MBED OS - Fundamentals

- **mbed** is a shared-SW platform for the development of embedded systems and IoT based on ARM devices.

- **mbed OS** is an open-source OS for **mbed**.
  - It allows a high abstraction degree of the HW details
  - It is specifically thought for Cortex-M processors of different manufacturers

- **Advantages:**
  - Wide and open development community
  - Fast prototyping
  - Easy code reusability

  Find help, share knowledge, reuse code with the community!

MBED RTOS - Fundamentals

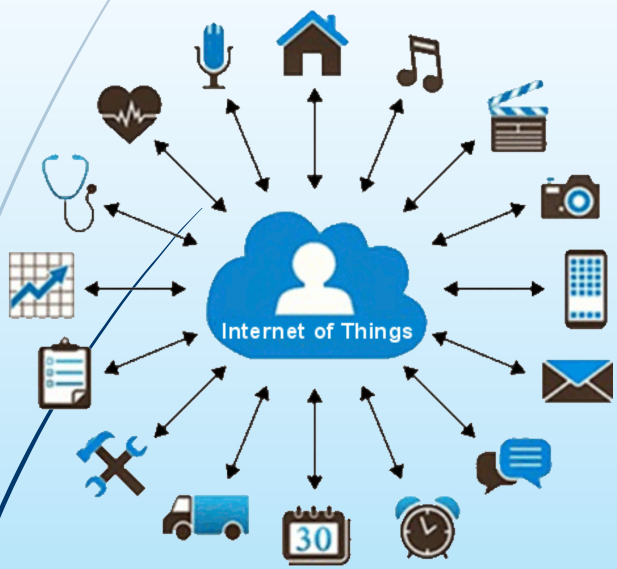
- **mbed RTOS**
  - Native threads support
  - It is a wrapper for C++ over CMSIS-RTOS RTX
- **CMSIS-RTOS RTX**
  - C/C++ API over the **Keil RTX kernel:**
- **CMSIS-RTOS**
  - Is the standard API for RTOS
  - Is able to support different kernels

- *Tiny kernel*
- Optimized for memory-limited devices
- Multitasking & concurrent threads
- Scheduling of the µP resources usage

RTOS Documentation

- **mbed OS 6**
  - https://os.mbed.com/
- **Real Time Kernel: Keil RTX**
  - http://www.keil.com/pack/doc/CMSIS/RTOS/html/rtxImplementation.html
- **mbed RTOS API**
  - https://os.mbed.com/docs/mbed-os/v6.15/apis/index.html

EPC for IoT - MBED OS



- MBED OS
  - Basic memory model

MBED OS - Memory

Basic memory model

RAM memory	
Scheduler/ISR stack	Last address of RAM
Heap cont.	
User thread n+1 stack	
User thread n stack	
Heap	
Global data	
Other stacks…	First address of RAM

- Static memory: allocated at compile time, no resize during runtime. Contains stacks for default threads
- Dynamic memory: Heap & stacks for user threads

MBED OS - Memory

Basic memory model

Flash memory	
Application	Last address of flash
Optional bootloader	
Vector table	First address of flash

More info: How much memory do I need for my arm Cortex-M applications?
https://community.arm.com/processors/b/blog/posts/how-much-stack-memory-do-i-need-for-my-arm-cortex--m-applications

Heap vs. Stack

Stack	Heap
Memory storing variables created by each function and managed by the CPU	*Memory not 'automatically' managed by the CPU*
int, char, typedef…	malloc(), calloc()…
LIFO type	Needs free() to deallocate! ⚠
Very fast access	*Slower* access
No resizing	Resizing allowed with realloc()
Only local variables	Variables are accessible by any function

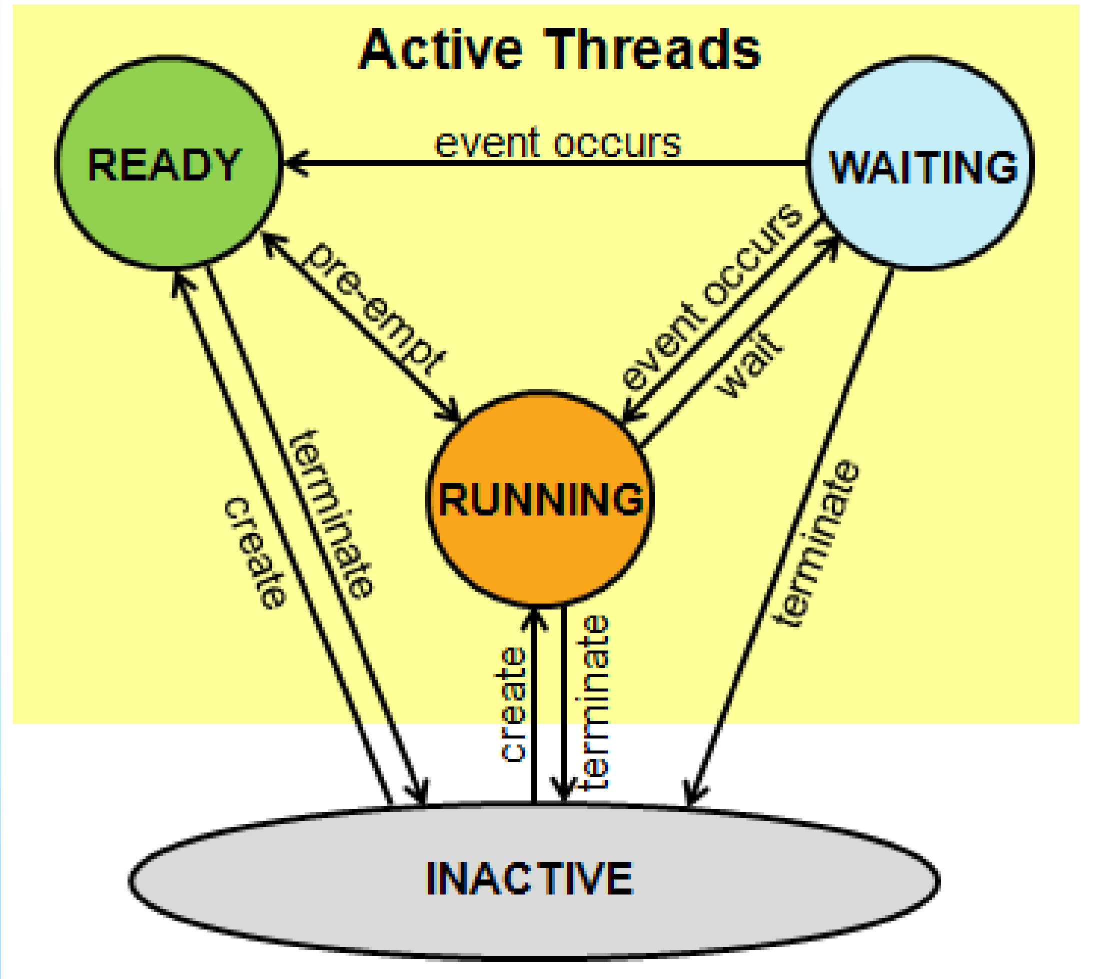
- Configurable in the startup file (*.s)
- It could be very useful to check out the *.map file

Extra tip: How much memory do I need for my arm Cortex-M applications?
https://community.arm.com/processors/b/blog/posts/how-much-stack-memory-do-i-need-for-my-arm-cortex--m-applications

RTOS - Tasks

Tasks may be in… state.

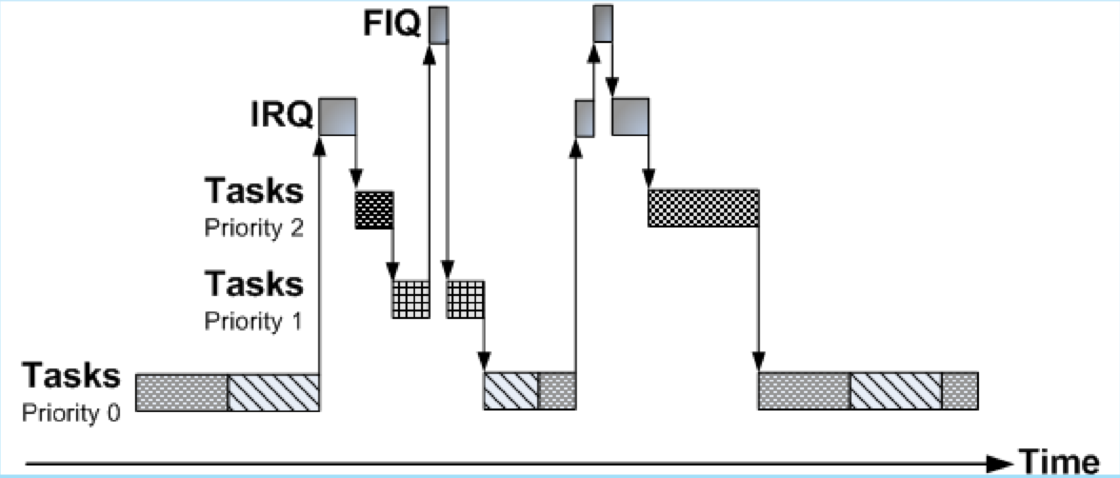
- **Running**: thread in execution
- **Ready:** thread is ready to be executed.
- **Waiting:** threads waiting for an event.
- **Inactive:** no created or no finished threads. No use of resources.



Source: ARM mbed documentation. https://docs.mbed.com

Scheduler

- Manages the execution of threads giving slots of processing time
- Periodical interrupts of one timer define the processing times
- Context changes:
  - Change in the running task
  - Independent stacks for each thread
  - Configurable task priority



RTOS - Scheduler - Main concepts

- **Timeslice:** slot of execution time given to each thread
  - Multiple number of SysTick Timer ticks

- Scheduling strategies:
  - Pre-emptive
  - Round-Robin
  - Round-Robin Pre-emptive
  - Cooperative Multitasking

To change the configuration of the system take a look at: RTX_Conf_CM.c

RTOS - Scheduling strategies

Pre-emptive

- Each thread has its own priority
- Higher priority threads thrown up lower priority threads

Round-Robin

- All threads have the same priority level
- All threads will be sequentially executed

To change the configuration of the system take a look at: RTX_Conf_CM.c

RTOS - Scheduling strategies

Round-Robin Pre-emptive

- Each thread has its own priority
- Threads with same priorities are executed in RR way while no higher priority threads are READY.
- **Important:** a bad priority configuration may cause a hang up.

Cooperative Multitasking

- All threads have the same priority level but no RR.
- First thread takes the CPU whilst no WAITs, then next thread in READY state runs.

To change the configuration of the system take a look at: RTX_Conf_CM.c

RTOS - Interrupts

- As it can be seen interrupts are managed in a different fashion comparing with a bare-metal… But! it is always important to keep **ISRs** as reduced as possible, even when having an RTOS over the HW.

RTOS - Threads

- https://os.mbed.com/docs/mbed-os/v6.15/apis/thread.html
- **Thread class**

Public Member Functions
Thread (osPriority priority=osPriorityNormal, uint32_t stack_size=OS_STACK_SIZE, unsigned char *stack_mem=NULL, const char *name=NULL)
thread.start(task_function);
thread.start(callback(task_function, params));

```
/// Priority values.
typedef enum {
  osPriorityNone          =  0,         ///< No priority (not initialized).
  osPriorityIdle          =  1,         ///< Reserved for Idle thread.
  osPriorityLow           =  8,         ///< Priority: low
  osPriorityBelowNormal   = 16,         ///< Priority: below normal
  osPriorityNormal        = 24,         ///< Priority: normal
  osPriorityAboveNormal   = 32,         ///< Priority: above normal
  osPriorityHigh          = 40,         ///< Priority: high
  osPriorityRealtime      = 48,         ///< Priority: realtime
  osPriorityISR           = 56,         ///< Reserved for ISR deferred thread.
  osPriorityError         = -1,         ///< System cannot determine priority or illegal priority.
} osPriority_t;
```

RTOS - Data-flow and shared resources

- **Global variables:** *maybe* an easier solution, but not worthy for complex applications…
- **Solution:**
  - Asynchronous data-exchange between threads.
- *“An application is a set of threads + dataflow between them”*
- Let’s define shared buffers… let’s synchronize them ☹
- Let’s use:
  - Mutex
  - Semaphores
  - Message queues
  - … others

Some of them may disable idle or low-consumption modes…