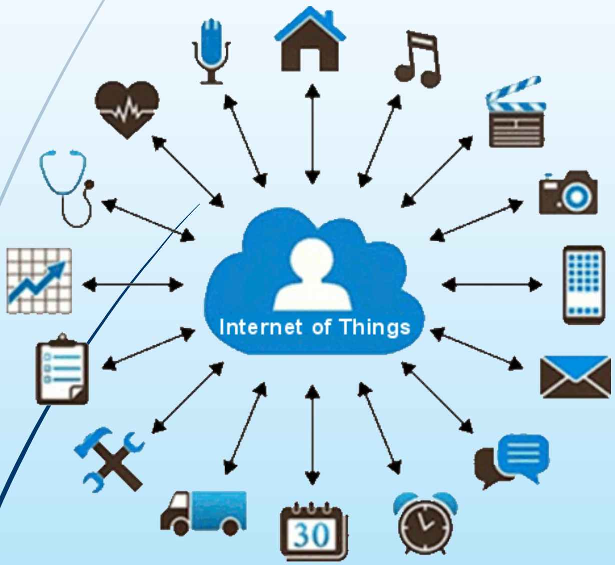
EPC for IoT - MBED OS



- MBED OS
    - Fundamentals

MBED OS - Fundamentals

- When programming for Embedded Systems we have two options:
- **Bare-metal programming**
  - No OS slice between applications and HW
  - The programmer must schedule and manage the tasks’ execution
  - As well as using HW resources
  - A while(1) loop will execute basic and simple functions
  - But… you are able to use ISRs to control events
  - Main advantage: reduced memory requirements

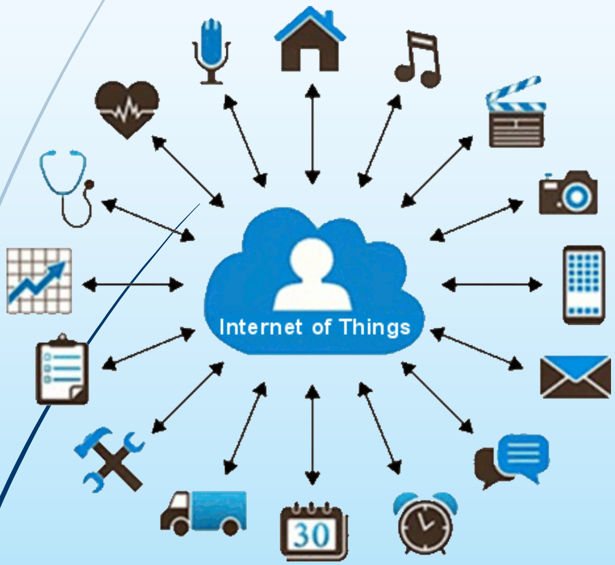
MBED OS - Fundamentals

- When programming for Embedded Systems we have two options:
- **Embedded Operating System / Real-Time Operating System (RTOS)**
  - It supports multitasking execution
  - Scheduling at runtime is supported by kernel
  - Threads can… be executed by priority, be *thrown up* from CPU…
  - The kernel is ‘controlled’ by the ISRs
    - Less overload than polling
    - The ISRs can activate threads
    - REAL-TIME !

MBED OS - Fundamentals

- When programming for Embedded Systems we have two options:
- **Embedded Operating System / Real-Time Operating System (RTOS) –** disadvantages**…**
  - But… It will require more memory ☹
  - *Be careful when designing and/or programming your apps.*

EPC for IoT - MBED OS



- MBED OS
  - Threads

RTOS - Thread

Thread: basic execution unit

Function	Thread

unsigned int function (void) {
  //actions
  return (output);
}
	void thread_x (void) {
	  while(1){
	  //actions
	  } }

- Object-oriented design
  - Independent coding and verification
  - Eases the debugging process
  - Eases code reusability
- Own stack per thread
- Main function: special thread which creates the rest of threads.

RTOS - Threads

- https://os.mbed.com/docs/mbed-os/v6.15/apis/thread.html
- **Example with threads**

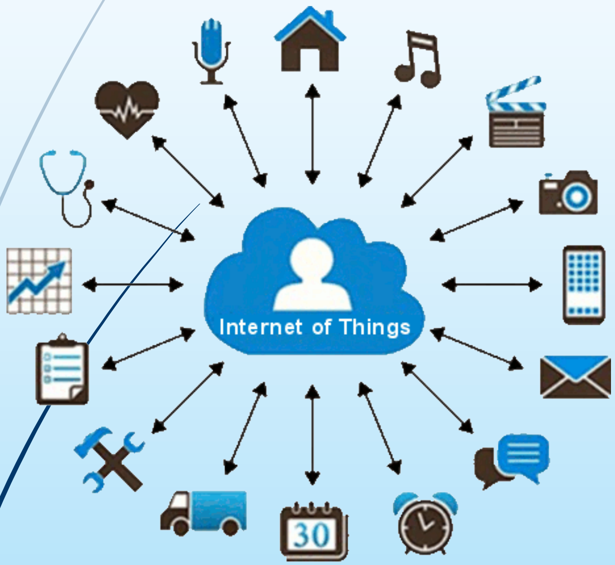
```
#include "mbed.h"
DigitalOut     led1(LED1);
DigitalOut     led2(LED2);
Thread         thread;

void led2_thread(void) {
  while(true) {
    led2 = !led2;
    wait(1);
  } }

int main(void) {
  thread.start(led2_thread);

 while(true){
    led1 = !led1;
    wait(0,5);
  } }
```

EPC for IoT - MBED OS



- MBED OS
  - Threads & Interrupts

RTOS - Synchronization with threads

Using signal events to synchronize threads

- Allow us to put threads in WAIT state and (re)activate them from other threads
- When the signal is received the thread changes to READY state.
- Signals (flags)
- There is a Timeout, when passed thread changes automatically to READY

RTOS - Interrupts

- The **ISRs** are the functions executed when an **IRQ** happens
- The **IRQs** have high priority in order to reduce the response latency
- Some limitations with the RTOS API for ISRs:
  - The code should not be into the ISR
  - Objective: reduce scheduling delays
- Solution:
  - Put the desired code into a thread and use synchronization tools
  - https://os.mbed.com/teams/mbed_example/code/rtos_signals/file/476186ff82cf/main.cpp/

```
ISR_callback
{
   …
   thread_id.signal_set(SIGNAL)
   …
}
```

```
Thread
{
   …
   Thread::signal_wait(SIGNAL);
   //Code
}
```