS_TASK_STATUS_DETACHED;
    initParam.usEstimateRunTime   = 30;
    ret = LOS_TaskCreate(&g_PRIOR_MID_taskLoId, &initParam);
    if (ret != LOS_OK) {
        LOS_TaskUnlock();

        PRINTK("Example_PRIOR_MID_TaskLo create Failed!\r\n");
        return;
    }
    PRINTK("Example_PRIOR_MID_TaskLo create Success!\r\n");

    /* 解锁任务调度，此时会发生任务调度，执行就绪队列中最高优先级任务 */
    LOS_TaskUnlock();

    return;
}
```

## 6. 调用并验证：

编写测试文件 test.c 调用新的系统调用：

```
#include <stdio.h>
#include <syscall.h>

void newSyscallSample(int num)
{
       syscall(SYS_new_syscall_sample,num);
  return;
}


int main(void)
{
  newSyscallSample(0);
  return 0;
}
```

编译 test 程序，并将其加载到开发板中：

```
book@ry-virtual-machine:~/openharmony/kernel/liteos_a$ cd /home/book/doc_and_source_for_openharmony/apps/hello
cd /home/book/openharmony/kernel/liteos_a/out/imx6ull/
mkfs.jffs2 -s 0x10000 -e 0x10000 -d rootfs -o rootfs.jffs2book@ry-virtual-machine:~/doc_and_source_for_openharmony/apps/hello$ clang -target arm-liteos --sysr
t hello.c/book/openharmony/prebuilts/lite/sysroot/ \-o test
hello.c:6:8: warning: implicit declaration of function 'syscall' is invalid in C99 [-Wimplicit-function-declaration]
      syscall(SYS_new_syscall_sample,num);
      ^
1 warning generated.
book@ry-virtual-machine:~/doc_and_source_for_openharmony/apps/hello$ cd /home/book/doc_and_source_for_openharmony/apps/hello
book@ry-virtual-machine:~/doc_and_source_for_openharmony/apps/hello$ cp test /home/book/openharmony/kernel/liteos_a/out/imx6ull/rootfs/bin
book@ry-virtual-machine:~/doc_and_source_for_openharmony/apps/hello$ cd /home/book/openharmony/kernel/liteos_a/out/imx6ull/
book@ry-virtual-machine:~/openharmony/kernel/liteos_a/out/imx6ull$ mkfs.jffs2 -s 0x10000 -e 0x10000 -d rootfs -o rootfs.jffs2
```

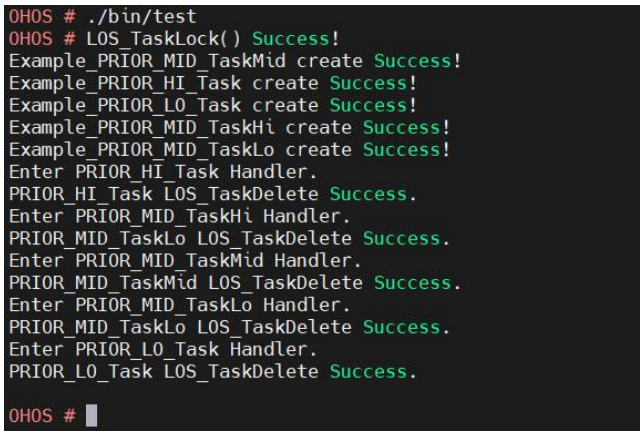
在开发板中运行 test 程序，观察任务创建顺序以及任务调用顺序可见程序成功执行修改后的调度算法，实验成功：

```
OHOS # ./bin/test
OHOS # LOS_TaskLock() Success!
Example_PRIOR_MID_TaskMid create Success!
Example_PRIOR_HI_Task create Success!
Example_PRIOR_LO_Task create Success!
Example_PRIOR_MID_TaskHi create Success!
Example_PRIOR_MID_TaskLo create Success!
Enter PRIOR_HI_Task Handler.
PRIOR_HI_Task LOS_TaskDelete Success.
Enter PRIOR_MID_TaskHi Handler.
PRIOR_MID_TaskLo LOS_TaskDelete Success.
Enter PRIOR_MID_TaskMid Handler.
PRIOR_MID_TaskMid LOS_TaskDelete Success.
Enter PRIOR_MID_TaskLo Handler.
PRIOR_MID_TaskLo LOS_TaskDelete Success.
Enter PRIOR_LO_Task Handler.
PRIOR_LO_Task LOS_TaskDelete Success.

OHOS #
```

## 四、 实验结果

本实验旨在修改鸿蒙 Liteos 中的调度算法。通过对鸿蒙系统的研究，我确定了 Liteos_a 中与调度算法相关的文件，并分析它们各自的作用。修改后的调度算法实现了在优先级排序的基础上，若任务处于同一优先级，则短作业优先。实验的主要步骤包括设计、修改、编码和测试调度算法。在测试环节，新的调度算法成功实现了预期的功能，如下图所示：




## 五、 实验分析

本次 Project 是实现了结合优先级和短作业优先的调度算法。在这种调度算法中，任务根据它们的优先级和估计运行时间被调度执行。这种方法的优点和缺点如下：

**优点:**

1. 响应时间：对于短作业来说，响应时间可以很快，因为它们会被更快地调度执行。

2. 吞吐量：吞吐量通常较高，因为短作业能够更快完成，系统可以在单位时间内完成更多的作业。

3. 优先级考虑：可以处理紧急任务，确保高优先级的任务得

到及时处理。

**缺点:**

1. 饥饿现象：长作业可能会遭受无限期的延迟，因为总是有更短或更高优先级的作业在前面。

2. 运行时间预测：准确预测作业的运行时间是挑战性的，不准确的估计可能导致调度效率低下。

因此该调度算法还有可以继续提升的地方，比如说结合老化（aging）技术，以避免饥饿问题。

## 六、 实验总结

本次实验目的是修改鸿蒙 Liteos_a 的优先级调度算法，以引入短作业优先（SJF）的特性，从而提高调度算法在特定场景下的性能。在这个实验中，我们首先审视了标准的优先级调度机制，它依据任务的优先级来分配处理器时间，通常优先服务高优先级任务。我们的目标是在保留优先级概念的基础上，增加对任务长度的考虑，以此来优化对短任务的处理效率。

实验过程包括设计算法、修改现有调度程序代码、实施新调度策略，并通过测试来评估其效果。通过本实验，可以发现:当任务长度变得可预测时，调度器能够有效地减少平均等待时间和提高吞吐量。然而，尽管引入 SJF 机制确实可以在某些情况下改善性能，但它也需要精心的设计来确保系统的公平性和效率。

最后，我认识到调度算法设计是一个平衡各种不同需求的过程，而且往往需要根据具体的应用场景来调整和优化。

## 七、 参考文献


1. 开发指导 LiteOS 内核 任务 华为云 (huaweicloud.com)
2. LiteOS 内核源码分析系列六 -任务及调度（2）-任务 LOS_Task-云社区-华为云 (huaweicloud.com)


## 八、 附录

1. syscall_demo.c ：

```c
#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
extern "C" {
#endif /* __cpluscplus */
#endif /* __cpluscplus */

#include "stdio.h"
#include "los_task_pri.h"

UINT32 g_PRIOR_MID_taskHiId;
UINT32 g_PRIOR_MID_taskLoId;
UINT32 g_PRIOR_MID_taskMidId;
UINT32 g_PRIOR_HI_taskId;
UINT32 g_PRIOR_LO_taskId;
#define TSK_PRIOR_HI 1
#define TSK_PRIOR_MID 2
#define TSK_PRIOR_LO 3

```

```
34.void Example_PRIOR_HI_Task(VOID)
35.{
36.
37.    PRINTK("Enter PRIOR_HI_Task Handler.\r\n");
38.    PRINTK("PRIOR_HI_Task LOS_TaskDelete Success.\r\n");
39.    return;
40.}
41.
42.void Example_PRIOR_LO_Task(VOID)
43.{
44.
45.    PRINTK("Enter PRIOR_LO_Task Handler.\r\n");
46.    PRINTK("PRIOR_LO_Task LOS_TaskDelete Success.\r\n");
47.    return;
48.}
49.
50.void Example_PRIOR_MID_TaskHi(VOID)
51.{
52.
53.    PRINTK("Enter PRIOR_MID_TaskHi Handler.\r\n");
54.    PRINTK("PRIOR_MID_TaskLo LOS_TaskDelete Success.\r\n");
55.    return;
56.}
57.
58.void Example_PRIOR_MID_TaskMid(VOID)
59.{
60.
61.    PRINTK("Enter PRIOR_MID_TaskMid Handler.\r\n");
62.    PRINTK("PRIOR_MID_TaskMid LOS_TaskDelete Success.\r\n");
63.    return;
64.}
65.
66.void Example_PRIOR_MID_TaskLo(VOID)
67.{
68.    PRINTK("Enter PRIOR_MID_TaskLo Handler.\r\n");
69.    PRINTK("PRIOR_MID_TaskLo LOS_TaskDelete Success.\r\n");
70.    return;
71.}
72.

73.void SysNewSyscallSample(int num)
74.{
75.    UINT32 ret;
76.    TSK_INIT_PARAM_S initParam;
77.
```

```
78.     /* 锁任务调度，防止新创建的任务比本任务高而发生调度 */
79.     LOS_TaskLock();
80.
81.     PRINTK("LOS_TaskLock() Success!\r\n");
82.
83.     initParam.pfnTaskEntry =
   (TSK_ENTRY_FUNC)Example_PRIOR_MID_TaskMid;
84.     initParam.usTaskPrio = TSK_PRIOR_MID;
85.     initParam.pcName = "PRIOR_MID_TaskMid";
86.     initParam.uwStackSize = OS_TASK_RESOURCE_STATCI_SIZE;
87.     initParam.uwResved   = LOS_TASK_STATUS_DETACHED;
88.     initParam.usEstimateRunTime   = 10;
89.     ret = LOS_TaskCreate(&g_PRIOR_MID_taskMidId, &initParam);
90.     if (ret != LOS_OK) {
91.         LOS_TaskUnlock();
92.
93.         PRINTK("Example_PRIOR_MID_TaskMid create Failed!\r\n");
94.         return;
95.     }
96.     PRINTK("Example_PRIOR_MID_TaskMid create Success!\r\n");
97.
98.
99.     initParam.pfnTaskEntry =
   (TSK_ENTRY_FUNC)Example_PRIOR_HI_Task;
100.        initParam.usTaskPrio = TSK_PRIOR_HI;
101.        initParam.pcName = "PRIOR_HI_Task";
102.        initParam.uwStackSize = OS_TASK_RESOURCE_STATCI_SIZE;
103.        initParam.uwResved   = LOS_TASK_STATUS_DETACHED;
104.        initParam.usEstimateRunTime   = 10;
105.        ret = LOS_TaskCreate(&g_PRIOR_HI_taskId, &initParam);
106.        if (ret != LOS_OK) {
107.            LOS_TaskUnlock();
108.
109.            PRINTK("Example__PRIOR_HI_Task create Failed!\r\n");
110.            return;
111.        }
112.        PRINTK("Example_PRIOR_HI_Task create Success!\r\n");
113.
114.
115.        initParam.pfnTaskEntry =
   (TSK_ENTRY_FUNC)Example_PRIOR_LO_Task;
116.        initParam.usTaskPrio = TSK_PRIOR_LO;
117.        initParam.pcName = "PRIOR_LO_Task";
118.        initParam.uwStackSize = OS_TASK_RESOURCE_STATCI_SIZE;
```

```
119.        initParam.uwResved   = LOS_TASK_STATUS_DETACHED;
120.        initParam.usEstimateRunTime   = 10;
121.        ret = LOS_TaskCreate(&g_PRIOR_LO_taskId, &initParam);
122.        if (ret != LOS_OK) {
123.            LOS_TaskUnlock();
124.
125.            PRINTK("Example__PRIOR_LO_Task create Failed!\r\n");
126.            return;
127.        }
128.        PRINTK("Example_PRIOR_LO_Task create Success!\r\n");
129.
130.        initParam.pfnTaskEntry =
    (TSK_ENTRY_FUNC)Example_PRIOR_MID_TaskHi;
131.        initParam.usTaskPrio = TSK_PRIOR_MID;
132.        initParam.pcName = "PRIOR_MID_TaskHi";
133.        initParam.uwStackSize = OS_TASK_RESOURCE_STATCI_SIZE;
134.        initParam.uwResved   = LOS_TASK_STATUS_DETACHED;
135.        initParam.usEstimateRunTime   = 5;
136.        ret = LOS_TaskCreate(&g_PRIOR_MID_taskHiId, &initParam);
137.        if (ret != LOS_OK) {
138.            LOS_TaskUnlock();
139.
140.            PRINTK("Example_PRIOR_MID_TaskHi create Failed!\r\n");
141.            return;
142.        }
143.        PRINTK("Example_PRIOR_MID_TaskHi create Success!\r\n");
144.
145.
146.        initParam.pfnTaskEntry =
    (TSK_ENTRY_FUNC)Example_PRIOR_MID_TaskLo;
147.        initParam.usTaskPrio = TSK_PRIOR_MID;
148.        initParam.pcName = "PRIOR_MID_TaskLo";
149.        initParam.uwStackSize = OS_TASK_RESOURCE_STATCI_SIZE;
150.        initParam.uwResved   = LOS_TASK_STATUS_DETACHED;
151.        initParam.usEstimateRunTime   = 30;
152.        ret = LOS_TaskCreate(&g_PRIOR_MID_taskLoId, &initParam);
153.        if (ret != LOS_OK) {
154.            LOS_TaskUnlock();
155.
156.            PRINTK("Example_PRIOR_MID_TaskLo create Failed!\r\n");
157.            return;
158.        }
159.        PRINTK("Example_PRIOR_MID_TaskLo create Success!\r\n");
160.
```

```
161.        /* 解锁任务调度，此时会发生任务调度，执行就绪队列中最高优先级任
    务 */
162.        LOS_TaskUnlock();
163.
164.        return;
165.    }
```

2. test.c ：

```
1. #include <stdio.h>
2. #include <syscall.h>
3.
4. void newSyscallSample(int num)
5. {
6.     syscall(SYS_new_syscall_sample,num);
7.     return;
8. }
9. int main(void)
10.{
11.  newSyscallSample(0);
12.    return 0;
13.}
```

3. Los_priqueue.c（仅修改部分） ：

```
1. VOID OsPriQueueEnqueue(LOS_DL_LIST *priQueueList, UINT32 *bitMap,
    LOS_DL_LIST *priqueueItem, UINT32 priority)
2. {
3.
4.     LOS_ASSERT(priqueueItem->pstNext == NULL);
5.
6.     if (LOS_ListEmpty(&priQueueList[priority])) {
7.         *bitMap |= PRIQUEUE_PRIOR0_BIT >> priority;
8.         LOS_ListTailInsert(&priQueueList[priority],
    priqueueItem);
9.     }else{
10.        LOS_DL_LIST *currentItem = priQueueList[priority].pstNext;
11.        LOS_DL_LIST *prevItem    = &priQueueList[priority];
12.        while(currentItem != &priQueueList[priority])
13.          {
14.              UINT32 estimateRuntime = *(UINT32*)(priqueueItem+1);
15.              UINT32 curEstimateRuntime =
    *(UINT32*)(currentItem+1);
16.              if(estimateRuntime < curEstimateRuntime)
17.              {
```

```
18.                 priqueueItem->pstNext=currentItem;
19.                 prevItem->pstNext=priqueueItem;
20.                 if(prevItem->pstPrev)
21.                   priqueueItem->pstPrev=prevItem;
22.                 if(currentItem->pstPrev)
23.                   currentItem->pstPrev=priqueueItem;
24.                 return;
25.               }
26.              prevItem = currentItem;
27.              currentItem = currentItem->pstNext;
28.             }
29.              prevItem->pstNext=priqueueItem;
30.              priqueueItem->pstNext = &priQueueList[priority];
31.              priqueueItem->pstPrev=prevItem;
32.       }
33.}
```