FreeRTOS

Liam

2023 年 9 月 15 日

摘要

Something to add...

1 裸机系统与多任务系统

1.1 裸机系统

1.1.1 轮询系统

定义：

轮询系统即是在裸机编程的时候，初始化好相关的硬件，然后将主程序在一个**死循环**里面不断地循环，顺序处理各种事件。（当有要检测外部信号的时候实时性较差）

1.1.2 前后台系统

定义：

前后台系统即是在轮询系统上加上了**中断**，外部事件的响应在终端里面完成，事件处理还是在轮询系统中完成。中断被称为**前台**，`main()`函数被称为**后台**。

相对于轮询系统，前后台系统确保事不会丢失，加上中断可嵌套，大大提高程序的实时响应功能。

1.2 多任务系统

相对于前后台系统，多任务系统的事件响应都是在中断中完成，但是事件的处理实在任务中完成的。

多任务系统中，**任务**与**中断**一样，也具有优先级，优先级高的任务会被优先执行。当一个紧急事件在中断中被标记之后，如果事件对应的任务的优先级足够高，就会立即得到相应。

在多任务系统中，根据程序的功能，我们把程序中体分割成**分割成一个个独立的，无限循环不能返回的小程序**—-任务。各个任务独立互不干扰，各自具备自身的优先级，它由操作系统调度管理。

	模型	事件响应	事件处理	特点
	轮询系统	主程序	主程序	轮询响应并处理事件
	前后台系统	中断	主程序	实时响应事件，轮询处理
	多任务系统	中断	任务	实时响应并处理事件

1.3 数据结构--列表与列表项

列表和列表项对应 C 语言中的链表和节点。

链表复习

1.3.1 单向列表

节点本身包含一个指针，用于只想后一个节点，并且可以携带一些私有信息。

列表一般包含一个**节点计数器，节点插入和删除操作器**

1.3.2 双向列表

节点中将会有两个节点指针，一个指向头一个指向尾部。

1.4 FreeRTOS 链表实现

FreeRTOS 中链表的实现，均在`list.h`和`list.c`中。示意图：

xLISt_tIEM
pxNext
pxPrevious
pvOwner
pvContainer

```
struct xLIST_ITEM
{
        TickType_t xItemValue;            //辅助值，帮助节点进行顺序排列
        struct xLIST_ITEM * pxNext;       //指向下一个节点
        struct xLIST_ITEM * pxPrevious;   //指向上一个节点
        void * pvOwner;                   //指向拥有该节点的内核对象
        void * pvContainer;               //指向该节点所在链表
};
typedef struct xLIST_ITEM ListItem_t;
```

在 FreeRTOS 中，凡是涉及数据类型的地方，标准的 C 数据类型用typedef重新设置在portmacro.h

1.4.1 实现根节点

```
        typedef struct xLIST
        {
                UBaseType_t uxNumberOfItems;        //链表节点计数器
                ListItem_t * pxIndex;               //链表节点索引指针
                MiniListItem_t xListEnd;            //链表最后一个节点
        } List_t;
```

链表根节点初始化

```
void vListInitialise(List_t * const pxList)
{
        pxList -> pxIndex = ( ListItem_t *) &(pxList->xListEnd);        //将链表索引指针指向最后
```

```
            一个节点
4           pxList->xListEnd.xItemValue = portMAX_DELAY;
                                    //辅助排序设置为最大
5           pxList->xListEnd.pxNext = (ListItem * ) &(
               pxList ->xListEnd);    //最后一个节点的
               pxNext和previous指向自己
6           pxList->xListEnd.pxPrevious = (ListItem_t * )
               &(pxList->xListEnd);
7           pxList->uxNumberOfItems = (UBaseType_t) 0U;
               //节点计数值为0，表示列表为空。
8   }
```

2 任务的定义与任务切换

2.1 任务

在多任务系统中，我们根据功能不同，将整个系统分割为一个个独立且无法返回的函数。

2.1.1 定义任务栈

在多任务系统中，要为每一个任务独立分配**栈空间**

```
1           #define TASK1_STACK_SIZE
2           StackType_t Task1Stack[TASK1_STACK_SIZE];
```

2.1.2 定义任务控制块

在裸机系统中，任务是 CPU 按照顺序执行的；在多任务系统中是由系统调度的。系统为顺利调取任务，为每个任务都定义了一个**任务控制块**，

```
1   typedef struct tskTaskControlBlock
2   {
3           volatile StackType_t *pxTopOfStack;
4           ListItem_t xStateListItem;  //这是一个内置在
               TCB控制块中的链表节点。
5           StackType_t *pxStack;
```

```
        char pcTaskName[configMAX_TASK_NAME_LEN];
}tskTCB;
typedef tskTCK TCB_t;
```

2.1.3 实现任务创建函数

任务栈、函数实体和任务控制块都需要最终联系起来才能由系统统一调度。