ECE254 Lab 2

October 2018

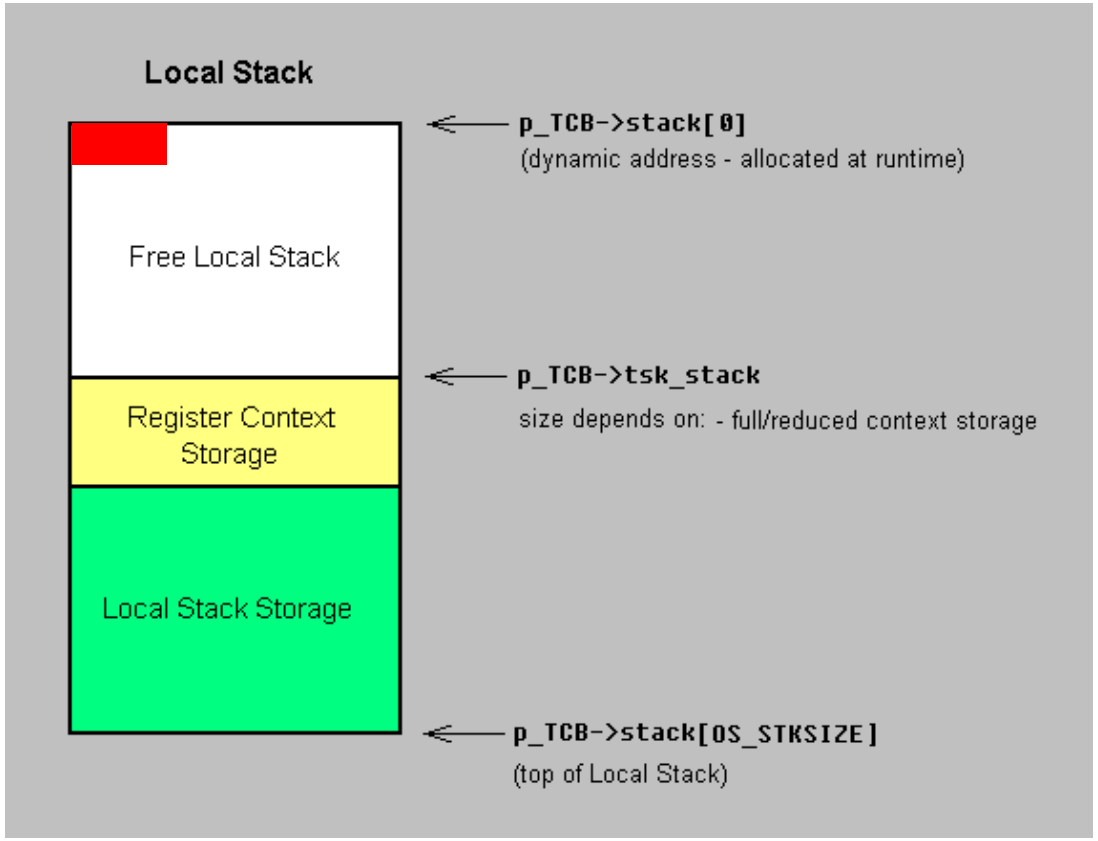
Administrative Information

- Lab Instructor: Jonathan Shahen (jmshahen@uwaterloo.ca)
- Lab 2 TA: Rui Zhou (r48zhou@uwaterloo.ca)

- **Lab 2 Due (Lab 203/206): Oct. 12th, 10:00 PM**
- **Lab 2 Due (Everyone Else): Oct. 19th, 10:00 PM**

Stack and MAGIC_WORD

- Stack[0] = MAGIC_WORD
- Calling a function places the parameters and return address on the stack
  - Stack pointer (PSP) stored in register, OS doesn't control pushing to stack
- What can happen if we go outside our allocate stack space?
- How can an OS stop this from occurring?



Goals of Lab 2

1. Work with Memory Pools
2. Work with Tasks in a Priority List
3. Changing a Task’s state
4. Blocking and Context Switching a RUNNING Task

Deliverable #1: Void *os_mem_alloc(void* mem_pool)

- **Kernel:** rt_mem_alloc() in rt_MemBox_ext.c
- **Assumption:** only 1 memory pool will be created by the user application; i.e. mem_pool will always be the same
- **Input:**
  - The starting address of the memory pool
- **Return:**
  - A pointer to an available memory block
- **Question:**
  - What if there is no memory block available when called?

Deliverable #2:
OS_RESULT os_mem_free(void *mem_pool, void* mem_box)

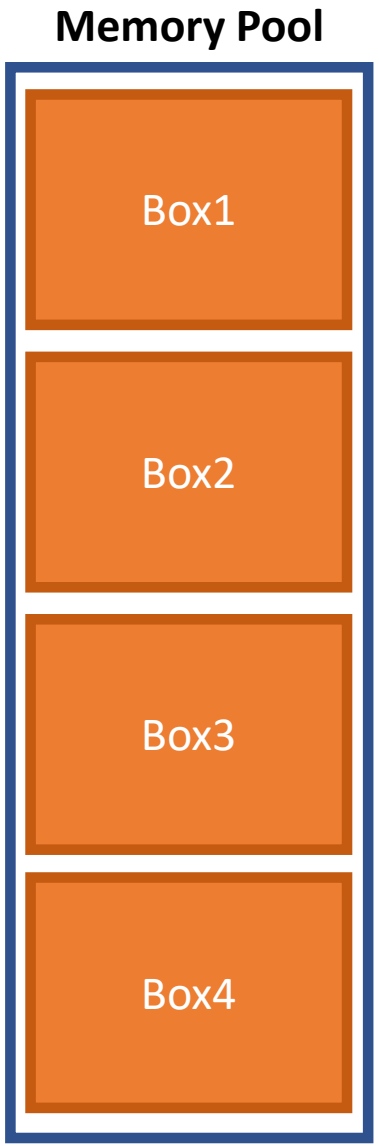
- **Kernel:** rt_mem_free() in rt_MemBox_ext.c
- **Assumption:** only 1 memory pool will be created by the user application; i.e. mem_pool will always be the same
- **Input:**
  - The starting address of the memory pool
  - Pointer to a memory block to be freed
- **Return:**
  - OS_R_OK – the function operated correctly
  - OS_R_NOK – an error occurred
- **Question:**
  - What if there are processes blocked waiting for memory?

Use Existing Kernel Functions

- The goal of this lab is task blocking and dispatching
- **rt_MemBox.c:**
  - void *rt_alloc_box(void *mem_pool)
  - int rt_free_box(void *mem_pool, void *box)
- **rt_List.c:**
  - void rt_put_prio (P_XCB p_CB, P_TCB p_task)
  - P_TCB rt_get_first (P_XCB p_CB)
- **rt_Task:**
  - void rt_block (U16 timeout, U8 block_state)
  - void rt_dispatch (P_TCB next_TCB)
- In **rt_Mailbox.c,** see rt_mbx_send() and rt_mbx_wait(), to see how the return value of function is set when a task is found waiting for a message and unblocked.

Lab 2 Steps

1. Create a global variable Memory Pool (Box Size, Num Boxes)
2. Initialize the Memory Pool in __*task init()*
3. T1: Use *os_mem_alloc()* to get box1
4. T1: Use *os_mem_alloc()* to get **box2**
5. T1: … box3 and box4
6. T2: Use *os_mem_alloc()* to have task blocked
7. T1: Use *os_mem_free(***box2***)* to unblock T2 and give **box2** to T2



Convenience Functions

- **Main_task_exp.c:** char *fp2name(void (*p)(), char *str)
  - Converts a function pointer (i.e. P_TCB->ptask) into a function name
- **Lab2_helper.c:** char *state2str(unsigned char state, char *str)
  - Converts a state (i.e. READY, WAIT_MEM) into a human readable string
- **Main_task_exp.c:** __task void task1(void)
  - Calls both functions you are going to implement (with NULL)
  - This is just a test to make sure you have everything connected correctly
- **Main_task_exp.c:** __task void task2(void)
  - Continuously prints out the TID, Task Name, Priority, and State of all tasks

Tip #1

1. Answer the Assignment Questions FIRST for this lab
   - It outlines where you should look and drops hints as to how to solve the lab
   - Refer back to the lab questions from Lab 1

2. **Create the Test Cases** at the end of 2.4.2 (**Test Driven Development**)
   - A task can allocate a memory box from a memory pool using os_mem_alloc()
   - A task will get blocked if there is no memory available when os_mem_alloc() is called.
   - A blocked task (state=WAIT_MEM) will be resumed once enough memory is available in the system.

Tip #2: Printf and Mutexes

- Keep printf messages small, use the debugger for viewing information (a buffer/stack overflow can occur that can cause errors)
- If your printf statements are overwriting each other, use a mutex

```
OS_MUT g_mut_uart; // global variable in helloworld.c
__task void task1() {
    os_mut_wait(g_mut_uart, 0xFFFF);
    printf("Test1: msg\n");
    os_mut_release(g_mut_uart);
}
__task init() {
    os_mut_init(&g_mut_uart);
```

Tip #3: Use Mutex to Signal Event

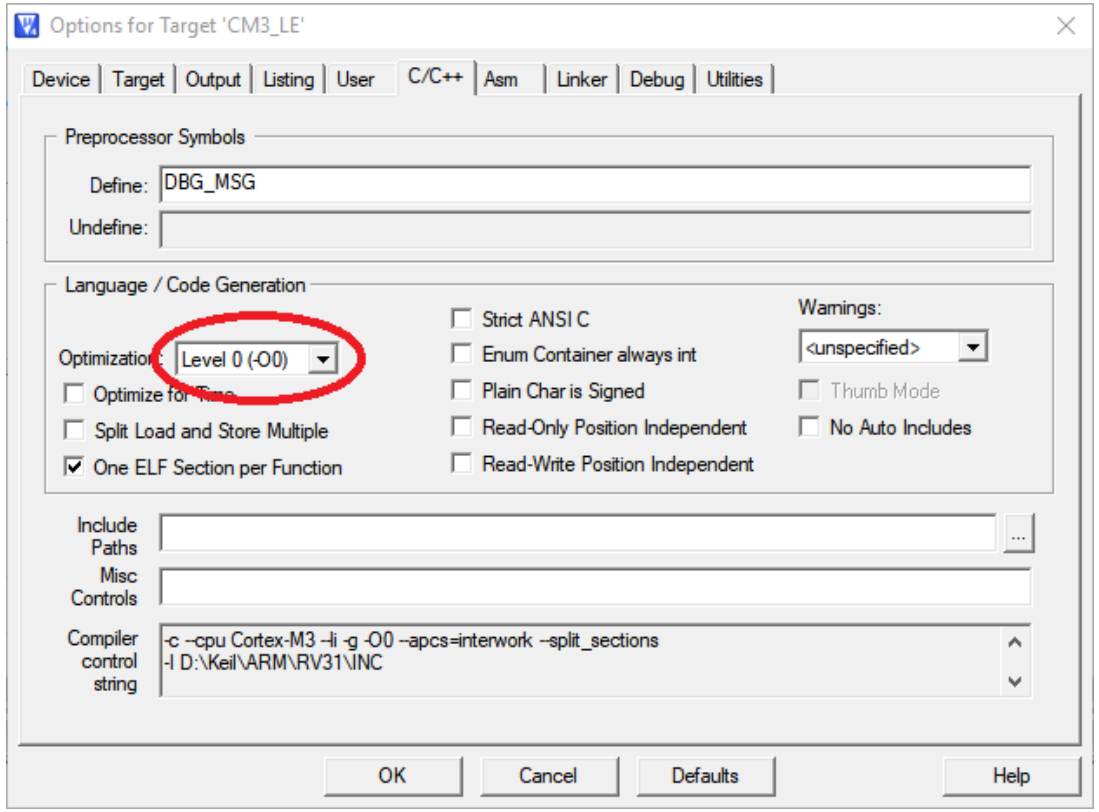
```
OS_MUT g_mut_evnt; // global variable

__task void task1() {
    os_mut_wait(g_mut_evnt, 0xFFFF);
    printf("Task1: Doing something – task2 must wait\n");
    os_dly_wait(3);
    os_mut_release(g_mut_evnt);
}

__task void task2() {
    printf("Task2: task1 should be doing something\n");
    os_dly_wait(2); //force task1 to grab mutex first
    os_mut_wait(g_mut_evnt, 0xFFFF);
    printf("Task2: task1 is done doing something\n");
    os_mut_release(g_mut_evnt);
}
__task init() {
    os_mut_init(&g_mut_evnt);
    os_tsk_create(task1, 1);
    os_tsk_create(task2, 1);
}
```

1. Task2: task1 should be doing something
2. Task1: Doing something – task2 must wait
3. Task2: task1 is done doing something

NOTE:
RL-RTX must have the wait/release for mutexes in the same task!
Cannot have a single wait in task1 and a single release in task2 (like previously shown).

Tip #4: Fixing <not in scope> Error

Name	Location/Value	Type
task1 : 2	0x00000718	Task
task2 : 3	0x00000724	Task
task3 : 4	0x0000076A	Task
rt_tsk_get	0x00001744	unsigned int f(unsign...
task_id	<not in scope>	param - unsigned int
p_task_...	<not in scope>	param - struct rl_task_...
size	<not in scope>	auto - unsigned int
stk_high	<not in scope>	auto - unsigned int
psp	<not in scope>	auto - unsigned int
p_tcb	<not in scope>	auto - struct OS_TCB *
SVC_Handl...	0x000001F6	void f()
os_idle_demo...	0x00000334	Task



Tip #5: Networking

- Post questions to the discussion form; save yourself time and effort by asking about problems you face

- Network with your classmates; you are allowed to discuss the lab with your classmates – you just need to have original work within your lab
  - Never let someone take a picture or copy your code!