Real Time Operating Systems

LY ETRX
Sem VIII

Arati Phadke
AY 2021-22

SOMAIYA
VIDYAVIHAR UNIVERSITY
K J Somaiya College of Engineering

Computer Organization and Architecture

Course Outcomes

At the end of successful completion of the course the student will be able to

CO1: Understand real time operating system concepts.

CO2: Create various instances of a task and send and receive data to and from a queue

CO3: Implement the APIs within an Interrupt Service Routine

CO4: Develop algorithms for real time processes using FreeRTOS

CO5: Understand memory allocation schemes

Module 1

Operating System Concepts

1.1 Basics of OS: Real time concepts, hard real time and soft real time, difference between general purpose operating system and real time OS, components of an OS, kernel, tasks and threads, Free RTOS

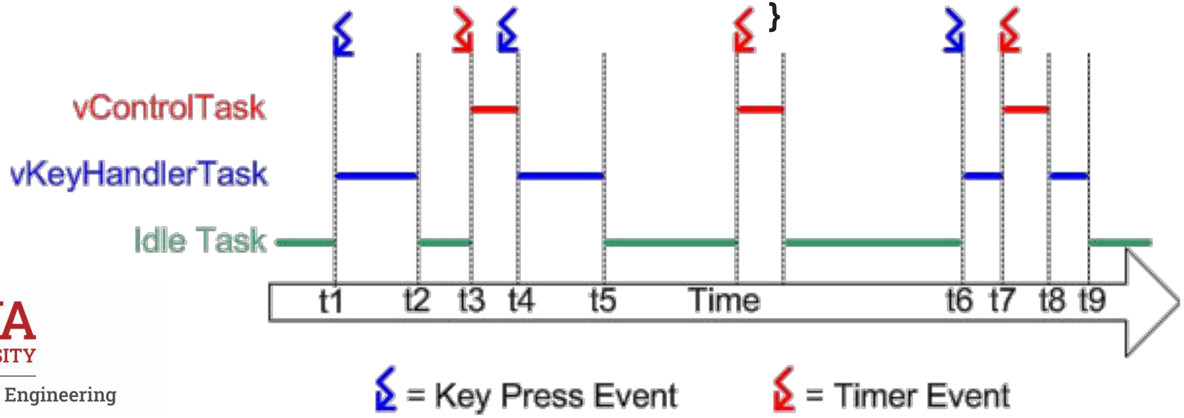
Real Time Systems

Concept of Keypad / Display interface

A user must get visual feedback of each key press within a **reasonable** period

```
void vKeyHandlerTask( void *pvParameters )
{    for( ;; )
   {  [Suspend waiting for a key press]
      [Process the key press]
   }
}
```

```
void vControlTask( void *pvParameters )
{
    for( ;; )
    {  [Suspend waiting for 2ms since the start of the previous cycle]
         [Sample the input]
         [Filter the sampled input]
         [Perform control algorithm]
         [Output result]
    }
}
```



Some Basic Concepts

- Foreground and background systems
- Tasks and Threads
- Context Switching
- Critical section
- Shared Resources,

Real Time Operating System

Ref: Real-time-operating-systems-foundations : Jim Cooling : Chapter 10

Choice of RTOS

For small, cost-sensitive units an RTOS must also be small. Need of limited functionality; so small OS should be sufficient.
Applying the same RTOS to a complex control system: Will it deliver the required performance ?
To help choose the best solution to our multitasking requirements we need to define the:

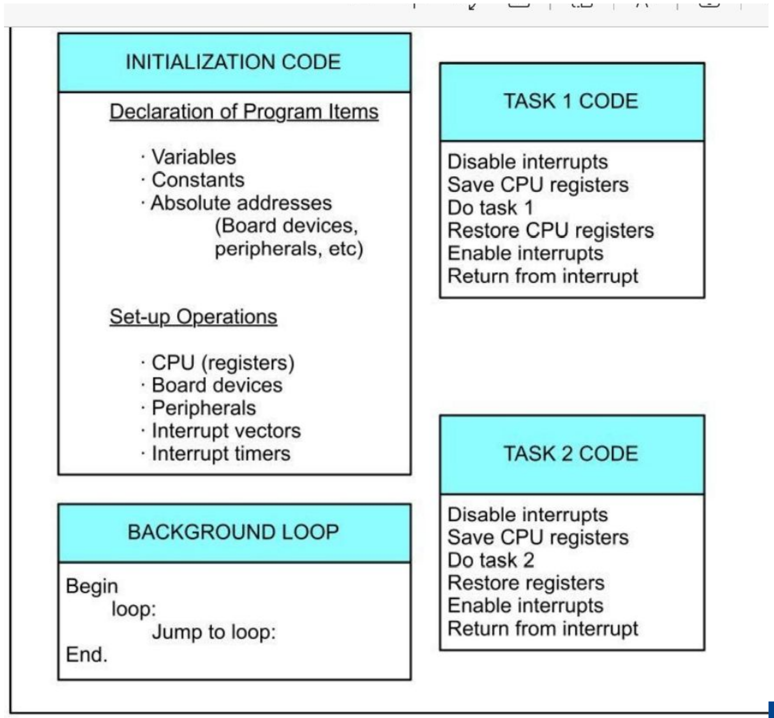
- Core functionality provided by an RTOS.
- Additional functionality needed to support more complex architectures.
- Component parts of both core and extended RTOS's.
- Structuring and interconnection of the various RTOS components.

Real Time Operating System

Ref: Real-time-operating-systems-foundations : Jim Cooling : Chapter 10

Simple multitasking via interrupts.

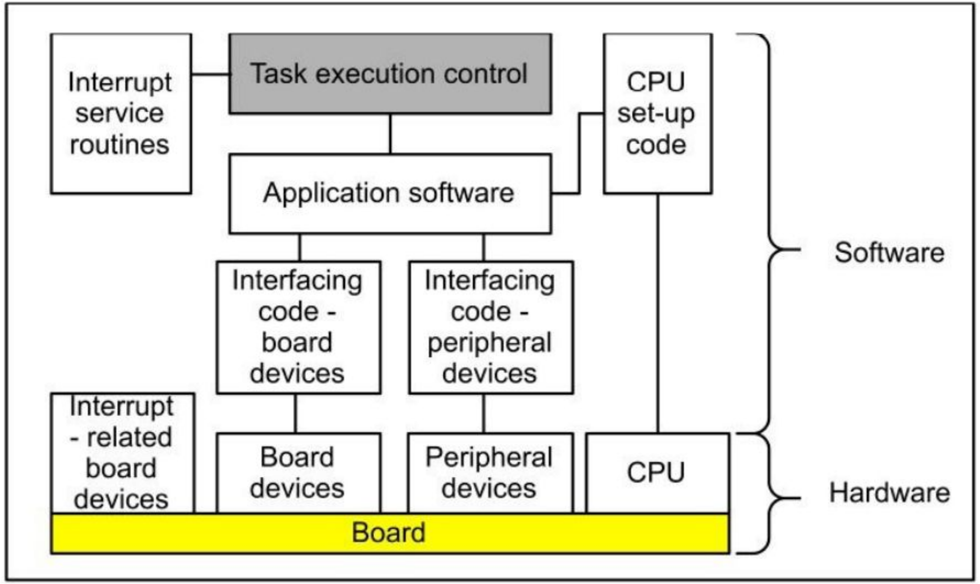
- Initialization Code
    - Declarations
    - Setup operations
- Background Processing
- Application Tasks



Real Time Operating System

Ref: Real-time-operating-systems-foundations : Jim Cooling : Chapter 10

- **Control of Hardware by Software Units**
  - ISRs
  - Interfacing code, board devices
  - Interfacing code, peripheral devices
  - CPU set-up code
- Application software interacts with the hardware via the interfacing code layer
- Task activation is carried out by ISRs, these being triggered by hardware generated signals.



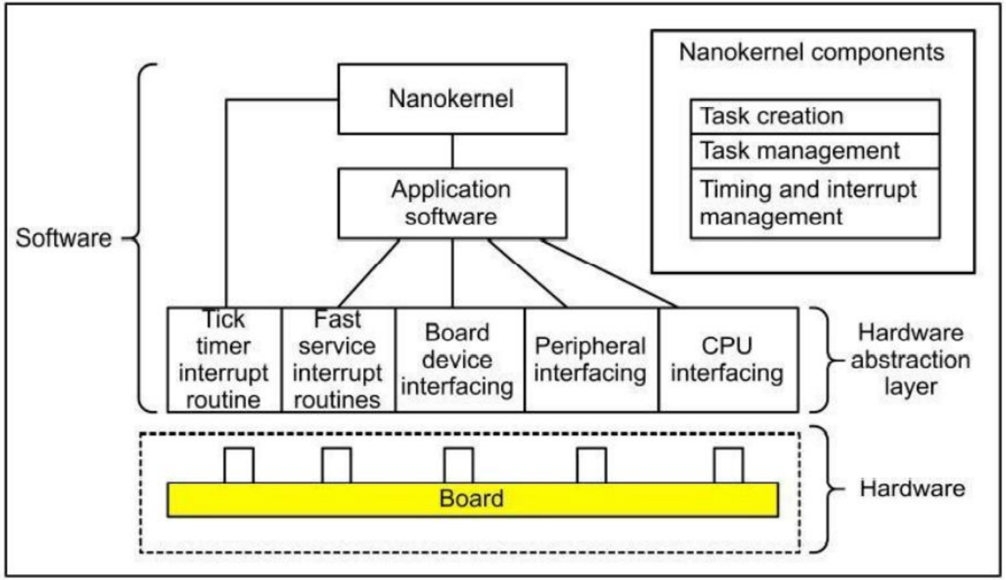
Real Time Operating System: Nanokernel

Ref: Real-time-operating-systems-foundations : Jim Cooling : Chapter 10

A nanokernel is a small kernel that offers hardware abstraction, but without system services.

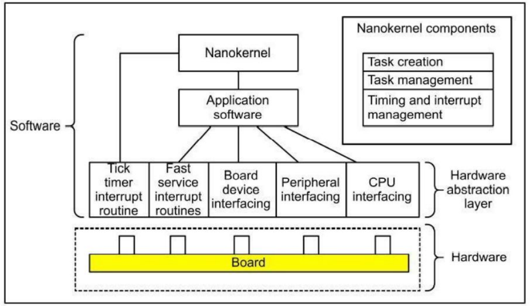
Minimal set of OS services:

- Task creation.
- Task management - scheduling and dispatching.
- Timing and interrupt management.



Real Time Operating System: Nanokernel

Ref: Real-time-operating-systems-foundations : Jim Cooling : Chapter 10



- All application tasks - except time-critical ones - are controlled by the kernel. Critical tasks activated by the fast service interrupt routines which invoke immediate task dispatch (thus by-passing the kernel). A separate interrupt routine -the tick timer, basic timing mechanism of the nanokernel.
- Hardware Abstraction Layer, HAL.
- an abstract interface to the application software and the kernel itself.
- Should encapsulate all code needed for the set-up and operation of the board hardware.
- Depends on the nature of the computer hardware.
- Is usually produced by the programmer as a special-purpose design.

Real Time Operating System: Nanokernel

Ref: Real-time-operating-systems-foundations : Jim Cooling : Chapter 10

- Initialize the OS.
- Create a Task.
- Delete a Task.
- Delay a Task.
- Control the Tick (start time-slicing, set the time slice duration).
- Start the OS.
- Set the Clock Time.
- Get the Clock Time.

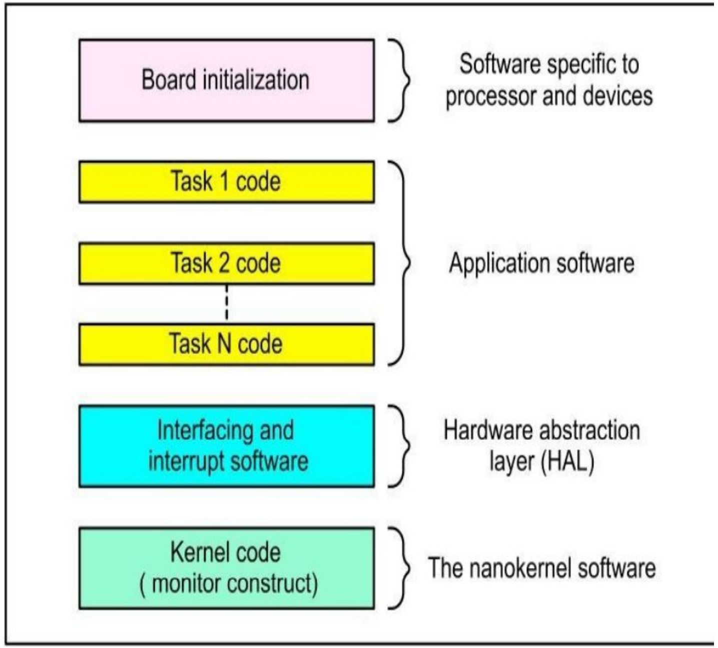


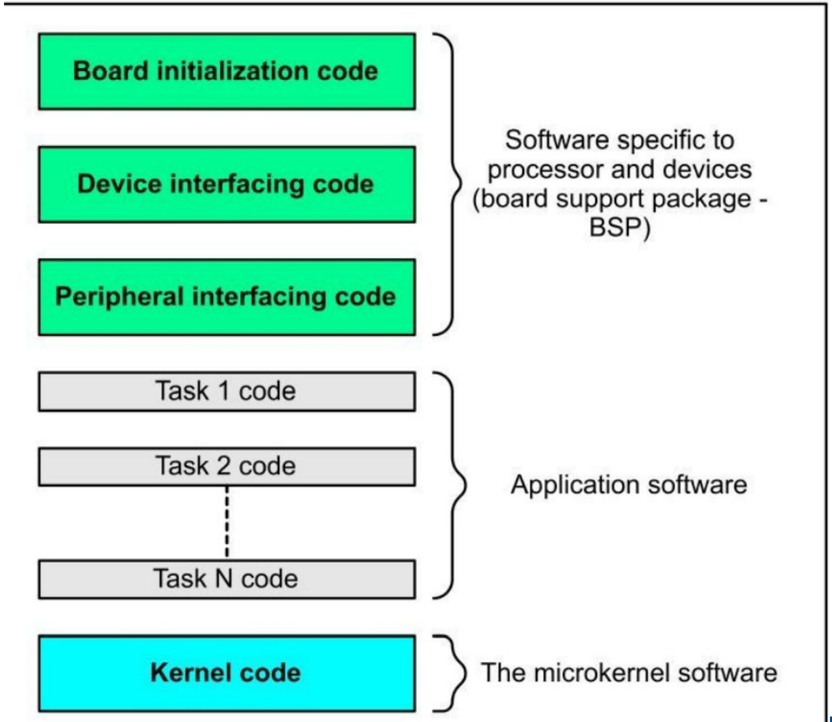
Figure 10.5 Code organization - nanokernel-based system

Real Time Operating System: Microkernel

Ref: Real-time-operating-systems-foundations : Jim Cooling : Chapter 10

Significant changes from nanokernel

- Increased kernel functionality
- Use of a board support package (BSP)



Real Time Operating System: Microkernel

Ref: Real-time-operating-systems-foundations : Jim Cooling : Chapter 10

- BSP, -part of the microkernel package to support both custom and standard hardware designs. , to minimize the efforts involved in developing interfacing software for new designs.
- Board initialization, device interfacing and peripheral interfacing code are all contained in the BSP
- Generally following facilities are found in many packages:
  - Board-specific functions, including general initialization, RTOS initialization and interrupt configuration.
  - Device-specific driver software, supplied in template form. This is board independent and therefore needs configuration by the programmer.
  - Detailed low-level code used by the device drivers, but applicable to specific devices (e.g. Intel 82527 communications controller).
  - Support for the development of special-purpose BSP functions.

Microkernel Features

Ref: Real-time-operating-systems-foundations : Jim Cooling : Chapter 10

- System set-up and special functions.
  - Initialise the OS (if not part of the BSP).
  - Set up all special (non-kernel) interrupt functions.
  - Start execution of the application programs.
- Process (task) scheduling and control.
  - Declare a task. ,Start a task. Stop a task. Destroy a task.
  - Set task priorities.
  - Lock-out task (make it non preemptive). , Unlock a task.
  - Delay a task. , Resume a task.
  - Control the real-time clock (tick, relative time and absolute time functions).
  - Control use of interrupts.
- Mutual exclusion.
  - Gain control using semaphores (entry to critical region). Release control using semaphores (exit from critical region). Gain control using monitors. Release control using monitors. Wait in a monitor.

Real Time Operating System: Microkernel

Ref: Real-time-operating-systems-foundations : Jim Cooling : Chapter 10

- Synchronization functions -
  - no data transfer. Initialise a signal/flag. Send a signal/flag (with and without timeouts). Wait for a signal/flag (with and without timeouts). Check for a signal/flag.
- Data transfer without synchronization.
  - Initialise a channel/pool. Send to a channel/write to a pool (with and without timeouts). Receive from a channel/read from a pool (with and without timeouts). Check channel state (full/empty).
- Synchronization with data transfer.
  - Set up a mailbox. Post to a mailbox (with and without timeouts). Pend on a mailbox (with and without timeouts). Check on a mailbox.
- Dynamic memory allocation.
  - Allocate a block of memory. Deallocate a block of memory.

Real Time Operating System: Microkernel

Ref: Real-time-operating-systems-foundations : Jim Cooling : Chapter 10

-

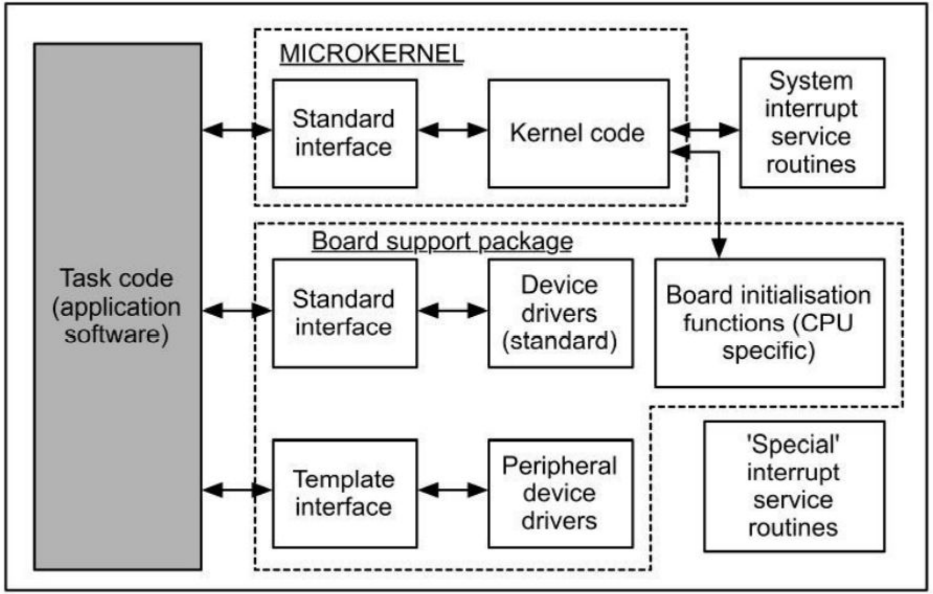


Figure 10.7 Software conceptual model - small microkernel-based system

Real Time Operating System: Microkernel

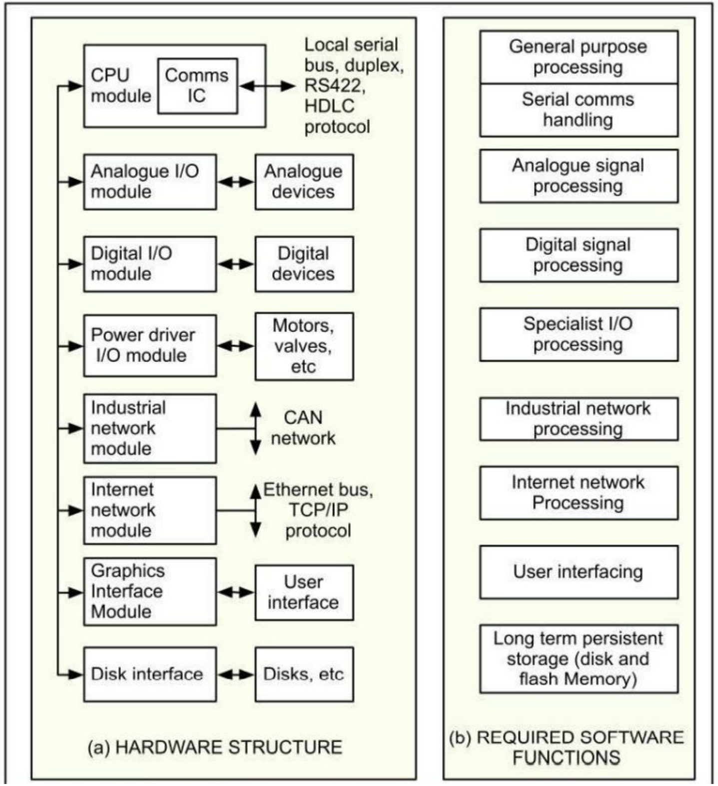
Ref: Real-time-operating-systems-foundations : Jim Cooling : Chapter 10

- Self-host approach :Real-time application is developed on the full-fledged operating system, and once the application runs satisfactorily it is fused on the target board on a ROM or flash memory along with a stripped down version of the same operating system.
- Self Host Op Sys use Microkernel architecture
- core functionalities such as interrupt handling and process management are implemented as kernel routines.
- other functionalities such as memory management, file management, device management, etc are implemented as add-on modules which operate in user mode.
- easy to configure , micro kernel is lean and efficient.
- drivers, file systems, and protocol stacks , all kernel processes run in separate memory-protected address spaces. So, system crashes on this count are very rare, and microkernel-based operating systems are very **reliable.**

A general-purpose embedded RTOS.

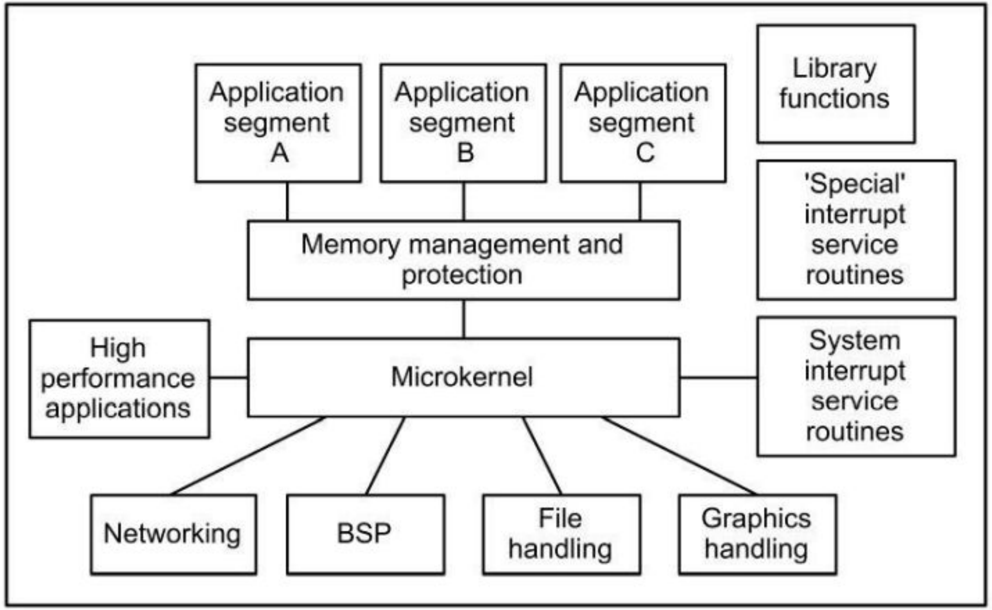
Ref: Real-time-operating-systems-foundations : Jim Cooling : Chapter 10

-



A general-purpose embedded RTOS.

Ref: Real-time-operating-systems-foundations : Jim Cooling : Chapter 10

- 

Introduction to FreeRTOS.

- FreeRTOS is an open-source freely available real-time operating system.
- There are two approaches to multitasking, regular tasks and light-weight co-routines:

1. tasks that have four states: READY, RUNNING, BLOCKED and SUPSENDED, while
2. co-routines have only the first three states.

- Co-routines have lower priority than any task.
- All co-routines share the same stack—consequently, if a co-routine is blocked, only static local variables maintain their values.

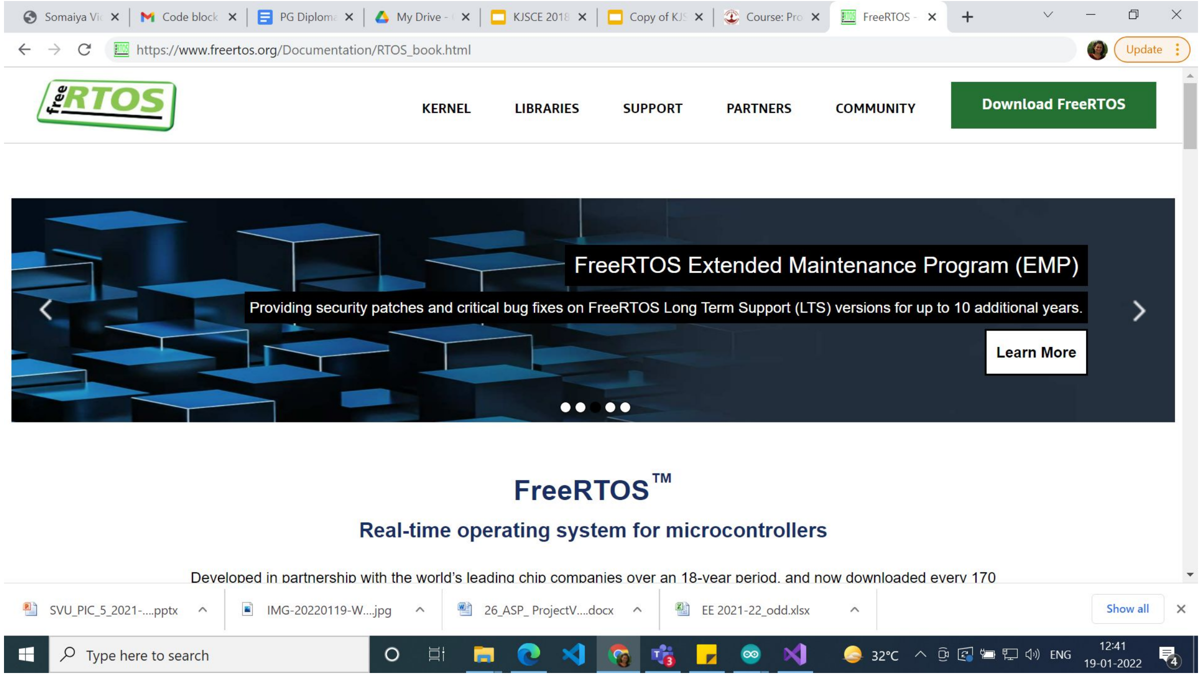
Introduction to FreeRTOS.

- For inter-process communication, there are five data structures:
  - queues for message passing,
  - binary semaphores,
  - counting semaphores,
  - mutexes (a binary semaphore where the task holding the mutex must be the one to post to it), and
  - recursive mutexes.
- The latter is a mutex where the same task can recursively wait on it, incrementing a counter. That task must then issue a post for each wait.
- Five dynamic memory allocation implementations that come with FreeRTOS.
- FreeRTOS has a site / book describing Running the RTOS on an ARM Cortex-M Core.

Introduction to FreeRTOS.

- For inter-process communication, there are five data structures:
  - queues for message passing,
  - binary semaphores,
  - counting semaphores,
  - mutexes (a binary semaphore where the task holding the mutex must be the one to post to it), and
  - recursive mutexes.
- The latter is a mutex where the same task can recursively wait on it, incrementing a counter. That task must then issue a post for each wait.
- Five dynamic memory allocation implementations that come with FreeRTOS.
- FreeRTOS has a site / book describing Running the RTOS : https://www.freertos.org/

Introduction to FreeRTOS.



Introduction to FreeRTOS.

- Downloading FreeRTOS
- Using with some IDE
- Working on Microcontroller
- Examples
- Creating and deleting a task
  - On Visual Studio
  - On Arduino Mega