in Sys/init/sys_init.c/SysApi_InitApplModules()

```
    HalApiIsr_Register(true, false, HAL_ISR_TYPE_TIM, SysTimerIrq_Isr);
    HalApiIsr_Unmask(false, HAL_ISR_TYPE_TIM);      // arm interrupts are supposed to be currently disabled.
```

HAL_ISR_TYPE_TIM就是tick interrupt

in Hal/6220/src/hal)isr.c

```
struct isr_array_struct {
    uint32_t        type;
    void(*isr)(void);           // for EPU_ISRC bits
    bool            fiq;            // true if this handler shall be mapped to fiq handling
    bool            enabled;        // true if this handler is enabled - please note that an unmasked isr must not be disabled.
    uint32_t        max_cpu_cntr_execution_time;
    uint32_t        int_num;        // irq_bits of different isr_array members must be disjunct
} isr_array[] = {
        {HAL_ISR_TYPE_TIM,          NULL, false, false, 0, INTNUM_TIMEBASE_10MS},       // TIM0
        {HAL_ISR_TYPE_UART,             NULL, false, false, 0, INTNUM_UART_INT_UART2},
  ......
    };
uint32_t MAX_ISR_TYPES = sizeof(isr_array)/sizeof(struct isr_array_struct);
```

tick是10ms,即每秒100 tick.

SysTimerIrq_Isr()是scheduler

Sys/timer/sys_timer_irq.c/SysTimerIrq_Isr()

```
void SysTimerIrq_Isr(void)      // this will run in interrupt kontext
{

    static uint32_t last_os_tick_count = 0;

    /* maintain an exact counter of os ticks. */
    HalApiCpuTimer_UpdateTickCount(false);


    if (last_os_tick_count + (RTOS_TICK_DIVIDER-1) < HalApiCpuTimer_os_tick_count()) {

        last_os_tick_count = HalApiCpuTimer_os_tick_count();

#ifdef OS_threadx
        _tx_timer_interrupt();
#elif defined OS_freertos
        SysTick_IRQHandler();   // RTOS ThreadX timer pulse
#endif


#define SYNCHRONIZE_FREERTOS_CLOCK
#ifdef SYNCHRONIZE_FREERTOS_CLOCK
        // shall/must we do this?
        if (HalApiCpuTimer_os_tick_count() != xTaskGetTickCountFromISR()) {
            vTaskSetTickCount(HalApiCpuTimer_os_tick_count(), true);
        }
#endif


        /* add stable link status detection */
        if(DeferredLinkEvent > 0)        DeferredLinkEvent--;
        if(DeferredLinkEvent == 1) {
            DeferredLinkEvent--;
            SysSystem_SystemNotifyLinkUp();
        }
        if( GlPhyState != HalApiLan_LinkStateGet())  {
            GlPhyState = HalApiLan_LinkStateGet();
#if 0
            /* for now we will let this activated - as long as it does any harm. */
            SysApiDebug_PrintfIRQ(SYS_DBGCOMP_ALL, SYS_DBGCAT_ALL,
                "%s ticks=%08x 2800=%04x 0f04=%08x\n", __func__,
                HalApiGet_CpuCounter(),
                HalLan_RegisterRead2(0x2800),
                HalLan_RegisterRead(0x0f04));
#endif
            if(GlPhyState == HAL_LINKSTATE_NOLINK)   {
                if (DeferredLinkEvent)
                    DeferredLinkEvent = 0;              // there are cases where the link flickers (at 10mbit)
                else
                    SysSystem_SystemNotifyLinkDown();
```

```
            } else  {
                /* the link up event is deferred because of the mac-auto-upd
ate feature. */
                if (!DeferredLinkEvent) {   /* only setup the counter if the
re wasnt already
                                        a link up event for the last deferri
ng period */

                    // this depends on OS_TICK_IN_NS
                    DeferredLinkEvent = 50;  //  ~490 ms
                }
            }
        }

    }


    HalApiUpdate_CpuTimer(OS_TICK_IN_NS/RTOS_TICK_DIVIDER);

}
```

in Os/freertos/portable/GCC/6220/port.c

```
void SysTick_IRQHandler( void )     // this may be the wrong place -> sys_ti
mer* stuff?
{
/*  OS_DBG_MSG(APP_DBGCOMP_ALL, APP_DBGCAT_WARN,"SysTick_IRQHandler() called
 \n");*/

/*  if(configUSE_TIMER->SR & TIM_FLAG_OC1) */
    {
        /* Reset the timer counter to avoid overflow. */
        //configUSE_TIMER->OC1R += s_nPulseLength;

        /* Increment the tick counter. */
        xTaskIncrementTick();

        #if configUSE_PREEMPTION == 1
        {
            /* The new tick value might unblock a task.  Ensure the highest
task that
            is ready to execute is the task that will execute when the tick
ISR
            exits. */
            vTaskSwitchContext();
        }
        #endif

        //configUSE_TIMER->SR &= ~TIM_FLAG_OC1;
    }

}
```

vTaskSwitchContext()是freertos提供的code.
该function只完成选择下一个task(并不真正switch context)!

> PRIVILEGED_DATA TCB_t * volatile pxCurrentTCB = NULL;

使得 `pxCurrentTCB` 指向要切换过去的task的pTCB.

freertos下context switch的情景如下(假设从task A ==> task B)

<1>. pxCurrentTCB -> task A
<2>. trigger tick interrupt
<3>. in Hal/6220/build/hal_vectors_gnu.asm

```
1.  IrqHandler:
2.      B   IrqHandler_second_stage
```

in Hal/6220/build/hal_2nd_init_gnu.asm

```
1.      .global IrqHandler_second_stage
2.  IrqHandler_second_stage:
3.      portSAVE_CONTEXT
4.  @   bl  Hal_Isr
5.    bl      irq_handler
6.      portRESTORE_CONTEXT
```

<4>. portSAVE_CONTEXT macro save task A's context
<5>. irq_handler() invoke vTaskSwitchContext()
pxCurrentTCB指向task B
<6>. portRESTORE_CONTEXT macro restore task B's context

另外，如果FreeRTOS要yield CPU，它是通过trigger "SWI 0"来实现的。

```
1.  #define taskYIELD_IF_USING_PREEMPTION() portYIELD_WITHIN_API()
2.
3.  #define portYIELD_WITHIN_API portYIELD
4.
5.  #define portYIELD()                 __asm volatile ( "SWI 0" )
```

而SWI的handler is in hal_vectors_gnu.asm

```
1.  SwiHandler:
2.      B   SwiHandler_second_stage
```

```
    .global SwiHandler_second_stage
SwiHandler_second_stage:
    add lr, lr, #4
    portSAVE_CONTEXT
    ldr     r0, =vTaskSwitchContext
    mov     lr, pc
    bx  r0
    portRESTORE_CONTEXT
```

当"SWI 0"返回时，context已经switched.