ToRTOS RTOS 数据结构文档

本次补充：

- IPC/线程/消息队列结构字段完整描述
- 栈帧 / 上下文切换顺序示意
- 优先级位图与调度路径 ASCII 图
- 消息队列内存池布局与节点结构
- 互斥量优先级继承字段说明
- 线程状态转换更清晰的文字图
- 未来结构扩展占位

---

1. 基础类型与状态码

`tdef.h` 定义的整型别名保证与编译器一致。
`t_status` 枚举：详见ToRTOS_API.md。
宏常量统一前缀：`TO_`。

---

2. 双向循环链表：t_list

```
typedef struct list {
    struct list *next;
    struct list *prev;
} t_list_t;
```

性质：

- 所有内核队列（就绪队列、定时器、IPC 等待列表）使用该结构。
- 头节点也是一个 `t_list_t`，空时 next== prev == head。

相关操作：

- `t_list_init`
- `t_list_insert_after`
- `t_list_insert_before`
- `t_list_delete`
- `t_list_isempty`

使用技巧：通过 `T_LIST_ENTRY(node, type, member)` 获取父结构。

---

3. 线程控制块：t_thread

```
typedef struct thread {
    void        *psp;               /**< Saved process stack pointer
                                     (hardware context next restore point) */
    t_thread_entry_t    entry;      /**< Entry function */
    void        *arg;
    void        *stackaddr;         /**< Stack base (low address) */
    t_uint32_t  stacksize;          /**< Stack size in bytes */
```

```
    t_list_t    tlist;              /**< Run / wait queue list node */
    t_uint8_t   current_priority;   /**< Current (possibly boosted) priority */
    t_uint8_t   init_priority;      /**< Original priority at creation */
    t_uint32_t  number_mask;        /**< Bit mask for ready group */

    t_uint32_t  init_tick;          /**< Time slice length (ticks) */
    t_uint32_t  remaining_tick;     /**< Remaining time slice */
    t_int32_t   status;             /**< Thread lifecycle status flags */
    t_timer_t   timer;              /**< Per-thread sleep/timeout timer */

#if (TO_USING_STATIC_ALLOCATION && TO_USING_DYNAMIC_ALLOCATION)
    t_uint8_t   is_static_allocated;
#endif
} t_thread_t;
```

状态字段

使用宏：

```
TO_THREAD_INIT
TO_THREAD_READY
TO_THREAD_RUNNING
TO_THREAD_SUSPEND
TO_THREAD_TERMINATED
TO_THREAD_DELETED
```

状态转换详见 API 文档表格。

位图调度结构

- `number_mask = 1U << current_priority`
- 全局 `t_thread_ready_priority_group` 组合所有 READY 线程优先级。

栈初始化

- `t_stack_init` 写入初始 PC=entry，LR 指向 `t_thread_exit`（避免直接 return 崩溃）。

---

4. 定时器：t_timer

```
typedef struct timer {
    t_list_t    row[TO_TIMER_SKIP_LIST_LEVEL];  /**< Skip-list / level nodes */
    void        (*timeout_func)(void *p);       /**< Timeout callback */
    void        *p;                             /**< User parameter */
    t_uint32_t  init_tick;                      /**< Initial duration (ticks) */
    t_uint32_t  timeout_tick;                   /**< Absolute expiration tick */
} t_timer_t;
```

特点：

- 当前实现为单层按到期时间排序链表。
- `timeout_tick` = 安排时刻 + init_tick。
- 回调执行在 `t_tick_increase` -> `t_timer_check` 调用路径（中断上下文）。

---

5. IPC 父类：t_ipc_t

```
typedef struct {
    t_ipc_type_t   type;          /* IPC type */
    union
    {
        t_queue_pointers_t queue;     /* Used for queue */
        t_sema_data_t   sema;      /* Used for semaphore/mutex */
    } u;

    t_list_t     wait_list;      /* Thread wait list */

    t_uint16_t   msg_waiting;    /* Current item count or resource count */
    t_uint16_t   length;         /* Max number of items or max count */
    t_uint16_t   item_size;      /* Size of each item */
    t_uint8_t    status;         /* 1=valid, 0=deleted */
    t_uint8_t    mode;           /* FIFO / PRIO */
#if (TO_USING_STATIC_ALLOCATION && TO_USING_DYNAMIC_ALLOCATION)
    t_uint8_t   is_static_allocated;
#endif
} t_ipc_t;
```

为信号量/互斥量/消息队列等共享：

- `mode` == TO_IPC_FLAG_FIFO / TO_IPC_FLAG_PRIO
- `wait_list` 链表元素是 `thread.tlist`

---

6. 信号量：t_ipc_t (IPC_SEMA)

```
typedef struct {
    t_thread_t  *holder;        /* Current mutex owner */
    t_uint16_t  recursive;      /* Recursive count for mutex */
    t_uint8_t   original_prio;  /* Owner's original priority */
} t_sema_data_t;
```

行为：

- `msg_waiting>0` 直接获取
- `msg_waiting==0` 按等待策略挂入 `wait_list`
- 释放：有挂起线程→先自增 msg_waiting 再唤醒一个；否则直接自增

限制：`msg_waiting <= 0xFFFF`。

---

7. 互斥量 (t_ipc_t IPC_MUTEX / IPC_RECURSIVE_MUTEX)

```
typedef struct {
    t_thread_t  *holder;        /* Current mutex owner */
    t_uint16_t  recursive;      /* Recursive count for mutex */
    t_uint8_t   original_prio;  /* Owner's original priority */
} t_sema_data_t;
```

补充：

- 递归上限 `MUTEX_RECURSIVE_COUNT_MAX`
- 简单优先级继承：高优先级等待时提升 holder->current_priority
- 释放最后一层时恢复 `original_prio`

---

8. 消息队列 (t_ipc_t IPC_QUEUE)

```
typedef struct {
    t_uint8_t *head;       /* Start of buffer */
    t_uint8_t *tail;       /* End marker */
    t_uint8_t *read_from;  /* Last read position */
    t_uint8_t *write_to;   /* Next write position */
} t_queue_pointers_t;
```

内存布局（池）：

```
+-----------+---------+   (node0)
| next ptr  | payload |
+-----------+---------+   (node1)
| next ptr  | payload |
...
```

初始化构建自由链表：FIFO 形式 → 分配 O(1)。

---

9. 全局调度相关

```
t_thread_t *t_current_thread;
t_uint8_t t_cur_num_of_ready_tasks;
t_list_t t_thread_priority_table[TO_THREAD_PRIORITY_MAX];
t_uint32_t t_thread_ready_priority_group;
t_list_t t_thread_defunct_list;
volatile t_uint32_t t_tick;
```

位图：

```
t_thread_ready_priority_group
bit i = 1 ↔ priority i 有至少一个 READY 线程
```

查找：

```
highest = __t_ffs/__t_fls(bitmap) - 1;
```

---

10. 调度结构与位图优先级查找

READY 队列（示例 0..3）：

```
prio0: [HEAD] <-> T0a <-> T0b <-> [HEAD]
prio1: [HEAD] <-> T1a <-> [HEAD]
prio2: [HEAD] <-> (empty)
prio3: [HEAD] <-> T3a <-> T3b <-> T3c <-> [HEAD]
```

此时位图：

```
bitmap: b00011011 (LSB=prio0)
```

上下文切换触发来源：

- 新线程变 READY 且优先级高于当前
- 当前线程时间片耗尽（轮转）
- 阻塞 / 删除 / 退出当前线程
- 显式 yield

---

11. 线程状态转换文字图

```
INIT --startup--> READY --dispatch--> RUNNING
 ^                   ^  ^               |
 |                   |  |               |
 |               (timeout/IPC release)  |
 |                   |  |               v
 |<--restart-- DELETED  |         (sleep / IPC wait)
 |                      |               |
 |              READY <--- SUSPEND <----+
 |
 +-- idle cleanup -- TERMINATED <- (delete / exit from RUNNING)
          |
          +--> DELETED --(restart)--> INIT
```

---

12. 定时器链表示例

```
(now) tick=100
[HEAD] -> (timeout=120) -> (130) -> (210) -> ...
插入 timeout=125 node：
遍历到 130 停止，插入其前
```

比较采用：

```
(s_int32_t)(current_tick - node->timeout_tick) >= 0
```

或双链表机制

以处理 tick 回绕。

---

13. 上下文切换关键变量

变量	说明
t_prev_thread_sp_p	上一线程 PSP 存放位置地址
t_next_thread_sp_p	下一线程 PSP 存放位置地址
t_interrupt_flag	触发 PendSV 标志（防止重复配置）

14. 一致性策略

场景	顺序
阻塞	关中断 → 从 READY 移除 → 状态=SUSPEND → 加入等待队列 → 开中断 → 调度
唤醒	关中断 → 从等待队列移除 → 状态=READY → 插入 READY → 开中断
删除线程	从 READY 移除 → 停止私有定时器 → 状态=TERMINATED → 入 defunct
Idle 清理	遍历 defunct → 状态=DELETED → 摘链

15. 关键宏与可配置项（摘要）

宏	作用
TO_THREAD_PRIORITY_MAX	优先级数量
TO_TICK	Tick 频率 Hz
TO_TIMER_SKIP_LIST_LEVEL	定时器层级（当前=1）
TO_IDLE_STACK_SIZE	Idle 栈大小
TO_USING_SEMAPHORE / MUTEX / RECURSIVE_MUTEX / QUEUE / IPC	子系统开关
TO_DEBUG	启用调试输出
TO_PRINTF_BUF_SIZE	printf 临时缓冲

16. 变更记录（文档）

- 2026-01-21 更新：根据最新 tdef.h 和 ToRTOS.h 更新结构体和宏名称。
- 2025-08-27 增补：互斥量/消息队列等api简单介绍。
- 2025-08-26 增补：互斥量/消息队列字段说明、上下文切换流程、位图调度示意、未来扩展占位。
- 2025-08-24 初版结构描述。

如发现与源码不符，请以源码为准并提交 Issue。

Happy hacking with ToRTOS!