WM_W800_SDK_OS 移植指导

V1.0

北京联盛德微电子有限责任公司（winner micro)

地址：北京市海淀区阜成路 67 号银都大厦 18 层

电话：+86-10-62161900

公司网址：www.winnermicro.com

文档修改记录

版本	修订时间	修订记录	作者	审核
V0.1	2019/9/25	[C]创建文档	Zhangwl	
V0.2	2020/7/8	统一字体	Cuiyc	
V1.0	2020/8/10	升级版本号	Cuiyc	

目录

1 引言

1.1 编写目的

本文档描述指导 Winnermicro W800 芯片的 SDK 开发人员如何移植或切换到新的 OS。

1.2 预期读者

W800 SDK 的开发人员及工程实现人员

1.3 术语定义

术语	说明
API	Application Program Interface
OS	Operating System
SDK	Software Development Kit

1.4 参考资料

无

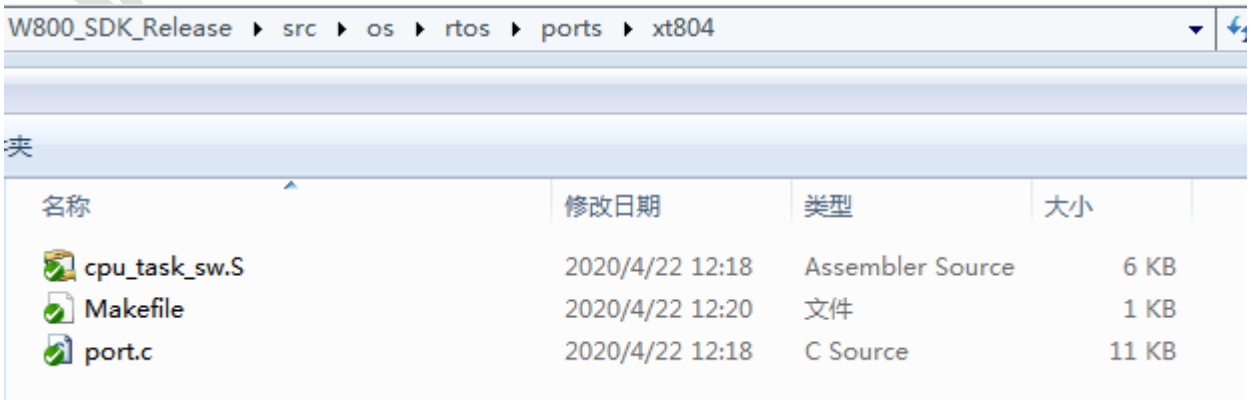
2 SDK Kernel 移植

如图 1 所示，SDK 的操作系统相关代码均放置在./Src/OS 目录下，目前已经移植了 FreeRTOS 实时操作系统。如果用户有需要使用其他实时操作系统，可以在此目录下建立新的文件夹，添加自己需要的操作系统

图 1 放置目录

3 系统调度相关移植

相关函数参见 sdk_root_dir/src/os/rtos/ports/ck804/下的 cpu_task_sw.S 文件及 port.c 文件。此处以 FreeRTOS 为例，用户使用此操作作系统时，需要实现这两个文件中用所到的相应接口。如果需要要使用其它操作系统，建议下载 xt804 的 sdk，找到其示例代码中已经移植好的操作系统直接使用；如果没有现有可用的，则需要根据所用操作系统的要求实现相应接口，这里可能会需要内核(XT804)原厂(平头哥)的技术支持。



4 内存相关移植

在 wm_mem.c 文件可以看到 SDK 中为 FreeRTOS 提供的 malloc 内存申请方式。因为 FreeRTOS 系统有特殊的要求，中断函数和普通函数的内存申请方式不一样。

如果用户的操作系统有相关要求，则也需要做相关处理；如果没有相关要求，则关闭 wm_os_config.h 中的 TLS_OS_FREERTOS 宏定义就是 C 库的 malloc 函数申请内存了。

5 printf 函数

在 retarget.c 文件里可以看到两个函数：

1）int wm_vprintf(const char *fmt, va_list arg_ptr);

2）int wm_printf(const char *fmt,...);

这是两个 Wi-Fi 里会使用的打印函数，用户可以依据编译环境自行实现此函数，也可以使用已实现的函数，以与 printf 函数区分。

6 OSAL 层移植

为了方便操作系统的移植，操作系统和 W80X SDK 应用程序之间封装了一层抽象层，即对程序中需要用到的操作系统的接口函数进行了重新封装，程序采用统一的接口调用操作系统相关函数。

需要实现的接口函数在./include/OS 目录下的 wm_osal.h 中定义

6.1 系统时钟

声明 HZ 为常量：

```
const unsigned int HZ = 500;
```

在 wm_osal.h 文件里引用

```
extern const unsigned int HZ;
```

操作系统时钟定义，此处表明 500 个系统时钟为 1HZ 的频率，即 2ms 产生一个 sysTick 中断，用户可以根据需要自己需要修改，对于部分操作系统需要将操作系统中对应的时间配置设置与我们定义的一致，例如 FreeRTOS 中需要修改 freeRTOSConfig.h 中的设置

```
#define configTICK_RATE_HZ          ( ( portTickType ) 500u )    //时 间 片 中 断 的 频 率
```

6.2 操作系统类型

```
/** ENUMERATION of OS */
enum TLS_OS_TYPE{
    OS_UCOSII = 0,
    OS_FREERTOS = 1,
    OS_MAX_NUM
};
```

操作系统类型定义，用于指示当前操作系统的类型，用户可以添加自己的操作系统类型。主要是用于特殊场合的兼容性设计（暂时没有使用）。

6.3 OSAL 层接口函数

OSAL 层接口函数，用户可以参考现有 SDK 中支持的 FreeRTOS 里面的实现方法将自己的操作系统相关函数封装在这些接口里面。

FreeRTOS 的函数封装在 wm_osal_rtos.c 文件中。

6.3.1 时钟相关函数

函数原型	void tls_os_timer_init(void); void tls_os_time_tick(void *p);
函数介绍	时钟相关的两个函数，这两个函数已经在我们的工程中实现。

函数原型	u32 tls_os_get_time(void);
函数介绍	获取当前操作系统已经运行的 tick 值。

6.3.2 系统初始化函数

函数原型	void tls_os_init(void *arg);
函数介绍	操作系统初始化函数，程序初始化的时候调用，用户根据自己所用的操作系统需要实现。

函数原型	int tls_os_get_type(void);
函数介绍	获取当前操作系统类型，不同的操作系统可能在某些程序应用中的处理方式可能不同，程序中会判断此函数的返回值来决定程序处理方式。

6.3.3 任务相关函数

函数原型	void tls_os_start_scheduler(void);
函数介绍	启动任务调度，创建完成任务后调用启动系统任务的执行。

函数原型	tls_os_status_t tls_os_task_create(tls_os_task_t *task,const char* name, void (*entry)(void* param),void* param,u8 *stk_start,u32 stk_size,u32 prio,u32 flag);
函数介绍	任务创建函数，需要注意两点： 1. stk_start 任务栈不能使用到其它外设使用的内存外设使用内存在 wm_ram_config.h 中定义 2. prio 任务优先级数值越大，优先级越低，0 是最高优先级，部分操作系统不是按照这顺序需要调整，可以参考 FreeRTOS 的封装方法（configMAX_PRIORITIES - prio）。

函数原型	tls_os_status_t tls_os_task_del(u8 prio, void (*freefun)(void));
函数介绍	删除任务，目前程序中未使用删除任务，可以不实现。

6.3.4 信号量相关函数

函数原型	tls_os_mutex_create tls_os_mutex_delete tls_os_mutex_acquire tls_os_mutex_release
函数介绍	互斥信号量的创建、删除、获取、释放。 注意：对于信号量的释放，部分操作系统需要区分是否在中断中操作，如果是在中断中操作的话需要调用中断处理专门的接口。而我们的应用程序中的调用都是统一接口，因此用户需要在封装函数的内部来判断当前函数是否处于中断之中。参考 FreeRTOS 中的实现方式，tls_get_isr_count 接口来判断当前是否在中断函数中调用。

函数原型	tls_os_sem_create tls_os_sem_delete tls_os_sem_acquire tls_os_sem_release tls_os_sem_set
函数介绍	计数信号量的创建、删除、获取、释放、设置计数值。 注意：对于信号量的释放，部分操作系统需要区分是否在中断中操作，如果是在中断中操作的话需要调用中断处理专门的接口。而我们的应用程序中的调用都是统一接口，因此用户需要在封装函数的内部来判断当前函数是否处于中断之中。参考

	FreeRTOS 中的实现方式，tls_get_isr_count 接口来判断当前是否在中断函数中调用。

6.3.5 队列相关函数

函数原型	tls_os_queue_create tls_os_queue_delete tls_os_queue_send tls_os_queue_flush tls_os_queue_receive
函数介绍	消息队列的创建、删除、发送、刷新、接收。 注意：对于消息队列的发送，部分操作系统需要区分是否在中断中操作，如果是在中断中操作的话需要调用中断处理专门的接口。而我们的应用程序中的调用都是统一接口，因此用户需要在封装函数的内部来判断当前函数是否处于中断之中。参考 FreeRTOS 中的实现方式，tls_get_isr_count 接口来判断当前是否在中断函数中调用。

6.3.6 邮箱消息相关函数

函数原型	tls_os_mailbox_create tls_os_mailbox_delete

	tls_os_mailbox_send tls_os_mailbox_receive
函数介绍	邮箱消息的创建、删除、发送、接收。 注意：对于邮箱消息的发送，部分操作系统需要区分是否在中断中操作，如果是在中断中操作的话需要调用中断处理专门的接口。而我们的应用程序中的调用都是统一接口，因此用户需要在封装函数的内部来判断当前函数是否处于中断之中。参考 FreeRTOS 中的实现方式，tls_get_isr_count 接口来判断当前是否在中断函数中调用。

6.3.7 临界区相关函数

函数原型	tls_os_set_critical(void)
函数介绍	关闭当前系统所有中断。

函数原型	void tls_os_release_critical(u32 cpu_sr)
函数介绍	重新使能当前系统所有中断。

6.3.8 系统定时器相关函数

函数原型	tls_os_timer_create
函数介绍	定时器回调任务创建。

	注意：定时器任务创建完成后为调用 tls_os_timer_start 前，定时器是不启动的。对于需要封装的操作系统也必须要满足这一点，如果操作系统可选是否 auto_run，则选择不启动。

函数原型	tls_os_timer_start tls_os_timer_stop tls_os_timer_delete
函数介绍	定时器任务的启动、停止、删除。

函数原型	tls_os_timer_change
函数介绍	修改定时器任务的定时值。 注意：调用此函数后，**定时器会重新启动**，封装中需要注意这一点，可参考 FreeRTOS 中的实现。

函数原型	void tls_os_time_delay(u32 ticks)
函数介绍	任务延时，单位 tick。

6.3.9 其他

函数原型	void tls_os_disp_task_stat_info(void)
函数介绍	打印当前操作系统的运行情况，调试时可以使用，用户根据需要实现。

7 在工程中切换 Kernel

Keil 工程更换 Kernel，需要找到 wm_os_config.h 文件，选择当前要使用的操作系统。修改需要选用的 Kernel 后，在 keil 工程中将 OS 组下的 OS 操作系统相关.c 文件替换成用户移植的 kernel 文件。

GCC 更换操作系统，直接修改 wm_config_gcc.inc 文件即可，如果用户使用自己的操作系统还需要修改对应的 Makefile 文件，具体见《WM_W800_SDK 脚本编译指南》。