用uXos开发项目

charlie weng

2010-03-27

用uXos开发项目

uXos 协作式操作系统

uXOS的设计目标是资源相当紧缺的MCU上运行，目标是代替前后台程序，易维护的特点。uXOS的底层是与Protothreads的设计实现一致。

主要特点:

1. 与硬件平台无关，方便移植到任何的MCU中，包括32位的MCU
2. 统一堆栈，适合RAM苛刻的MCU系统中应用
3. 系统开销小，最新的uXos 与前后台代码框架相当
4. 易于维护

用uXos开发项目

什么是 Protothreads

Protothreads 是由瑞典计算机科学院的科学家Adam Dunkels 所创的一种新的线程编程方法。按 Adam Dunkels 所说,Protothreads 是专为资源紧张的系统设计的一种耗费资源少,且不使用堆栈的线程模型,它可以不使用复杂的状态机机制来实现顺序流的控制。

Protothreads 也可以用于操作系统当中。简单地说, Protothreads 借鉴了用 C 语言实现协同(co-routine)的原理,它应用 switch-case 或goto(GCC扩展)语句的直接跳转功能,实现了有条件阻塞(conditional block) ,最终实现了虚拟的并行处理功能(concurrent) 。

实际上,Protothreads并不是真正的线程,在多任务的切换中并不会真正涉及上下文的切换,其线程的调度也仅仅是依靠隐式的 return ,进而退出函数体来完成的。但是 Protothreads 的优点却是实实在在的。首先它不需要堆栈空间,而正如笔者用宏实现的那样,Protothreads 也实现了很多只有线程编程方法才能实现的机制,比如阻塞。而用宏进行了封装之后,使用者完全可以像使用线程一样使用它们,而且其逻辑更加简化,这大大增加了程序的清晰度,并降低了开发维护的难度。

用uXos开发项目

uXos 是构建在Protothreads的一种协作式操作系统，虽然短小精悍，但功能强大，其工程目录包含如下文件

用uXos开发项目

Protothreads API

函数	注释
PT_INIT(pt)	初始化任务变量，只在初始化函数中执行一次就 行
PT_BEGIN(pt)	启动任务处理，放在函数开始处
PT_END(pt)	结束任务，放在函数的最后
PT_WAIT_UNTIL(pt, condition)	等待某个条件（条件可以为时钟或其它变量，IO等）成立，否则直接退出本函数，下一次进入本函数就直接跳到这个地方判断
PT_WAIT_WHILE(pt, cond)	和上面一个一样，只是条件取反了
PT_WAIT_THREAD(pt, thread)	等待一个子任务执行完成
PT_SPAWN(pt, child, thread)	新建一个子任务，并等待其执行完退出
PT_RESTART(pt)	重新启动某个任务执行
PT_EXIT(pt)	任务后面的部分不执行，直接退出重新执行
PT_YIELD(pt)	挂起任务
PT_YIELD_UNTIL(pt, cond)	锁死任务并在等待条件成立，恢复执行

用uXos开发项目

uXos 协作式操作系统API

函数	注释
uXOS_BEGIN( )	启动任务处理，放在函数开始处
uXOS_END( )	结束任务，放在函数的最后
uXOS_WAIT_UNTIL(condition)	等待某个条件（条件可以为时钟或其它变量，IO等）成立，否则直接退出本函数，下一次进入本函数就直接跳到这个地方判断
uXOS_WAIT_WHILE(cond)	和上面一个一样，只是条件取反了
uXOS_WAIT_THREAD(child, thread)	等待一个子任务执行完成
uXOS_SPAWN(child, thread)	新建一个子任务，并等待其执行完退出
uXOS_RESTART( )	重新启动任务执行
uXOS_EXIT( )	任务后面的部分不执行，直接退出重新执行
uXOS_YIELD( )	挂起任务
uXOS_START(tcb)	重新启动某个任务执行
uXOS_YIELD_UNTIL(cond)	锁死任务并在等待条件成立，恢复执行
uXOS_SEM_INIT(s, c)	信号量初始化
uXOS_SEM_WAIT(s)	等待一个信号量
uXOS_SEM_SIGNAL(s)	发送一个信号量
uXOS_DELAY(param)	等待指定为param 倍TICK的时间

用uXos开发项目

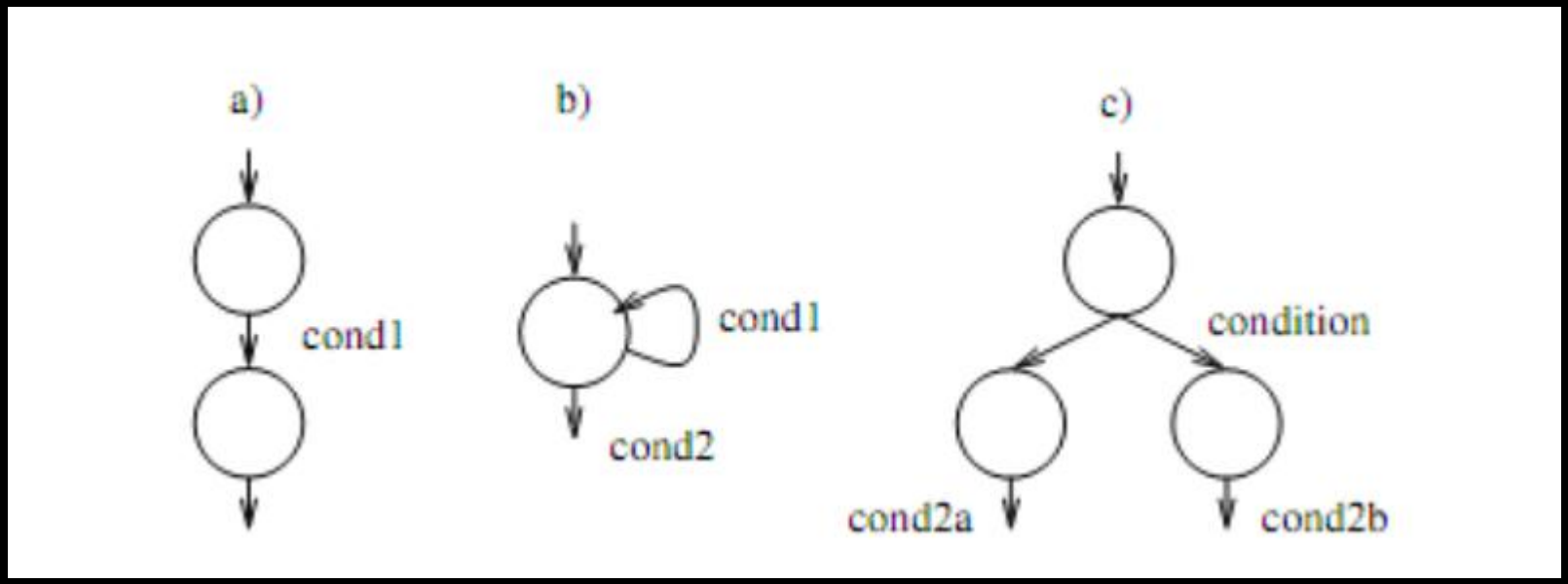
函数	注释
uXOS_MSGQ_INIT(struct MSG_Q *m, BYTE* msgQ, BYTE msg_size, BYTE q_size)	初始化消息队列
uXOS_MSGQ_WAIT(struct MSG_Q *m, BYTE* msg)	等待一个消息
uXOS_MSGQ_POST(struct MSG_Q *m, BYTE* msg)	发送一个消息
void os_init(void (*cb)(void));	系统回调函数初始化
HANDLE create_task(uXOS_THREAD(Task),BYTE pri);	创建一个任务
void os_start()	uXOS系统启动调度执行
void ettimer_set(TM *tm, TICK timeout);	事件定时器设定
BYTE etimer_expired(TM *tm);	等待事件定时器超时（1为超时，0为没有超时）
void utimer_init(void);	调用定时器初始化
void utimer_stop(struct uTimer *timer);	停止定时器
void utimer_start(struct uTimer *timer);	启动定时器
void utimer_delete(struct uTimer *timer);	删除定时器
struct uTimer* utimer_create(void (*cb)(void),TICK timeout, BYTE period);	创建一个定时器
void utimer_ctrl(struct uTimer *timer,BYTE cmd, void* arg);	设置定时器参数
struct uTimer* utimer_callback(void (*cb)(void), TICK timeout, BYTE option);	调用定时器回调函数设定
uXOS_EXEC(child, thread);	调用一个线程，非阻塞式

用uXos开发项目

程序控制状态机之uXos描述

所有的程序控制流状态机可以分解为三个基本模式 ： 顺序，迭代和选择。

如下图所示:



a)顺序　　b)迭代　　c)选择

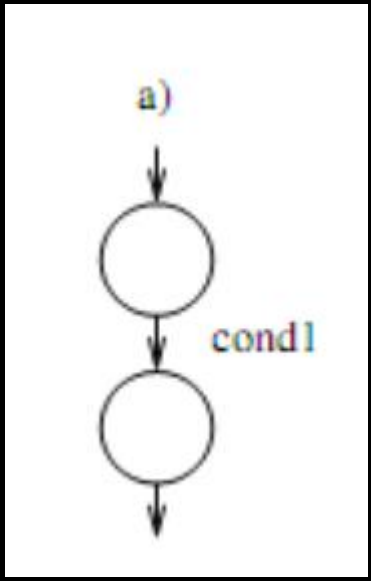
本小节我们将探讨如何将状态图映射到基于 uXos 的结构中去

下图所示为上面这三种基本模型的 uXos的实现伪代码:

用uXos开发项目

顺序模式

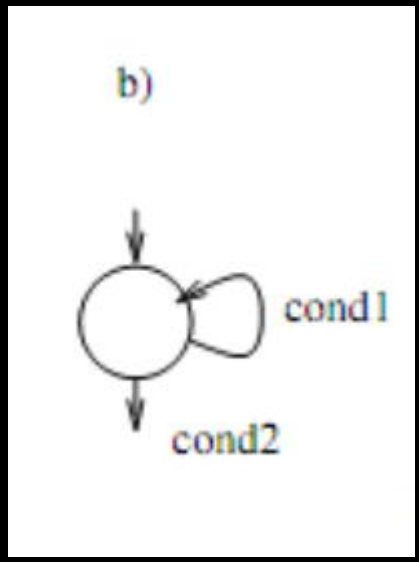
```
uXOS_BEGIN();

Function1();
uXOS_WAIT_UNTIL( cond1 );
Function2();

uXOS_END();
```



用uXos开发项目

迭代模式

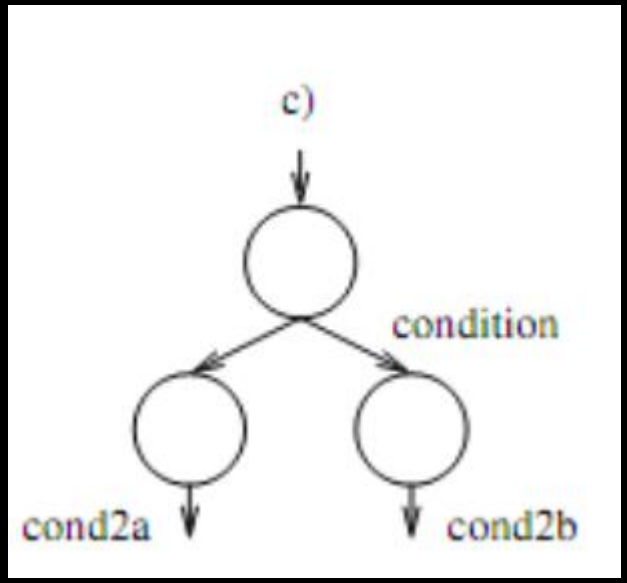
```
uXOS_BEGIN();

Function1();
while( cond1 )
        uXOS_WAIT_UNTIL( cond1 or cond2 );
Function2();

uXOS_END();
```



用uXos开发项目

选择模式

```
uXOS_BEGIN();
Function1();
if(cond1){
        uXOS_WAIT_UNTIL( cond2a );
        Function2a();
}
else{
        uXOS_WAIT_UNTIL( cond2b );
        Function2b();
}
uXOS_END();
```



用uXos开发项目

uXOS程序结构

以按键处理为例：

```
uXOS_THREAD( KeyProcess )
{
        uXOS_BEGIN();

        while(1)
        {
                uXOS_DELAY(  10  ):
                g_KeyInfo.CurKey = GetKey( );
                KeyModuleProcess( &g_KeyInfo );
        }

        uXOS_END();
}
```

任务以uXOS_THREAD宏开始

10ms采集一次按键

用uXos开发项目

主程序结构

```
void  main( void )
{
        initDevices();
        os_init( fnTickCallBack );

        create_task( keyProcess  , 1 );
        create_task( dispProcess , 2 );
        create_task( ctrlProcess  , 10 );

        os_start();
}
```

系统回调函数，用于执行高优先级的代码

添加要执行的任务，并指定优先级

启动系统调度执行

用uXos开发项目

uXOS使用两种软件定时器: etimer, utimer

eTimer:

用来为系统或应用层提供一些超时，定时服务。他提供了两个API接口，其函数原型为:

```
void etimer_set(TM *tm, TICK timeout );
BYTE etimer_expired(TM *tm );
```

etimer定时器数据结构为TM，使用时定义一个结构体，如:

```
TM et;
etimer_set( &et,    200 );   ----- 设置eTimer超时时间为200 TICK
etimer_expired( &et )        ----- 超时时此函数返回1，否者为0
```

用uXos开发项目

utimer:

用来为系统或应用层提供周期性或一次性的回调任务，如通信上要随机延时调用发送函数发送数据包等。其函数原型如下：

```
void    utimer_init( void );
void    utimer_stop( struct uTimer *timer );
void    utimer_start( struct uTimer *timer );
void    utimer_delete( struct uTimer *timer );
struct uTimer* utimer_create( void (*cb)(void),TICK timeout, BYTE period );
void   utimer_ctrl( struct uTimer *timer,BYTE cmd, void* arg );
struct uTimer* utimer_callback( void (*cb)(void), TICK timeout, BYTE option );
```

用uXos开发项目

utimer函数说明:

void utimer_init(void);

utimer的初始化，使用时调用此函数。

void utimer_stop(struct uTimer *timer);

停止定时器

void utimer_start(struct uTimer *timer);

启动定时器

void utimer_delete(struct uTimer *timer);

删除定时器

struct uTimer* utimer_create(void (*cb)(void),TICK timeout, BYTE period);

创建一个定时器

void utimer_ctrl(struct uTimer *timer,BYTE cmd, void* arg);

设置定时器参数

struct uTimer* utimer_callback(void (*cb)(void), TICK timeout, BYTE option);

定时器回调: cb 是回调函数，timeout是超时间隔，option为一次还是周期性

uXOS提供的这两种软件定时器是十分有用的功能，灵活应用功能将十分强大。

用uXos开发项目

eTimer的使用---- 超时处理:

```
TM    et;
BYTE  t;

for( i = 0;  i < 3;  i++  )
{
        SendPacket( “0123456789",10 );
        etimer_set( &et,   200 );

        uXOS_WAIT_UNTIL( AckReceived( ) ||          // 等待应答
                (t = etimer_expired( &et )) );      // 如果超时重发，三次无效退出

        if( !t )
        {
                break;
        }
}
```

设置eTimer定时器，200ms超时

200ms超时，etimer_expired函数返回1

eTimer定时器主要用来处理程序的超时，延时等，如上面的通信超时处理。

用uXos开发项目

utimer的使用

```
struct uTimer* t;
utimer_init( );   //当要使用utimer时，初始化它

 t = utimer_callback( sysTimeTick  ，500, UTIMER_PERIODIC );
      500ms周期性执行sysTimeTick函数
 utimer_callback( SetPanelStatus, 5000, UTIMER_PERIODIC );
      5000ms周期性执行SetPanelStatus函数
utimer_callback( SetPanelStatus, 3000, UTIMER_ONESHOT );
    3秒后执行SetPanelStatus函数（只执行一次，自动删除定时器）

 utimer_delete( t);
    删除 t 定时器
```

utimer定时器主要用来调用一次性或周期性的任务而设计的

用uXos开发项目

消息队列：生产者和消费者模型

```
struct MSG_Q *app_msg;
static BYTE   msgQ_data[8];

uXOS_MSGQ_INIT( app_msg, msgQ_data, 1, sizeof(msgQ_data) );
```

```
uXOS_THREAD( postMsgProcess )
{
        BYTE    ret;

        uXOS_BEGIN();

        while(1)
        {
            ret = process();

            uXOS_MSGQ_POST( app_msg, &ret );

            ...

            uXOS_YIELD();

        }

        uXOS_END();
}
```


```
uXOS_THREAD( revMsgProcess )
{
        static BYTE    msg;

        uXOS_BEGIN();

        while(1)
        {
            uXOS_MSGQ_WAIT( app_msg , &msg  );

            ...

        }

        uXOS_END();
}
```

用uXos开发项目

信号：一个类似QT的信号和槽函数的功能

signal api：

```
int         connect( SIGNAL sig, void *slot );
int         disconnect( SIGNAL sig );
void        emit( SIGNAL sig, int arg_num, ... );
void        signal_slot_init( void );
SIGNAL  create_signal( void );
```

用uXos开发项目

信号：一个类似QT的信号和槽函数的功能

```
#include “signal.h“

signal_slot_init();

SIGNAL sig = create_signal();

connect( sig, SLOT(test1_slot) );
connect( sig, SLOT(test2_slot) );

emit( sig, 2, 5, 10 );
emit( sig, 1, 50 );
```

信号触发时，连接该信号的槽函数都会被调用

创建一个可用的信号

连接信号与槽函数

发送信号

信号与槽函数的参数不一定要匹配，但要注意不要取没有匹配的参数

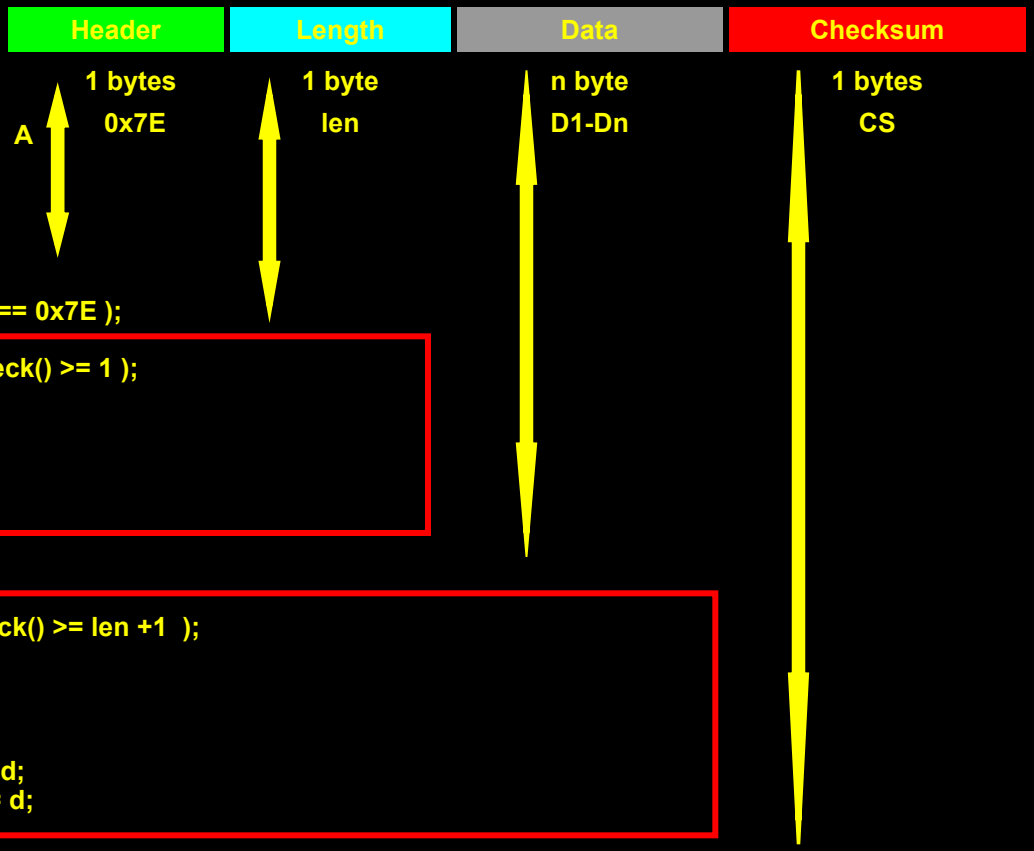
```
槽函数：
void test1_slot( int  x )
{
    ...
}
```

```
槽函数：
void test2_slot( int  x, int y )
{
    ...
}
```

用uXos开发项目

uXos在通信系统上的应用

解析数据包




```
uXOS_THREAD( revPacketProcess )
{
        static BYTE    d, len,i;
        static BYTE    CheckSum;

        uXOS_BEGIN();

        uXOS_WAIT_UNTIL( uartGetc() == 0x7E );

        uXOS_WAIT_UNTIL( fifoCntCheck() >= 1 );

        d   = uartGetc();
        revPacket[0] = d;

        len = d;

        CheckSum = 0;

        uXOS_WAIT_UNTIL( fifoCntCheck() >= len +1  );

        for( i = 0; i < len ; i++ )
        {
                d = uartGetc();
                revPacket[i+1] = d;
                CheckSum      += d;
        }
        d  = uartGetc();

        if( CheckSum != d )
        {
                uXOS_RESTART();
        }

        uXOS_END();
}
```

用uXos开发项目

```
uXOS_THREAD( commProcess )
{
    static BYTE          i,CmdIndx;
    static TASK_TCB   chlidTcb;

    uXOS_BEGIN();
    InitUart();

    while( 1 )
    {
        uXOS_SPAWN( &chlidTcb, revPacketProcess );     // 处理接收到的数据包
        CmdID   = GetRxPacketCmd();                    // 解出命令
        Length  = GetRxPacketDatalength();             // 数据包的长度
        dataPtr = GetRxPacketData();                   // 数据区的指针
        if( (CmdID >= 0xA0)&&(CmdID <= 0xAD ) )        // 命令范围判断
        {
            CmdIndx = CmdID - 0xA0;                    // 计算执行函数索引号
            (*MsgPacketReqEntry[CmdIndx])( CmdID, Length, dataPtr );
        }
    }
    uXOS_END();
```

没有收到正确的数据包会阻塞在这里

收到正确的数据包后查找对应的应答包

用uXos开发项目

应答包处理入口

```
static void ( *MsgPacketReqEntry[] )( BYTE CmdId,  BYTE len,   BYTE* dataPtr  )=
{
        msgSetSysOnOffReq       ,   // CmdID 0xA0
        msgSetTempReq           ,   // CmdID 0xA1
        reserved                ,   // CmdID 0xA2
        reserved        ,   // CmdID 0xA3
        msgSetDelayReq          ,   // CmdID 0xA4
        msgSetTime              ,   // CmdID 0xA5
        msgQueryDeviceReq       ,   // CmdID 0xA6
        msgQueryFuncReq         ,   // CmdID 0xA7
        reserved        ,   // CmdID 0xA8
        msgQueryBathTimeReq     ,   // CmdID 0xA9
        reserved        ,   // CmdID 0xAA
        msgQueryDelayReq        ,   // CmdID 0xAB
        reserved        ,   // CmdID 0xAC
        msgQueryAllStateReq     ,   // CmdID 0xAD
        reserved                    // reserved
};
```

应答包列表（函数入口地址）

用uXos开发项目

用uXos开发项目

谢谢！