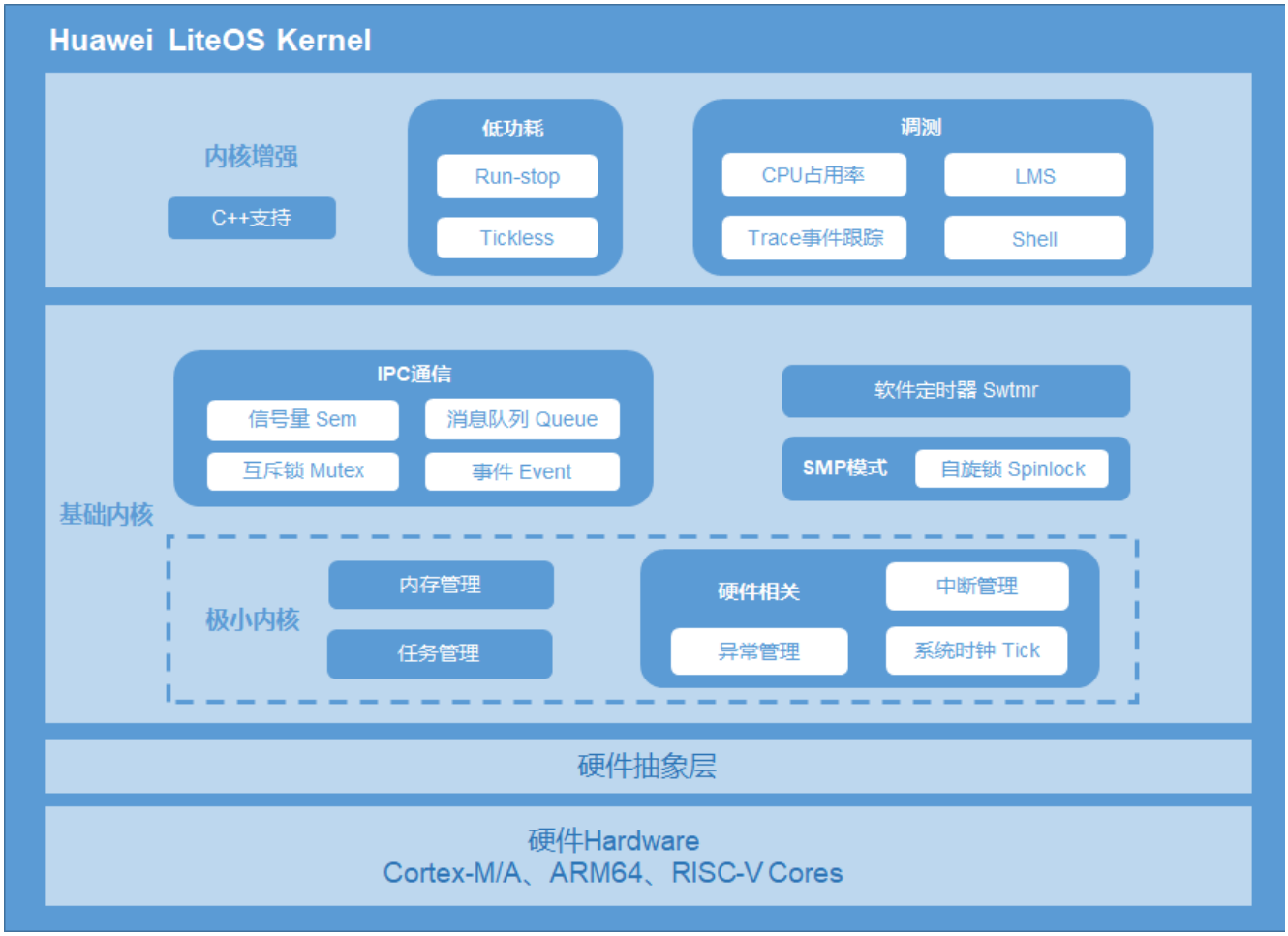
LiteOS 任务调度

任务模块



任务是操作系统一个重要的概念，是竞争系统资源的最小运行单元。任务可以使用或等待 CPU 、使用内存空间等系统资源，并独立于其它任务运行。

任务的**优先级**决定了在发生任务切换时即将要执行的任务，*就绪队列* 中最高优先级的任务将得到执行。

用户创建任务时，系统会初始化**任务栈**，预置上下文。此外，系统还会将“任务入口函数”地址放在相应位置。这样在任务第一次启动进入运行态时，将会执行“任务入口函数”。

任务上下文

任务在运行过程中使用的一些资源，如寄存器等，称为**任务上下文**。当这个任务挂起时，其他任务继续执行，可能会修改寄存器等资源中的值。如果任务切换时没有保存任务上下文，可能会导致任务恢复后出现未知错误。

因此，Huawei LiteOS 在任务切换时会将切出任务的任务上下文信息，保存在自身的**任务栈**中，以便任务恢复后，从栈空间中恢复挂起时的上下文信息，从而继续执行挂起时被打断的代码。

TaskContext上下文结构体

任务上下文（Task Context）指的是任务运行的环境，例如包括程序计数器、堆栈指针、通用寄存器等内容。在多任务调度中，任务上下文切换（Task Context Switching）属于核心内容，是多个任务运行在同一 `CPU` 核上的基础。

任务栈

每个任务都拥有一个独立的栈空间，我们称为任务栈。栈空间里保存的信息包含局部变量、寄存器、函数参数、函数返回地址等。

`LOS_StackInfo` 任务栈结构体定义

```
typedef struct {
    VOID *stackTop;     // 栈顶指针
    UINT32 stackSize;   // 栈大小
    CHAR *stackName;    // 栈名称
} StackInfo;
```

另外定义了一个宏函数 `OS_STACK_MAGIC_CHECK(topstack)` 用于检测栈是否有效，当栈顶等于 `OS_STACK_MAGIC_WORD` 栈是正常的，没有溢出，否则栈顶被改写，发生栈溢出。

```
/* 1:有效正常的栈 0:无效，发生溢出的栈 */
#define OS_STACK_MAGIC_CHECK(topstack) (*(UINTPTR *)(topstack) == OS_STACK_MAGIC_WORD)
```

LOS_StackInfo 任务栈支持的操作

任务栈初始化

栈初始化函数 `VOID OsStackInit()` 使用 2 个参数，一个是栈顶指针 `VOID *stacktop`，一个是初始化的栈的大小。把栈内容初始化为 `OS_STACK_INIT`，把栈顶初始化为 `OS_STACK_MAGIC_WORD`。

```
VOID OsStackInit(VOID *stacktop, UINT32 stacksize)
{
     (VOID)memset_s(stacktop, stacksize, (INT32)OS_STACK_INIT, stacksize);
    *((UINTPTR *)stacktop) = OS_STACK_MAGIC_WORD;
}
```

该函数被创建任务时的 `OsTaskCreateOnly()` 方法调用，完成新创建任务的任务栈初始化。

获取任务栈水线

随着任务栈入栈、出栈，当前栈使用的大小不一定是最大值，`OsStackWaterLineGet()` 可以获取的栈使用的最大值即水线 `WaterLine`。

任务控制块 TCB

每个任务都含有一个任务控制块 (TCB)。TCB 包含了任务上下文栈指针（stack pointer）、任务状态、任务优先级、任务 ID、任务名、任务栈大小等信息。TCB 可以反映出每个任务运行情况。

```
typedef struct {
    VOID            *stackPointer;      /* 任务栈指针 */
```

```
    UINT16          taskStatus;         /* 任务状态 */
    UINT16          priority;           /* 任务优先级 */
    UINT32          taskFlags : 31;     /* 任务扩展标记，支持标记为自删除任务OS_TASK_FLAG_DETACHED、
系统级任务OS_TASK_FLAG_SYSTEM*/
    UINT32          usrStack : 1;       /* 是否使用用户栈 */
    UINT32          stackSize;          /* 任务栈大小 */
    UINTPTR         topOfStack;         /* 栈顶指针 */
    UINT32          taskId;             /* 任务Id */
    TSK_ENTRY_FUNC  taskEntry;          /* 任务入口函数 */
    VOID            *taskSem;           /* 任务持有的信号量 */
#ifdef LOSCFG_LAZY_STACK
    UINT32          stackFrame;         /* 栈帧: 0=Basic, 1=Extended */
#endif
#ifdef LOSCFG_COMPAT_POSIX
    VOID            *threadJoin;        /* pthread 适配 */
    VOID            *threadJoinRetval;  /* pthread 适配 */
#endif
    VOID            *taskMux;           /* 导致任务阻塞的互斥锁 */
#ifdef LOSCFG_OBSOLETE_API
    UINTPTR         args[4];            /* 任务入口函数的参数，兼容遗留API */
#else
    VOID            *args;              /* 任务入口函数的参数 */
#endif
    CHAR            *taskName;          /* 任务名称 */
    LOS_DL_LIST     pendList;           /* 任务就绪队列等链表节点 */
    SortLinkList    sortList;           /* 任务超时排序链表节点 */
#ifdef LOSCFG_BASE_IPC_EVENT
    EVENT_CB_S      event;
    UINT32          eventMask;          /* 事件掩码 */
    UINT32          eventMode;          /* 事件模式 */
#endif
    VOID            *msg;               /* 分给给队列的内存*/
    UINT32          priBitMap;          /* BitMap for recording the change of task priority,
                                           the priority can not be greater than 31 */
    UINT32          signal;             /* 任务信号 */
#ifdef LOSCFG_BASE_CORE_TIMESLICE
    UINT16          timeSlice;          /* 剩余的时间片 */
#endif
#ifdef LOSCFG_KERNEL_SMP
    UINT16          currCpu;            /* 当前运行的CPU核编号 */
    UINT16          lastCpu;            /* 上次运行的CPU核编号 */
    UINT32          timerCpu;           /* 任务阻塞或请求的CPU核编号 */
    UINT16          cpuAffiMask;        /* CPU亲和性掩码，最多支持16个核 */
#ifdef LOSCFG_KERNEL_SMP_TASK_SYNC
    UINT32          syncSignal;         /* 同步信号处理标记 */
#endif
#ifdef LOSCFG_KERNEL_SMP_LOCKDEP
    LockDep         lockDep;
#endif
#endif
#ifdef LOSCFG_DEBUG_SCHED_STATISTICS
    SchedStat       schedStat;          /* 调度统计*/
#endif
#ifdef LOSCFG_KERNEL_PERF
    UINTPTR         pc;
    UINTPTR         fp;
```

```
#endif
} LosTaskCB;
```

任务模块初始化

*任务模块初始化*函数 `OsTaskInit()` 会计算需要申请的内存大小 `size`，为任务控制块 `TCB` 数组 `g_taskCBArray` 申请内存。初始化双向链表 `g_losFreeTask` 用作空闲的任务链表、`g_taskRecycleList` 可以回收的任务链表。然后循环初始化每一个任务，任务状态未使用 `OS_TASK_STATUS_UNUSED`，初始化任务 `Id`，并把任务挂在空闲任务链表上，然后初始化优先级队列与排序列表。

系统启动阶段还会创建 `idle` 空闲任务，空闲任务的入口执行函数为 `OsIdleTask()`，它调用 `LOS_TaskResRecycle()` 回收任务栈资源。

任务管理支持的操作

- **Huawei LiteOS** 任务管理模块提供
  - 任务创建
  - 任务删除
  - 任务延时
  - 任务挂起
  - 任务恢复
  - 更改任务优先级
  - 锁定任务调度
  - 解锁任务调度
  - 根据任务控制块查询任务ID
  - 根据ID查询任务查询任务控制块信息功能。
- 任务创建时，如果 OS 的系统可用空间少于任务，则创建失败，反之亦然。
- 用户创建任务时，系统会将任务栈进行初始化，预置上下文。
  - **任务入口函数** 也放到了相应的位置，在任务第一次执行时便可执行 **任务入口函数**。

调度模块 (Schedule)

调度，`Schedule`。负责选择系统要处理的下一个任务。调度模块需要协调处于就绪状态的任务对资源的竞争，按优先级策略从就绪队列中获取高优先级的任务，给予资源使用权。

- `LiteOS` 内核调度源代码 包括调度模块的私有头文件kernel\base\include\los_sched_pri.h、`C` 源代码文件kernel\base\sched\sched_sq\los_sched.c，这个对应单链表就绪队列。还有个`调度源代码文件kernel\base\sched\sched_mq\los_sched.c，对应多链表就绪队列。本文主要剖析对应单链表就绪队列的调度文件代码，使用多链表就绪队列的调度代码类似。
- 调度模块汇编实现代码 调度模块的汇编函数有 `OsStartToRun`、`OsTaskSchedule` 等，根据不同的 `CPU` 架构，分布在下述文件里：`arch\arm\cortex_m\src\dispatch.S`、`arch\arm\cortex_a_r\src\dispatch.S`、`arch\arm64\src\dispatch.S`。

调度模块常用接口

`los_sched.c` 定义的调度接口，包含 `VOID OsSchedPreempt(VOID)`、`VOID OsSchedResched(VOID)` 两个主要的调度接口。两者的区别是，前者需要把当前任务放入就绪队列内，再调用后者触发调用。后者直接从就绪队列里获取下一个任务，然后触发调度去运行下一个任务。这2个接口都是内部接口，对外提供的调度接口是宏函数 `STATIC INLINE VOID LOS_Schedule(VOID)`，三者有调用关系 `STATIC INLINE VOID LOS_Schedule(VOID)--->VOID OsSchedPreempt(VOID)--->VOID OsSchedResched(VOID)`。

抢占调度函数 VOID OsSchedResched(VOID)

1. 验证需要持有任务模块的自旋锁。
2. 判断是否支持调度
3. 获取当前运行任务，从就绪队列中获取下一个高优先级的任务。验证下一个任务 newTask 不能为空，并更改其状态为非就绪状态。
4. 判断当前任务和下一个任务不能为同一个，否则返回。
5. 更改2个任务的运行状态，当前任务设置为非运行状态，下一个任务设置为运行状态。
6. 如果支持多核，则更改任务的运行在哪个核
7. 如果支持时间片调度，并且下一个新任务的时间片为0，设置为时间片超时时间的最大值 LOSCFG_BASE_CORE_TIMESLICE_TIMEOUT 。
8. 设置下一个任务 newTask 为当前运行任务，会更新全局变量 g_runTask 。然后调用汇编函数 OsTaskSchedule(newTask, runTask) 执行调度

抢占调度函数 VOID OsSchedPreempt (VOID)

把当前任务放入就绪队列，从队列中获取高优先级任务，然后尝试调度。当锁调度，或者没有更高优先级任务时，调度不会发生。

如果开启时间片调度并且当前任务时间片为0，则执行把当前任务放入就绪队列的尾部，否则执行把当前任务放入就绪队列的头部，同等优先级下可以更早的运行。调用函数 OsSchedResched() 去调度。