ECE254 Lab 1

September 2018

Overview of Labs

Labs 1 and 2

- Based on the ARM RL-RTX OS
- Code runs on a small computer located in the desks of E2-2363
- Building the operating system from source
- Adding custom OS extensions

Labs 3 and 4

- Based on the Linux OS
- Code runs on the large ECE Linux servers, accessed via SSH
- Comparing inter-process communication methods
- Comparing Memory Management Algorithms

Methodology

Labs 1 and 2

- Small development boards
- Documentation is sparse
- Online help is limited
- Reading the source code is much more informative than the documentation

Labs 3 and 4

- Very popular operating systems
- Documentation is detailed
- Online help is available everywhere
- Reading the man pages is expected

Specifics about the Labs

- Labs 1 and 3 will introduce you to the different technologies associated with different operating systems design
- Lab 3 is a long lab that requires 2 separate projects to be completed, thus it has 2 lab sessions associated with it
- Lab 4 is difficult and this term has a very quick due date; prepare before hand
- Only 1 TA will be marking each lab, thus grades should be expected a couple weeks after the due date

Administrative Information

- Lab Room: E2-2363
- Access Code: 015908
- Lab Instructor: Jonathan Shahen (jmshahen@uwaterloo.ca)
- Lab 1 TA: Huanyi Chen (h365chen@uwaterloo.ca)

- **Lab 1 Due (Lab 203/206): Sept. 26th, 10:00 PM**
- **Lab 1 Due (Everyone Else): Oct. 3rd, 10:00 PM**

Github Repo

- https://github.com/jonathan-shahen/ECE254
- Contains the lab manual and starter code for each lab
- Lab manual has been update this term

Goals of Lab 1

1. Ability to create projects using uVision
2. Importing the RL-RTX library and building it from source
3. Running uVision projects in the Simulator and on the MCB1700 board
4. Modifying the RL-RTX operating system to add functionality

What is ARM RL-RTX and uVision 4?

- **ARM** – Small, portable, CPU manufacturer, and licenser for ARM instruction sets
- **RL** – Real-time Library
- **RTX** – Real Time Execution
- RL-RTX is an operating system that is made to run on an ARM CPU
- **KEIL uVision 4** is a Integrated Development Environment (IDE); which allows for programming, compiling, debugging, and loading the software onto the ARM powered board's storage
- uVision supports many CPUs and boards; thus you see it in industry

What is a Task?

- RL-RTX operating system is made to run multiple “tasks” at the same time
- A task is similar to a process on Windows/Linux
- A task points to code that needs to be run, the current state that task is in, the priority level, etc..
- The context switching of one task to another allows for a single-core CPU to run multiple tasks “at the same time”
- The operating system handles creating, destroying, and context switching tasks that are created by a programmer

User Mode vs. Kernel Mode

- CPUs have been developed with different “modes” they can be run in
- **Kernel mode (Ring 0):** this mode allows for access to all memory regions, all peripherals, etc. Ex: memory management, interprocess communication,
- **User mode:** memory regions are limited to only it’s space, and access to peripherals is limited. Ex: skype, windows explorer, uVision, gcc
- A user-mode application can call functions from programs that run in kernel mode, if that kernel program exposes that function. This is called a “system call”.

Simulator vs Hardware

Simulator

- Can use the debugger
- Slower than hardware
- Variable memory and stack size, but should be set to hardware
- Can be used anywhere uVision 4 is available/installed (can install on laptops)

Hardware

- Can use the debugger
- Faster than simulator
- Fixed limitations for memory and stack size
- Must be physically connected to the board in E2 2363 (can use laptops)

Deliverable #1: int **os_tsk_count_get** (void)

- **Kernel:** rt_tsk_count() in rt_Task_ext.c
- **Assumption:** A task is considered **ACTIVE** when its state is **not set to INACTIVE in the TCB**. The **os idle task is a valid task** that you should check the state as well.
- **Input:**
  - None
- **Return:**
  - The number of active tasks in the system
- **Question:**
  - How can we verify that the number we return is correct? What tests can I write in user-space to verify the number?

Deliverable #2:
OS_RESULT **os_tsk_get** (OS_TID task_id, RL_TASK_INFO *buf)

- **Kernel:** rt_tsk_get() in rt_Task_ext.c
- **Assumption:**
- **Input:**
  - Task_id – the ID of the task to lookup
  - Buf - a pointer to already allocated space that can hold the rl_task_info structure
- **Return:**
  - Returns OS_R_OK if the function succeeded, OS_R_NOK otherwise.
- **Question:**
  - What do you do if a invalid task_id is given? What about other invalid data?

Exposing Kernel Functions to User-Mode

- A user-mode application cannot call any kernel function, the kernel must be built to allow access to that specific function
- We have written the code to expose the 2 functions for you in RTL_ext.h

```
extern OS_RESULT rt_tsk_get (OS_TID task_id, RL_TASK_INFO *p_task_info);
#define os_tsk_get(task_id, p_task_info) _os_tsk_get((U32)rt_tsk_get, task_id,
    p_task_info)
extern OS_RESULT _os_tsk_get (U32 p, OS_TID task_id, RL_TASK_INFO *p_task_info) __SVC_0
```

1. Show that the function exists, it is just in a different file
2. Setup the function so that the programmer can call it without memorizing the request for kernel-mode
3. The function name here doesn't matter, we just give the address to the kernel function as well as the parameters and ask to run in __SVC_0 mode (see manual for more details about what this means)

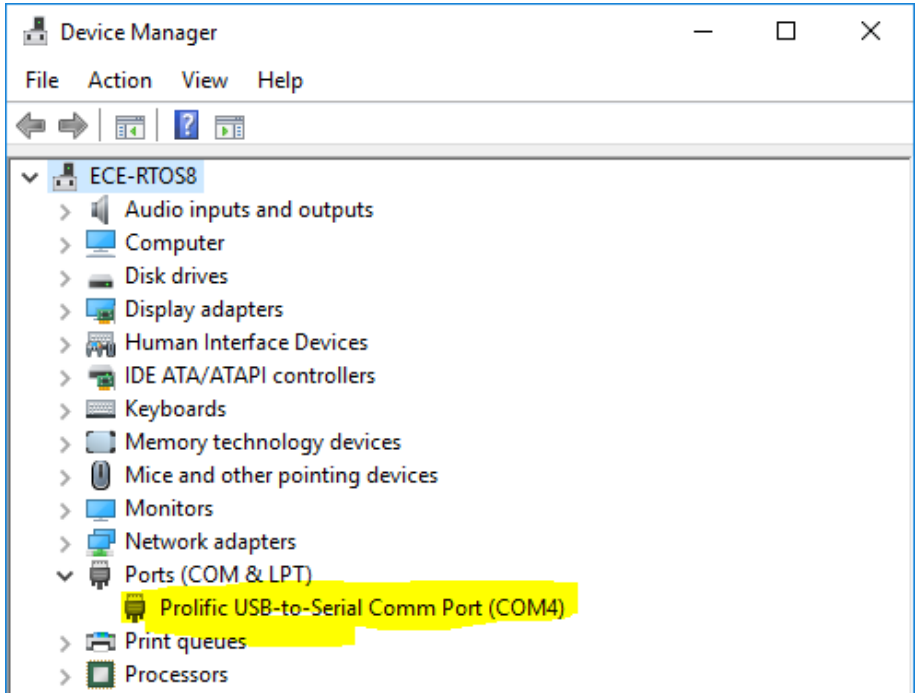
Read the Manual and the Source Code

- The manual is written for programmers working in user-mode
  - You will be writing a user-mode program to show that your kernel-mode functions actually work
- The source code will help for programming in kernel-mode
  - The assignment questions point to specific parts in the kernel source code that contain information relevant to your lab
  - Understanding how tasks are managed is crutial to working on Lab 2
  - A question about the RL_RTX OS will be on the midterm
  - Companies would rather have you pay them to extend an operating system than provide you with detailed knowledge of how it works

Tips #1

- You can use your own laptop if you want (follow installation steps in the lab manual). Make sure to install **version 4** (not 5) of uVision, and plug in both the black and blue USB cables.

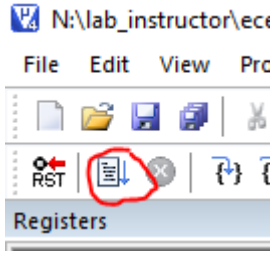
- Check which COM port is in use on the computer:



Tips #2

- Make sure that you BATCH BUILD when you make changes to the kernel

- When you debug, make sure you press RUN (F5)



- Debugging in the simulator outputs printf messages to the UART1 window (Peripherals -> UART -> UART1)

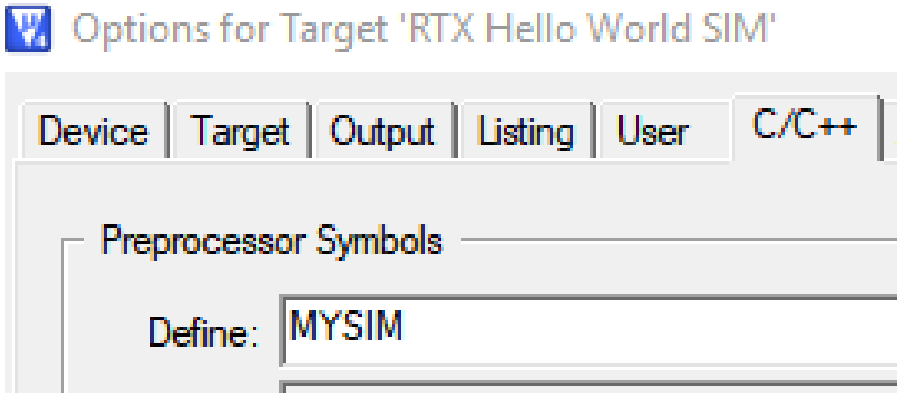
Tips #3

- Keep printf messages small, use the debugger for viewing information (a buffer/stack overflow can occur that can cause errors)

- Save your time by only BATCH BUILDING when you edit the kernel, also perform most of your testing in the simulator and occasionally test on the board to ensure that it ACTUALLY works

Tips #4

- Dynamically reduce `os_dly_wait()` times when running the simulator

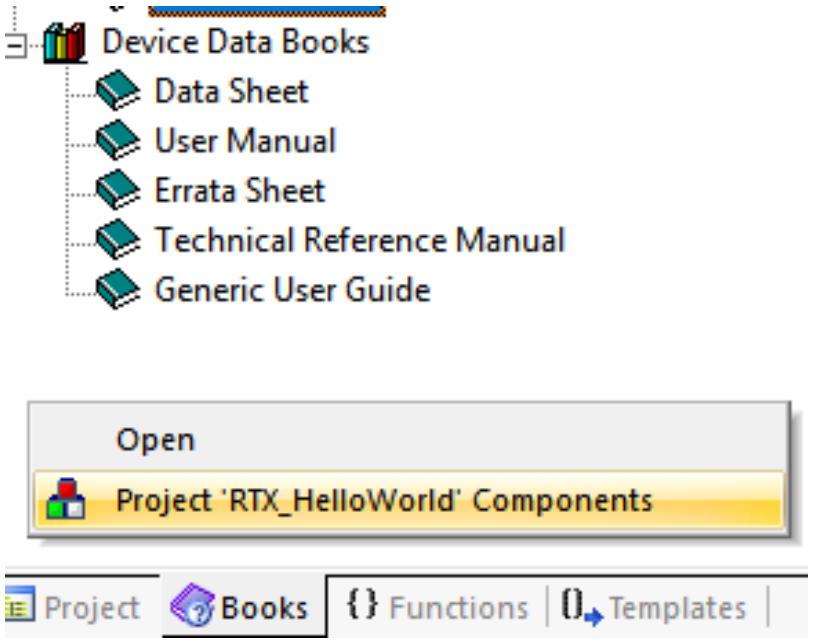


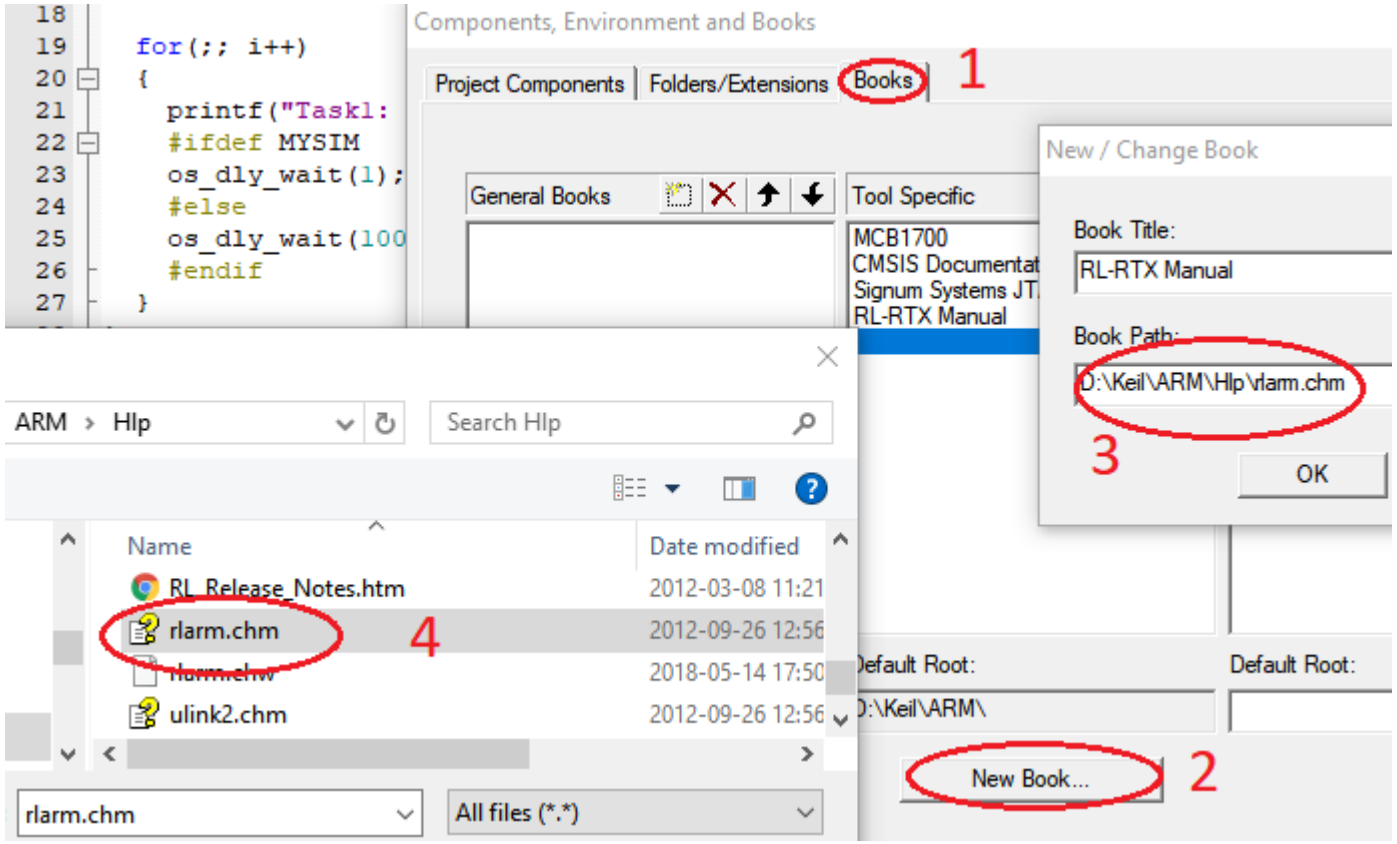

```
__task void task1()
{
  unsigned int i = 0;

  for(;; i++)
  {
    printf("Task1: %d\n", i);
    #ifdef MYSIM
    os_dly_wait(1);
    #else
    os_dly_wait(100);
    #endif
  }
}
```

Tips #5

- How to Add a Missing RTX Manual:





Tip #6: 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!