Fully Preemptive nxtOSEK Kernel with Preemption Threshold Scheduling

Jimit Doshi (jdoshi@ncsu.edu) and Saransh Gupta (sgupta20@ncsu.edu)

We present below the work done so far to enable nested preemptions on the nxtOSEK kernel.

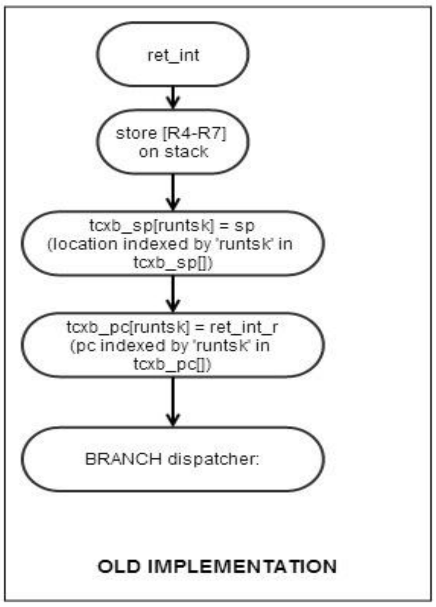
Algorithm:

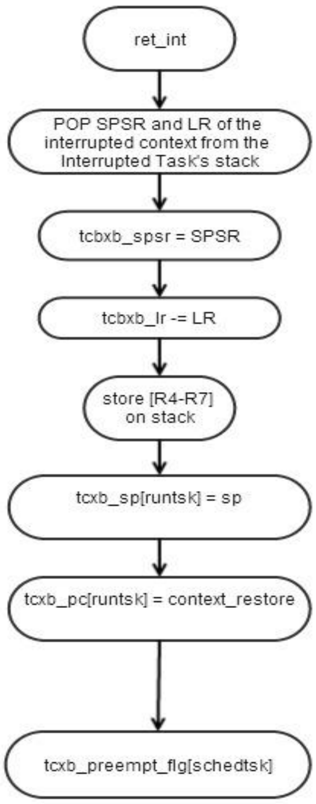
After analyzing the original algorithm, we have come up with a new approach to allow nested preemptions.

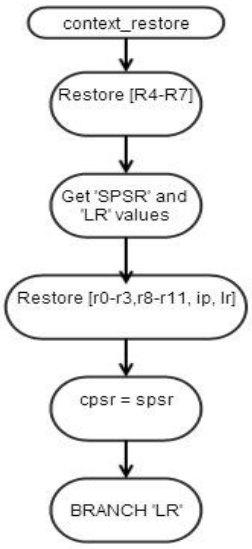
Summary of the new program flow:

1. We create 3 new data structures in the task control block as below :
    a. tcxb_spsr[] : To store the SPSR value of the interrupted task
    b. tcxb_lr[] : To store the return address of the interrupted task i.e. address of the location where a given tasks was preempted
    c. tcxb_preempt_flg []: Flag used to indicate preemption. Whenever a high priority task preempts a lower priority task, the high priority task's tcxb_preempt_flg is SET.
2. While running the kernel assembly code in ISR, if it is realized that (runtsk! = schedtsk), then we set the tcxb_preempt_flg[schedtsk] = 1. Also, the interrupted task's spsr and lr values are saved in its tcxb_spsr & tcxb_lr locations respectively. tcxb_pc of runtsk is set to 'context_restore'.
3. After this, the 'int_return_preemption' code is run where the ISR is 'returned' to a high priority task. *This is in contrast to the previous execution where the preempting task is run while still the ISR has not yet returned, which was the main reason why nested preemptions were not possible with that approach.*
4. Subsequently, whenever the preempted task is scheduled again, first its context is restored from 'context_restore'. Thereafter, SPSR is restored and then pc is set to LR value.

<u>Flowchart & Code Snippets:</u>


ret_int
store [R4-R7] on stack
tcxb_sp[runtsk] = sp (location indexed by 'runtsk' in tcxb_sp[])
tcxb_pc[runtsk] = ret_int_r (pc indexed by 'runtsk' in tcxb_pc[])
BRANCH dispatcher:
OLD IMPLEMENTATION


ret_int
POP SPSR and LR of the interrupted context from the Interrupted Task's stack
tcbxb_spsr = SPSR
tcbxb_lr -= LR
store [R4-R7] on stack
tcxb_sp[runtsk] = sp
tcxb_pc[runtsk] = context_restore
tcxb_preempt_flg[schedtsk]

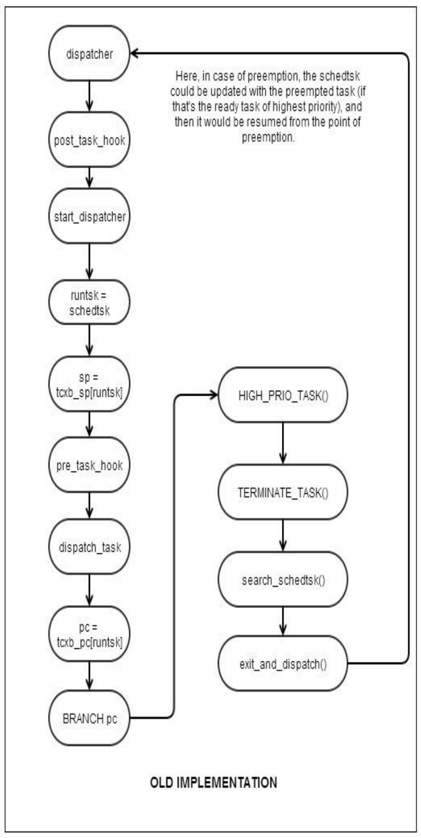


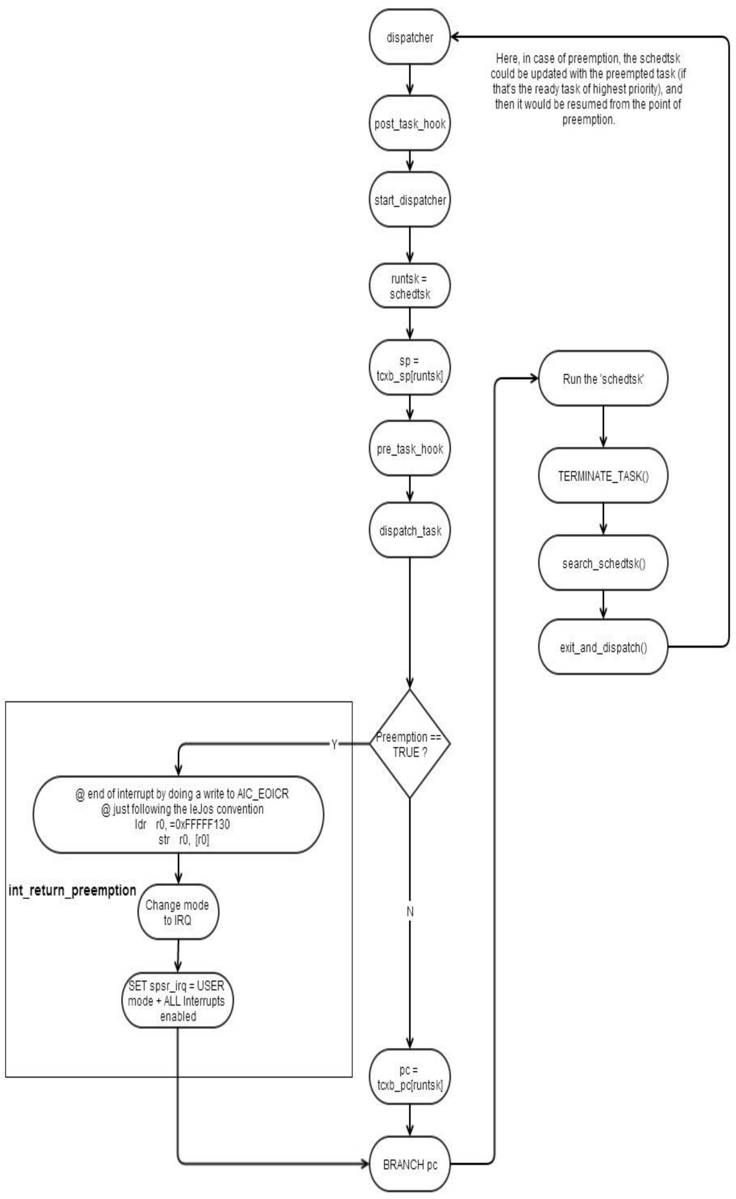
```
context_restore:
        ldmfd   sp!, {r4-r7}                    // restore non destructive registers
        b       task_return

task_return:
        @get spsr and lr in r0, r1
        ldmfd   sp!, {r0,r14} //r0 = spsr, r14 = lr
        mov     r9, sp        // save task stack sp in r9
        ldr     sp, =__system_stack__ //Store spsr and lr on _system_stack
        bic     sp, sp, #7
        stmfd   sp!, {r0-r1}
        mov     sp,r9       //Change sp again to task stack
        ldmfd   sp!, {r0-r3,r8-r11, ip, lr}

ret_int:
        ldr     r0, =runtsk                     //  tcxb_sp[runtsk] = sp
        ldrb    r0, [r0]
        ldr     r1, =tcxb_spsr
        ldmfd   sp!, {r2,r3}                    //r2 = spsr, r3 = lr
        str     r2, [r1, r0, asl #2]
        ldr     r1, =tcxb_lr
        str     r3, [r1, r0, asl #2]
        stmfd   sp!, {r4-r7}                    // save remaining nondestructive regs
        ldr     r1, =tcxb_sp
        str     sp, [r1, r0, asl #2]
```

```
    ldr      r1, =tcxb_pc                     // tcxb_pc[runtsk] = "context_restore"
    ldr      r2, =context_restore
    str      r2, [r1, r0, asl #2]
    ldr      r0, =schedtsk                    //  tcxb_preempt_flg[schedtsk] = 1
    ldrb     r0, [r0]
    ldr      r1, =tcxb_preempt_flg
mov     r2, #1
    str      r2, [r1, r0, asl #2]
    b        dispatcher                       // jump to "dispatcher"
```





```
@ end of interrupt by doing a write to AIC_EOICR
@ just following the leJos convention
ldr   r0, =0xFFFFF130
str   r0,  [r0]
```

```
dispatch_task:
        ldr     r0, =tcxb_pc
        ldr     r1, =runtsk
        ldrb    r1, [r1]
        ldr     r0, [r0, r1, asl #2]

        ldr     r2, = tcxb_preempt_flg
        ldr     r3, [r0, r1, asl #2]
        cmp     r3, #1
        beq     int_return_preemption
        //CHECK_PREEMPTION_FLAG -> IF FLAG IS TRUE, JUMP TO
        'int_return_preemption'
        bx      r0

int_return_preemption:
        @ end of interrupt by doing a write to AIC_EOICR
        @ just following the lejos convention
        ldr     r0, =0xFFFFF130
        str     r0,  [r0]
        @switch to irq stack
        msr     cpsr, #0XD2
        msr     spsr, #0x10 // USER mode + no flags
     stmfd      sp!, {r0}
     ldmfd      sp!, {pc}^        // Return from interrupt & mode change!
```

<u>Risks & Open Points:</u>

1. We haven't been able to figure out how to manage the program flow when we want to return back to the preempted task. We have to perform two operations at that time :
   a. Restore all the registers that are saved on the task's stack
   b. Set pc to the LR value stored in tcxb_lr[]

   To retrieve the LR value strored in tcxb_lr , we need atleast 2 registers. However, we can't use any register once their context has been restored from the stack. And hence, unless we get the LR value somehow without using any of the registers, we won't be able to jump back to the previous context. This is the only point remaining before we begin to test our implementation.

2. Implementation of PTS does not appear to be feasible at this point as it has taken considerable efforts to decode the existing program flow and then modify it to suit our needs: all in ARM assembly language.

Tasks Accomplished (with reference to the Project proposal)**:**

Task	Team Member
Design algorithm to resolve the preemption issue	Jimit
Design & implementation of test cases	Saransh
Documentation of understanding, preparation of interim report	Jimit & Saransh

Tasks in progress / planned:

Task	Team Member	Date
Resolve the implementation issues and run the test cases	Saransh & Jimit	4/25/2014
Final report preparation	Saransh & Jimit	4/27/2014

PTS implementation shall be done if time permits.