How PendSV is used for Context Switching in FreeRTOS ??

Free RTOS

In this article let me explain, line by line, the PendSV handler which carry out the Context switch form one task to another task. That is nothing but saving the context of current task (Context is nothing but few processor registers, that's it) and retrieving the context of the next potential task.

So, here is the handler code, I copied from the **port.c** file of **ARM Cortex M3.**

If you want to locate this file then, you have to go to,

<FreeRTOs installation path>\FreeRTOS\FreeRTOSv9.0.0\FreeRTOS\Source\portable\GCC\ARM_CM3\

you can download and install the FreeRTOS from here:

https://sourceforge.net/projects/freertos/

Recent Posts

- FreeRTOS Lecture 40 – Exercise: Usage of taskYIELD() and explanation
- FreeRTOS Lecture 39 – Exercise: Testing our hello world application over UART
- FreeRTOS Lecture 38- Exercise: Testing UART prints
- FreeRTOS Lecture 37 – Exercise: UART Printmsg implementation using std periph. library
- FreeRTOS Lecture 36 – Exercise: UART Parameter Inits

```
void xPortPendSVHandler( void )
{
    /* This is a naked function. */

    __asm volatile
    (
    "   mrs r0, psp                         \n"
    "   isb                                 \n"
    "                                       \n"
    "   ldr r3, pxCurrentTCBConst           \n" /*
    "   ldr r2, [r3]                        \n"
    "                                       \n"
    "   stmdb r0!, {r4-r11}                 \n" /*
    "   str r0, [r2]                        \n" /*
    "                                       \n"
    "   stmdb sp!, {r3, r14}                \n"
    "   mov r0, %0                          \n"
    "   msr basepri, r0                     \n"
    "   bl vTaskSwitchContext               \n"
    "   mov r0, #0                          \n"
    "   msr basepri, r0                     \n"
    "   ldmia sp!, {r3, r14}                \n"
    "                                       \n" /*
    "   ldr r1, [r3]                        \n"
    "   ldr r0, [r1]                        \n" /*
    "   ldmia r0!, {r4-r11}                 \n" /*
    "   msr psp, r0                         \n"
    "   isb                                 \n"
    "   bx r14                              \n"
    "                                       \n"
    "   .align 4                            \n"
    "pxCurrentTCBConst: .word pxCurrentTCB  \n"
    ::"i"(configMAX_SYSCALL_INTERRUPT_PRIORITY)
    );
}
```

Now, let's explore this line by line.
First, i will write the code and below that i will give the explanation !

```
"   mrs R0, PSP                         \n"
```

Here, they are just saving the PSP value in to R0. Why? Let's see.

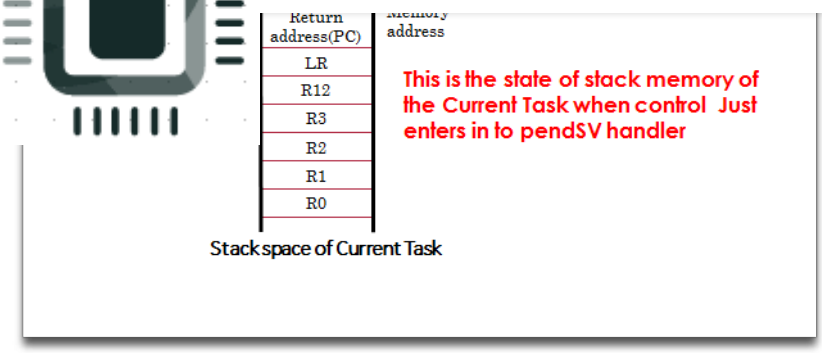
- June 2021
- May 2021
- March 2021
- February 2021
- January 2021
- December 2020
- November 2020
- August 2019
- July 2019
- June 2019

Categories

- Blog

Meta

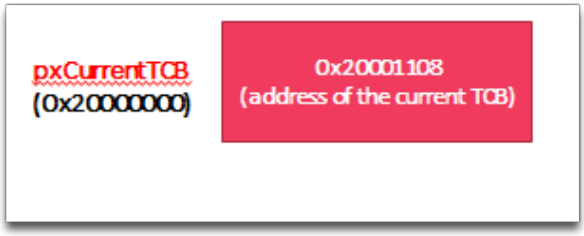
- Log in
- Entries RSS
- Comments RSS
- WordPress.org




```
1 "   ldr R3, pxCurrentTCBConst                \n"
```

Now, in this line, the address of the global variable "pxCurrentTCB" is stored in R3 register. "pxCurrentTCB" is a global variable defined in tasks.c (go ahead and check tasks.c) which holds the address of the Currently executing task's TCB.

Now, let's just assume that the address of the pxCurrentTCB is 0x20000000. And it is holding the address 0x20001108 as shown below.



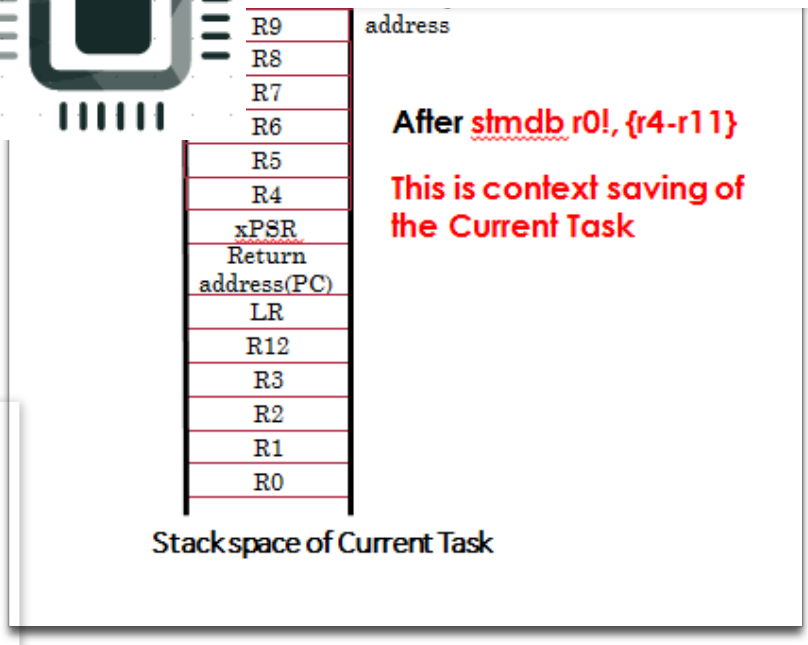
So, now **R3** = 0x20000000

```
1 "   ldr R2, [R3]                             \n"
```

(De-reference 0x20000000 and store the value into R2)
So, R2 = 0x20001108 (Address of the Current TCB)

```
1 "   stmdb R0!, {R4-R11}                      \n"
```

Now , R0 has PSP value right ?? So, using that PSP, we are saving the registers contents R4 to R11 on to the tasks stack memory. So, currently R4 to R11 are safe.




```
1 "    str R0, [R2]                                    \n"
```

(De-reference 0x20001108 and store the value in to R0)
What is 0x20001108? It's the address of the current TCB stored in pxCurrentTCB pointer.

That is, pxCurrentTCB is nothing but a pointer of type "tskTCB".
tskTCB is the TCB structure which is defined in tasks.c.
Go Go Go check tasks.c !!! . You will find the below structure.

```
1  /*
2   * Task control block.  A task control block (TCB)
3   * and stores task state information, including a p
4   * (the task's run time environment, including regi
5   */
6  typedef struct tskTaskControlBlock
7  {
8      volatile StackType_t    *pxTopOfStack;  /*< Poi
9
10     #if ( portUSING_MPU_WRAPPERS == 1 )
11         xMPU_SETTINGS   xMPUSettings;       /*< The
12     #endif
13
14     ListItem_t          xStateListItem; /*< The lis
15     ListItem_t          xEventListItem;     /*< Use
16     UBaseType_t         uxPriority;         /*< The
17     StackType_t         *pxStack;           /*< Poi
18     char                pcTaskName[ configMAX_TASK_
19
20     #if ( portSTACK_GROWTH > 0 )
21         StackType_t     *pxEndOfStack;      /*< Poi
22     #endif
23
24     #if ( portCRITICAL_NESTING_IN_TCB == 1 )
25         UBaseType_t     uxCriticalNesting;  /*< Hol
26     #endif
```

```
nfigUSE_MUTEXES == 1 )
    eType_t     uxBasePriority;     /*< The
    eType_t     uxMutexesHeld;
#endif

#if ( configUSE_APPLICATION_TASK_TAG == 1 )
    TaskHookFunction_t pxTaskTag;
#endif
#if( configNUM_THREAD_LOCAL_STORAGE_POINTERS > 0 )
    void *pvThreadLocalStoragePointers[ configN
#endif

#if( configGENERATE_RUN_TIME_STATS == 1 )
    uint32_t        ulRunTimeCounter;   /*< Sto
#endif

#if ( configUSE_NEWLIB_REENTRANT == 1 )
    /* Allocate a Newlib reent structure that i
    Note Newlib support has been included by po
    used by the FreeRTOS maintainers themselves
    responsible for resulting newlib operation.
    newlib and must provide system-wide impleme
    stubs. Be warned that (at the time of writi
    implements a system-wide malloc() that must
    struct  _reent xNewLib_reent;
#endif

#if( configUSE_TASK_NOTIFICATIONS == 1 )
    volatile uint32_t ulNotifiedValue;
    volatile uint8_t ucNotifyState;
#endif

/* See the comments above the definition of
tskSTATIC_AND_DYNAMIC_ALLOCATION_POSSIBLE. */
#if( tskSTATIC_AND_DYNAMIC_ALLOCATION_POSSIBLE
    uint8_t ucStaticallyAllocated;      /*< Set
#endif

#if( INCLUDE_xTaskAbortDelay == 1 )
    uint8_t ucDelayAborted;
#endif

} tskTCB;
```

In this structure take look in to the first member. What's that??
It's another pointer isn't? Or you can say a place holder, which holds address of the stack where the last item is placed by this task and remember that every task will have its own stack space.

So, the first member just holds the address of the last item present in the stack.

After "**stmdb R0!, {R4-R11} , R0**" will be pointing to the new top of the stack isn't it ??

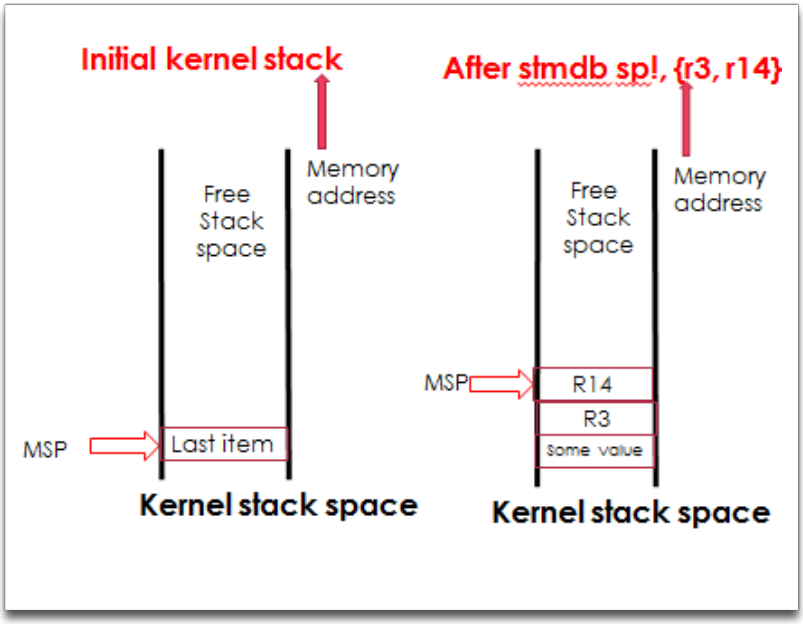
.e. value in the register R0.

used to restore the stack pointer, when this task switches in later point in time.

Ok. let's move ahead.

```
1  "    stmdb sp!, {R3, R14}                    \n"
```

We are going to branch out after this, that's why, save R3 and R14 for a time being.
R0 is already saved, no worries!
R1 is not required because we didn't used it so far,
R2 was just a place holder, now it's not required, so only save R3 and R14.
Note that here SP being used, since we are in the handler mode of the processor, SP is nothing but MSP (Main Stack Pointer).
So, we just pushed to kernels stack memory.

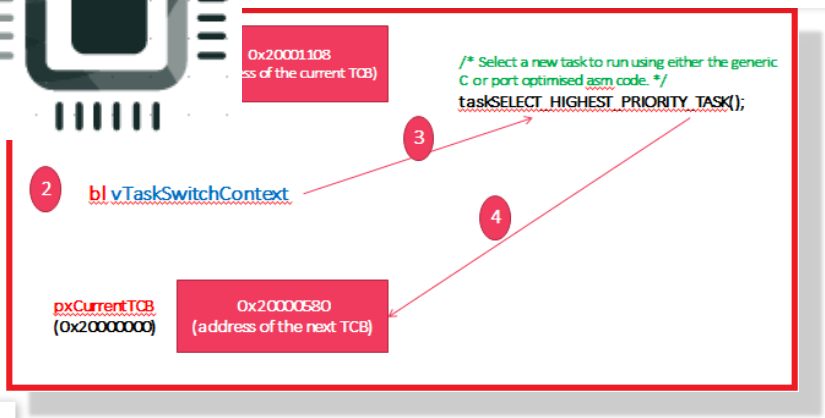



```
1  "    mov R0, %0                              \n"
2  "    msr basepri, R0                           \n"
```

/* disable the interrupts, */

```
1  "    bl vTaskSwitchContext                   \n"
```

branch out to **vTaskSwitchContext.**
vTaskSwitchContext is defined in tasks.c , which selects the next potential task based on priority .



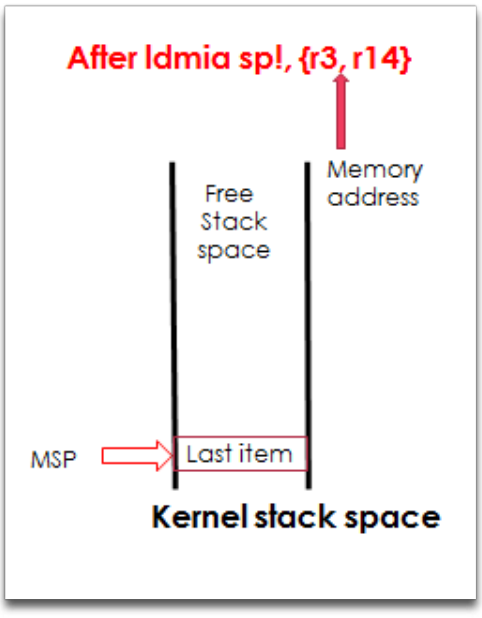
1. Initially the global pointer **pxCurrentTCB** was holding the address of the Current TCB .i.e **0x20001108**
2. After that call is made to **vTaskSwitchContext**
3. Inside **vTaskSwitchContext** , there is a call to **taskSELECT_HIGHEST_PRIORITY_TASK()**
4. Which changes the **pxCurrentTCB** value to the address of the next potential TCB

```
1 "   mov R0, #0                                \n"
2 "   msr basepri, R0
```

Disables the interrupt again, who knows what happened to basepri when we branched out above */

```
1 "   ldmia sp!, {R3, R14}                      \n"
```

POP R3 , R14.
Remember that R3 = 0X20000000




```
1 "   ldr R1, [R3]                              \n"
```


```
1 "   ldr R0, [R1]                    \n"
```

Dereference 0x20000580 and store the value in to R0 .

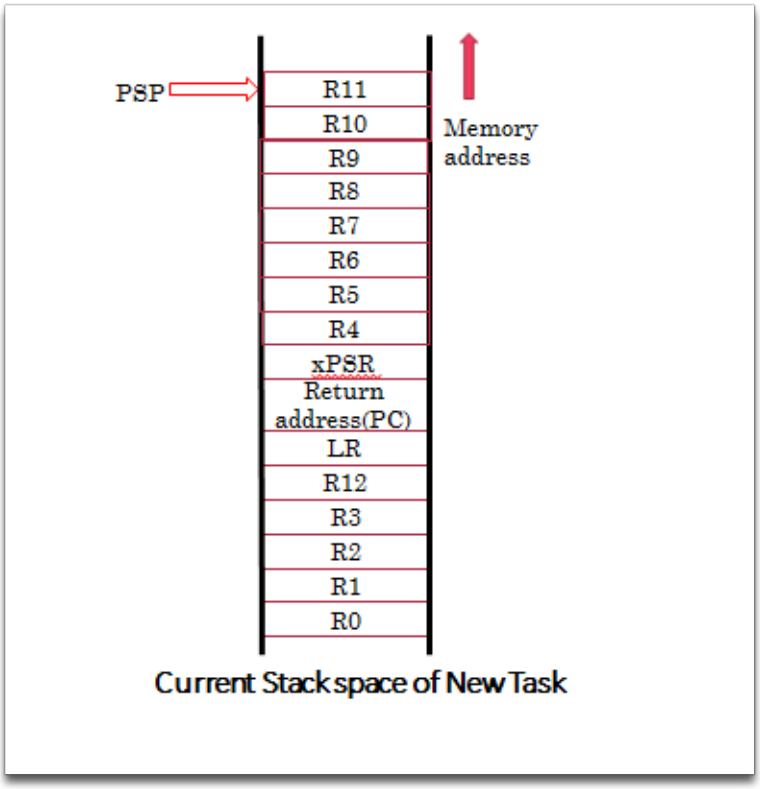
That means, store the value of the first member of the new TCB.
As you already know, first member is nothing but top of the stack.

So, R0 = Top of the stack of new Task, which is selected for switching in.

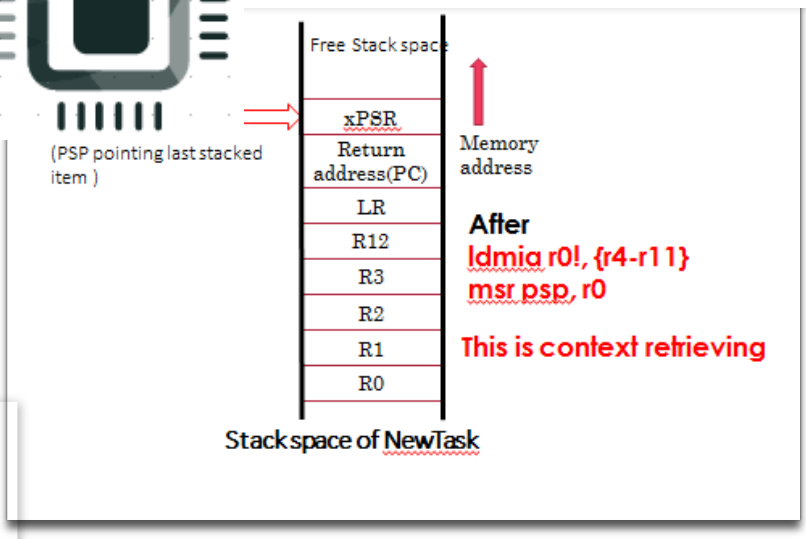
```
1 "   ldmia R0!, {R4-R11}             \n"
```

So, we know the top of the stack;
Let's use that to retrieve the context of new Task.

This is the Current Stack state of the New Task which is about to switch in!



This is the Current Stack state of the New Task after "ldmia R0!, {R4-R11}"



So, context of the new task is retrieved to register **R4** to **R11**.

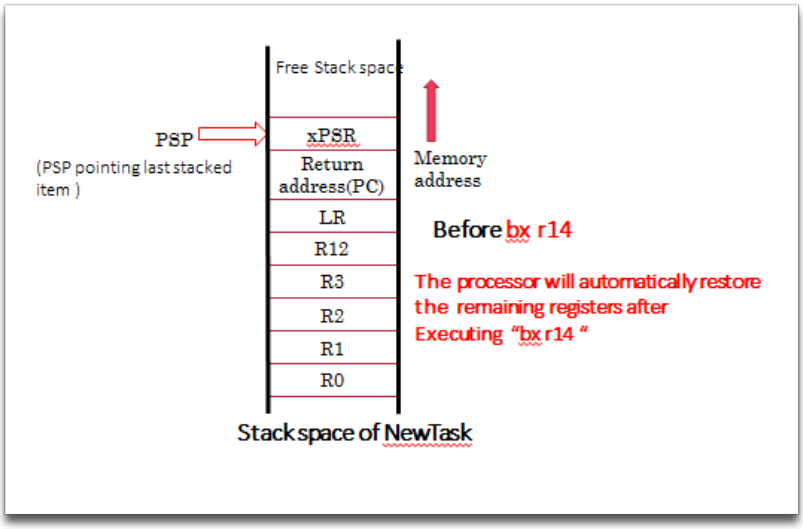
```
1 "    msr PSP, R0                    \n"
```

Now, let's initialize the PSP with the R0 value.

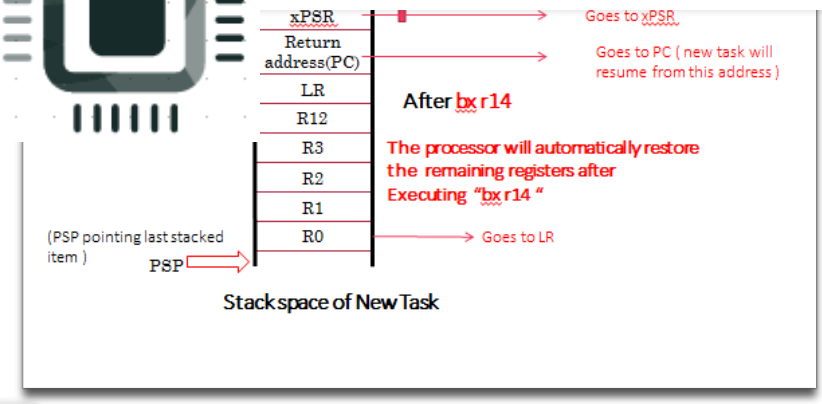
```
1 "    bx R14                         \n"
```

Return from the handler!

Before executing "bx R14"



After executing bx R14



So, the task will resume to the instruction during which it was switched out of the running state.

> ***Note: I have omitted explaining of "ISB" instruction. Please, read from here***

http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0552a/CHDEBIEG.html

Get the Full Course on FreeRTOS Porting and programming Here.

If you like this article, please consider sharing it in linkedin by clicking below.

Thanks.

By FastBitLab | June 28th, 2019

Share This Story, Choose Your Platform!

Related Posts

taskYIELD() and explanation

June 8th, 2021 | 0 Comments

FreeRTOS Lecture 39 – Exercise: Testing our hello world application over UART

June 7th, 2021 | 0 Comments

FreeRTOS Exercise: prints

June 7th, Commen

Leave A Comment

Comment...

Name (required) Email (required) Website

POST COMMENT

Yes, Add me to your email course

Sign me up for the newsletter