Real-Time Operating Systems

Hipólito Guzmán Miranda
Departamento de Ingeniería Electrónica
Universidad de Sevilla
hguzman@us.es

What is an Operating System?

According to Andrew S. Tanenbaum, it is difficult to give a clear definition because an OS performs multiple unrelated things. Let's try anyway:

- An OS is software that:
  - Extends the machine (abstracting us from the lowest level)
  - Manages resources (timers, memories, peripherals, …)
  - Controls the execution of programs (when tasks are being executed)

What is a Real-Time Operating System?

- An RTOS is an OS that does all that, but
  - Under critically defined time constraints

- What does this mean?
  - We can set time constraints for the tasks
  - Some tasks will be critical, others won't
  - The RTOS will make sure that the critical tasks get completed in time
  - If needed, it will kick non-critical tasks from the execution schedule (or move them around)

Surprise!
There is also an IEEE Standard

- *"IEEE Standard for a Real-Time Operating System (RTOS) for Small-Scale Embedded Systems,"* in IEEE Std 2050-2018, vol., no., pp.1-333, 24 Aug. 2018, doi: 10.1109/IEEESTD.2018.8445674

- Standardizes the kernel of the TRON project
  - Ken Sakamura, University of Tokyo / TRON Forum. 1984 – present

What does IEEE Std 2050-2018 (Standard for RTOS) say?

Standardizes an RTOS called µT-Kernel, which is divided in two layers:

- µT-Kernel/OS (Operating System)
  - Basic functions of RTOS: scheduling, synchronization, communication of tasks
- µT-Kernel/SM (System Manager)
  - Higher level management functions

What does IEEE Std 2050-2018 (Standard for RTOS) say?

- µT-Kernel/OS (Operating System)
  - Task management functions
  - Task synchronization functions
  - Task exception handling functions
  - Synchronization and communication functions
  - Extended synchronization and communication functions
  - Memory pool management functions
  - Time management functions
  - Interrupt management functions
- µT-Kernel/SM (System Manager)

What does IEEE Std 2050-2018 (Standard for RTOS) say?

- µT-Kernel/OS (Operating System)
- µT-Kernel/SM (System Manager)
  - Device management functions
  - Interrupt management functions
  - I/O port access support functions
  - Power management functions
  - System configuration information management functions
  - Memory cache control functions
  - Physical timer functions
  - Utility functions

What does IEEE Std 2050-2018 (Standard for RTOS) say?

- Task
  - The basic logical unit of concurrent program execution
- Scheduling
  - The processing that determines which task to execute next
- Dispatching
  - The switching of tasks executed by the processor
- Precedence
  - The order of task execution

IEEE Std 2050-2018
Task states

- RUNNING
- READY
- WAITING
- SUSPENDED
- WAITING-SUSPENDED
- DORMANT
- NON-EXISTENT

IEEE Std 2050-2018 Task states

- RUNNING
  - Currently being executed
- READY
  - Ready for running, but cannot run because a task with higher precedence is running
- WAITING
  - Execution is stopped because a system call was invoked that interrupts execution of the invoking task until some condition is met
- SUSPENDED
  - Execution was forcibly interrupted by another task

IEEE Std 2050-2018 Task states

- WAITING-SUSPENDED
  - Both WAITING and SUSPENDED at the same time. Happens when another tasks request the suspension of a WAITING task
- DORMANT
  - Not has been started, or has already completed execution
- NON-EXISTENT
  - Virtual state before task is created, or after it is deleted. Not actually registered in the system

The scheduler

- Is an important part of any OS
- Decides the order in which tasks are executed
- At the end of the day, a microprocessor core can just run a single task at a given time
- So, it is even more important in an RTOS
  - The scheduler is the only one who can make switch tasks between RUNNING and non-RUNNING states
- Preempt: stop a currently running task so a task with more priority can be run

What is FreeRTOS?

"FreeRTOS is a library that provides multi-tasking capabilities to what would otherwise be a single-threaded, bare-metal application."

So, it is a very *minimalist* OS

Perfect to start learning in practice!

FreeRTOS task states

- RUNNING
  - Currently being executed
- READY
  - Available to be selected by the scheduler to start running (not running, nor suspended, nor blocked)
- BLOCKED
  - Waiting for a specific event (either waiting for some time, or for a synchronization event coming from another task or an interrupt)
- SUSPENDED
  - Not available to the scheduler. Tasks are suspended and taken out of suspension using specific API functions

FreeRTOS: task priorities

- Tasks have priorities
- FreeRTOS ensures that the highest priority task that can be run is being run
- FreeRTOS defined a special task, the **Idle Task**, with priority 0
  - Only runs when there is nothing else to run
- It also defines an idle task hook
  - So you can do things when nothing needs to run
  - Such as low priority background tasks
  - Even better: measure spare processing capacity!

FreeRTOS uses Hungarian notation

Hungarian notation embeds some information in function and variable names

- v: returns void
- x: returns Basetype_t
- prv: private function
- pv: pointer to void
- pc: pointer to char
- us: unsigned short
- ux: unsigned Basetype
- px: pointer to Basetype

FreeRTOS: Creating tasks

Different API functions can be used:

- xTaskCreate()
  - Create a task
- xTaskCreateStatic()
  - Create a task, RAM is statically allocated
- xTaskCreateRestricted()
  - Create a Memory Protection Unit (MPU) restricted task
- xTaskCreateRestrictedStatic()
  - Create an MPU restricted task, RAM is statically allocated

xTaskCreate()

```
BaseType_t xTaskCreate( TaskFunction_t pvTaskCode,
                        const char * const pcName,
                        configSTACK_DEPTH_TYPE usStackDepth,
                        void * pvParameters,
                        UBaseType_t uxPriority,
                        TaskHandle_t * pxCreatedTask );
```

- pvTaskCode: name of C function (pointer)
- pcName: readable name, for debugging
- usStackDepth: depth of the stack, in *words*
- pvParameters: use this to pass arguments to the C function
- uxPriority: task priority
- pxCreatedTask: Pointer to a location to store a handle to the created task (so it can be modified or deleted)

xTaskDelete()

```
void vTaskDelete( TaskHandle_t xTask );
```

- xTask: pointer to the task to delete (the *pxCreateTask* returned by xTaskCreate())

Communicating tasks

Queues, semaphores, mutexes, stream buffers, message buffers, direct to-task notifications, …

- queues: send data between tasks
- semaphore: control access to a common resource
- mutex: mutual exclusion (binary semaphore)
- stream buffer: pass stream of bytes between tasks
- message buffer: very similar to stream buffers, but with messages of specific sizes
- direct to-task notifications: events sent directly to a task

Read the documentation for details

Queues

```
QueueHandle_t xQueueCreate( UBaseType_t uxQueueLength,
                            UBaseType_t uxItemSize );
```

```
BaseType_t xQueueSend(
                        QueueHandle_t xQueue,
                        const void * pvItemToQueue,
                        TickType_t xTicksToWait
                      );

BaseType_t xQueueReceive(
                           QueueHandle_t xQueue,
                           void *pvBuffer,
                           TickType_t xTicksToWait
);
```

- xTicksToWait: Maximum time (in ticks) to wait before giving up if the queue is full/empty

There are many RTOSes

- FreeRTOS
- TRON (The Real-Time Operating Nucleus)
- Real-Time Linux
- VxWorks
- Zephyr
- QNX
- …

Wikipedia category and comparison

Conclusions

- An RTOS lets you assign priorities and times/deadlines to software tasks
  - Higher priority tasks may displace lower priority tasks
  - Of course the microprocessor will eventually reach a limit
  - It's important to know how to measure if we are nearing this limit!
- When to use an RTOS?
  - When you want/need this time reliability/predictability in software

Conclusions

- Why use it if we have the FPGA fabric?
  - It allows us to (re)use software in a greater range of situations
  - We may not achieve reliable real-time with a normal OS (not that we should risk it anyway), but may be able to achieve it using an RTOS
  - Less effort than writing RTL for sequential tasks
  - Software reuse on time-critical tasks
  - Software task management in critical systems
- Consider RTOSes as part of our toolbox
  - Learning curve
  - Decide when to use them

Bibliography

- *"IEEE Standard for a Real-Time Operating System (RTOS) for Small-Scale Embedded Systems,"* in IEEE Std 2050-2018, vol., no., pp.1-333, 24 Aug. 2018, doi: 10.1109/IEEESTD.2018.8445674
- Richard Barry, *"Mastering the FreeRTOS Real Time Kernel: A Hands-On Tutorial Guide"*
- FreeRTOS team: "*FreeRTOS documentation*", including the API References

Autonomous work

- Read the links in the “Communicating Tasks” slide of this slideshow
- Read chapter 4, “Task Management”, of Richard Barry, *“Mastering the FreeRTOS Real Time Kernel: A Hands-On Tutorial Guide”*