Mechatronics Engineering

Multitasking

Prof. Ayman A. El-Badawy
Department of Mechatronics Engineering
Faculty of Engineering and Materials Science
German University in Cairo

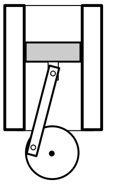
Introduction to Multitasking

- When multiple operations must be performed on a microcontroller at widely varying times, writing a single large program can become too complex and disorganized.

- Example: Automotive Engine Control Unit

  ❖ The engine requires performing different tasks with different rates (multirate):

    A. Firing the spark plug (higher rate >100 Hz).
    B. Reading the crankshaft position sensor.
    C. Controlling the air/fuel mixture.
    D. Reading the oxygen sensor for emission control.
    E. Reading gas pedal position (lower rate).



Introduction to Multitasking

- A common programming method is the cyclic executive.
  - ❖ A single loop includes all tasks and is executed periodically.
  - ❖ Example of cyclic executive code:

```
int main(void)
{
    /* Initialization code here*/
    for(;;) // Infinite main loop
    {
        /* Read Inputs */

        /* Execute periodic tasks here */

        /* Set Outputs*/

        /* Sleep until the start of the next period (E.g., wait
for timer compare match) */
    }
}
```

GUC German University in Cairo
Prof. Ayman A. El-Badawy
Department of Mechatronics Engineering
Faculty of Engineering and Material Science

Introduction to Multitasking

- Main strength of the cyclic executive:
  - ❖ Simplicity
  - ❖ Requires little memory and processing overhead to schedule the tasks.
- Main Drawbacks of the cyclic executive method:
  - ❖ Difficult to develop complex systems, since all the tasks' periods must be a multiple (nT) of the main loop's period (T).
  - ❖ Scheduling is hard coded into the program.
  - ❖ If processor loading is high, it can be very difficult and error-prone to make changes to the program.

GUC German University in Cairo
Prof. Ayman A. El-Badawy
Department of Mechatronics Engineering
Faculty of Engineering and Material Science

Introduction to Multitasking

- Accordingly, two abstractions must be introduced in order to build complex applications on microcontrollers:
  - ❖ The process, which is a single execution of a program. E.g., two different runs of the same program are considered two different processes.
  - ❖ The OS/kernel, which provides mechanisms for switching execution between different processes (Multitasking).
- The terms process and task are used interchangeably in this course, as is usually done in this field. A more precise definition would be that a task can be composed of multiple processes.
- Multiple processes that run concurrently and share a memory space are called threads.

GUC German University in Cairo
Prof. Ayman A. El-Badawy
Department of Mechatronics Engineering
Faculty of Engineering and Material Science

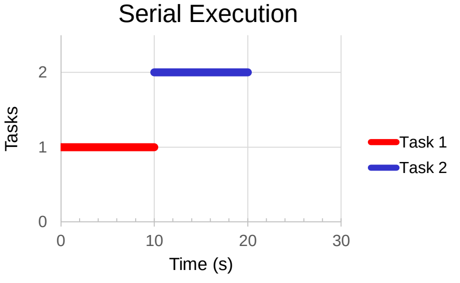
Introduction to Multitasking

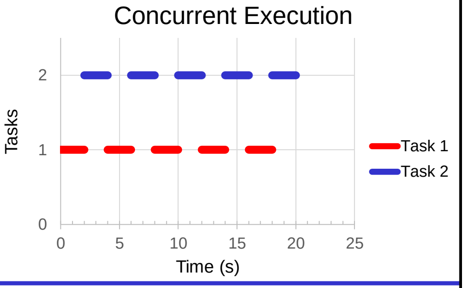
- Multitasking: Simultaneous execution of multiple tasks.
- Reasons for multitasking:
  1. Improve responsiveness to external stimuli. E.g., Sensor data.
  2. Improve performance by allowing a program to run simultaneously on multiple processors (for multi-processor systems).
  3. Directly control timing of external interactions, e.g., Periodic update of display at specific times.
- In this course, the freeRTOS real-time kernel is used: https://www.freertos.org/index.html

GUC German University in Cairo
Prof. Ayman A. El-Badawy
Department of Mechatronics Engineering
Faculty of Engineering and Material Science

Concurrency

- In a single-core processor, at any given point of time, only one process can be in a running state, while other processes are in a non-running state (e.g., waiting).
- Processor can switch between multiple processes, such that they appear to be running simultaneously.
- Different priorities can be assigned to each process. Accordingly, the scheduler determines which process should be running.

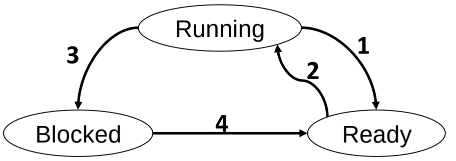




Prof. Ayman A. El-Badawy
Department of Mechatronics Engineering
Faculty of Engineering and Material Science

Concurrency

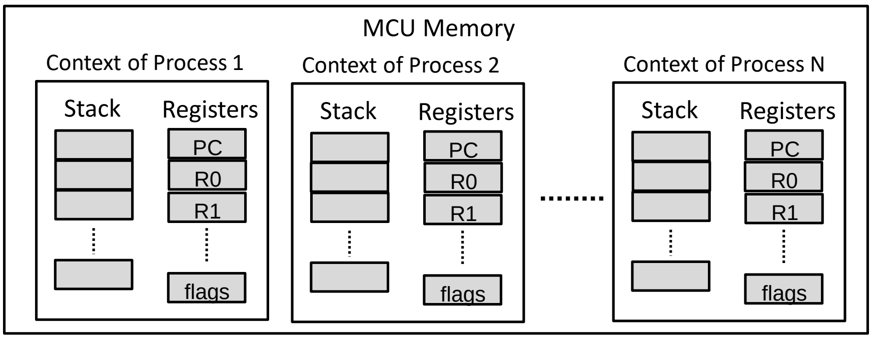
- Basic state diagram of a process:
  - ❖ Running State: the process is being executed (only one process can be in this state).
  - ❖ Ready State: Task ready and waiting to be executed.
  - ❖ Blocked: Waiting for an event to be complete, e.g., waiting for input from another process.
- State Transitions:
  1. (Running -> Ready) Scheduler selects another process to be in the running state.
  2. (Ready -> Running) Scheduler selects this process to run.
  3. (Running -> Blocked) Process blocks waiting for an event to occur.
  4. (Blocked -> Ready) The event occurs, and the process is ready to run.



Prof. Ayman A. El-Badawy
Department of Mechatronics Engineering
Faculty of Engineering and Material Science

Context Switching

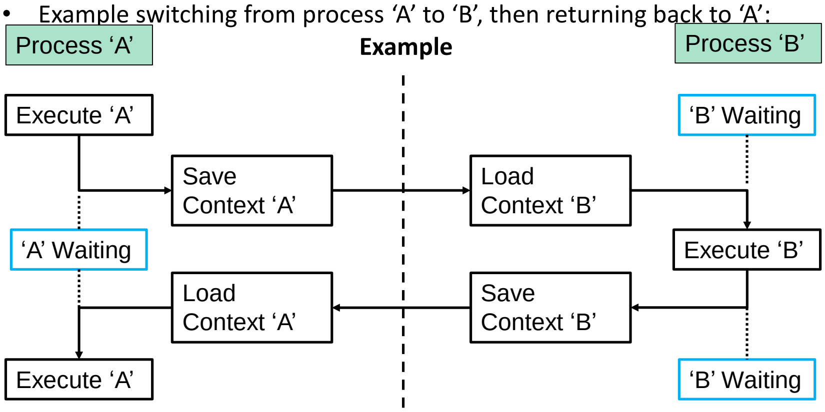
- Task execution context: During process execution, the microcontroller utilizes some registers, RAM and ROM like any program. These resources, including all CPU registers, the stack, etc., define the context of a process
- Each process maintains its own stack and register contents (context):



Prof. Ayman A. El-Badawy
Department of Mechatronics Engineering
Faculty of Engineering and Material Science

Context Switching

- Before switching between two processes, for example, from process ‘A’ to process ‘B’, a context switch from ‘A’ to ‘B’ first saves the context of ‘A’ and loads the context ‘B’ (This process is usually handled by the kernel/OS).
- Example switching from process ‘A’ to ‘B’, then returning back to ‘A’:



Prof. Ayman A. El-Badawy
Department of Mechatronics Engineering
Faculty of Engineering and Material Science

Real-time Systems

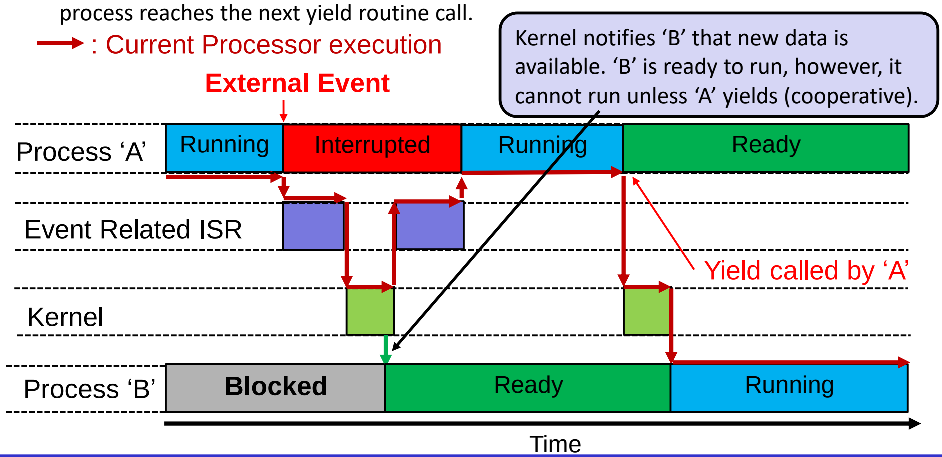
- In embedded systems, one of the major design requirement is to minimize the response time of the program. In other words, the processes should meet their deadlines.
- A soft-deadline requirement is one that state a time deadline, however, failing to meet this deadline will not result in failure of the system. e.g., Delayed response to keystrokes.
- A hard-deadline requirement is one that states a time deadline. Failing to meet this deadline will result in failure of the system. e.g., Excessively delayed response of an Antilock Braking System (ABS) in a car losing traction can be dangerous.
- A real-time kernel/scheduler is the part of the Real-time Operating System (RTOS) which determines what process should be running. It provides mechanisms to switch between processes. In addition, it handles communication, synchronization and priorities.
- The RTOS provide facilities required to satisfy real-time requirements.

Non-preemptive (Cooperative) Multitasking

- In non-preemptive multitasking, the running process is not interrupted unless the process itself calls a kernel routine (function) to perform the context switch.
- The process gives up control over the processor by calling this kernel routine, allowing another process to run. The context switch call is usually referred to as a “yield” in this multitasking technique.
- Problems with Cooperative Multitasking:
  1. If a currently running process does not call the yield routine frequently other processes may be starved.
  2. It is difficult to predict the worst-case response time of non-preemptive multitasking system.

Non-preemptive (Cooperative) Multitasking

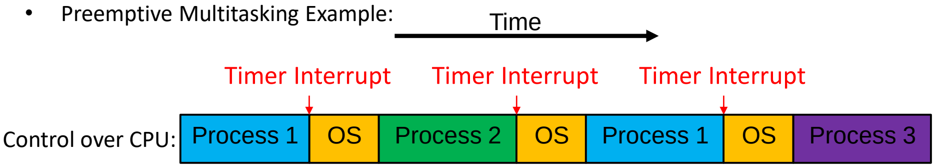
- Cooperative context switch example: If an external event occurs, the processor can be currently executing a process, named ‘A’, other than the one designed to process the event, named ‘B’. In this case, the event handling process will not run until the current process reaches the next yield routine call.



GUC
Prof. Ayman A. El-Badawy
Department of Mechatronics Engineering
Faculty of Engineering and Material Science

Preemptive Multitasking

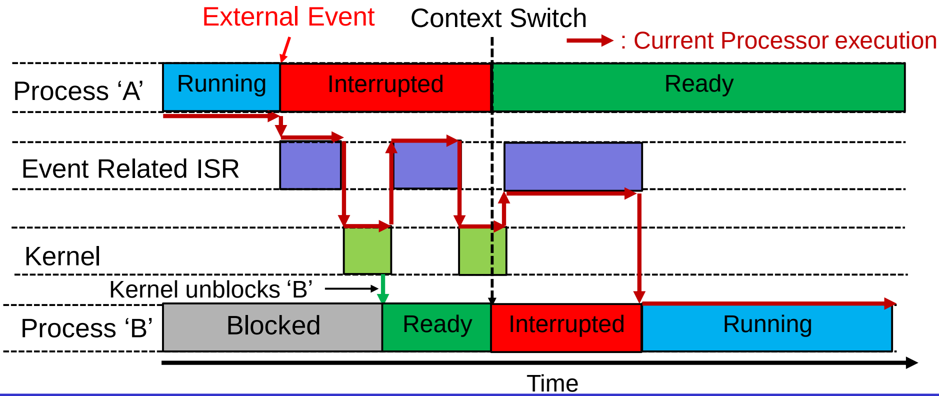
- In preemptive multithreading, hardware interrupts are used to trigger the context switching. , e.g., timer compare match interrupt.
- Most kernels include a system-clock ISR, that runs at fixed time intervals. This ISR allows maintaining a system clock and allows periodic invocation of the scheduler.
- By using this technique, System response time is significantly improved because the sequencing of process execution is appropriately driven by the order of external events.
- No need to call the yield routine in code, since the kernel regains control over the processor through hardware interrupts and the system-clock interrupt.
- Preemptive Multitasking Example:



GUC
Prof. Ayman A. El-Badawy
Department of Mechatronics Engineering
Faculty of Engineering and Material Science

Preemptive Multitasking

- Preemptive Context switch example: In case of an external event, the ISR corresponding to this event is called. The ISR includes calls to the kernel routine that allows the event handling process ‘B’ waiting for the data to be unblocked. Afterwards, the kernel decides which process needs the processor most (which can be the process to handle the external event ‘B’), then a context switch is performed to the most urgent process.



Prof. Ayman A. El-Badawy
Department of Mechatronics Engineering
Faculty of Engineering and Material Science

FreeRTOS Real-time Kernel

- FreeRTOS Overview:
  - Open-source light weight real-time kernel for microcontrollers and small microprocessors.
  - Provides mechanisms for both preemptive and cooperative multitasking.
  - Provides mechanisms for inter-process communication (communication between different tasks and ISRs).
  - Flexible task priority assignment and task notification mechanism.
  - Supports various processor architectures and compilers, including some MCUs from the AVR ATMega family and Arduino.
  - FreeRTOS: https://www.freertos.org/

Prof. Ayman A. El-Badawy
Department of Mechatronics Engineering
Faculty of Engineering and Material Science

FreeRTOS Task Management

- In FreeRTOS, a task may be defined as a C function. The only condition is that it should have a certain prototype defined as follows:

```
void ATaskDefinitionFunction( void *pvParameters );
```

- For a C function to be a valid task definition, it should have a return type "void" and a parameter of type pointer to void (void*).

- Typically, FreeRTOS tasks should run forever (include an infinite loop), and tasks should never include 'return' statements. If a task is no longer needed, it should be deleted using an API function.

- Typical task function definition implementation example:

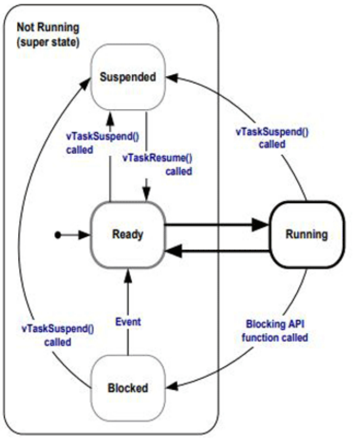
```
void ATaskDefinitionFunction( void *pvParameters )
{
        // Task Initialization code and variable definitions here
        for(;;) // Infinite loop
        {/* Code to implement the task functionality here*/ }
}
```

Prof. Ayman A. El-Badawy
Department of Mechatronics Engineering
Faculty of Engineering and Material Science

FreeRTOS Task Management

- A single task function definition can be used to create any number of tasks—each created task being a separate execution instance, with its own stack and its own copy of any automatic (stack) variables defined within the task itself.

- Full state machine of a task in freeRTOS:



Prof. Ayman A. El-Badawy
Department of Mechatronics Engineering
Faculty of Engineering and Material Science

FreeRTOS Task Management

- To create a task from task definition, use the `xTaskCreate()` Application Programming Interface (API) function.
- Prototype of `xTaskCreate()`:

```
BaseType_t xTaskCreate( TaskFunction_t pvTaskCode, const char * pcName, uint16_t
usStackDepth, void *pvParameters, UBaseType_t uxPriority, TaskHandle_t
*pxCreatedTask );
```

  - `pvTaskCode:` Pointer to function that implements the task, such as "`ATaskDefinitionFunction`" in the previous page.
  - `pcName:` Name of function mainly used for debugging.
  - `usStackDepth:` Specifies the stack size allocated to this task (defined in words, not bytes).
  - `pvParameters:` parameters passed to the task.
  - `uxPriority:` priority of the task, 0 is lowest priority, while (`configMAX_PRIORITIES` – 1) is highest priority.
  - `pxCreatedTask:` passes a handle to the task being created, can be used in API calls.

GUC German University in Cairo
Prof. Ayman A. El-Badawy
Department of Mechatronics Engineering
Faculty of Engineering and Material Science

FreeRTOS Task Management

- FreeRTOS Arduino port has a system clock tick interrupt every 15 ms by default (period adjustable). The clock ticks are managed by the watchdog timer in this port.

- If a task does not need to perform any processing for a specific duration, This running task can be placed in the block state for a specified number of clock ticks using the API function `vTaskDelay()`. The blocked task does not use any processing time allowing other tasks to utilize the processor.

- Prototype:

```
void vTaskDelay( TickType_t xTicksToDelay );
```

`xTicksToDelay:` The number of clock ticks for which the task will be blocked.

- Macro `pdMS_TO_TICKS( duration_ms )` can be used to convert duration_ms to clock ticks.

- Example call to block task for 30 ms:

```
vTaskDelay( pdMS_TO_TICKS(30) ); // block this task for 30 ms
```

GUC German University in Cairo
Prof. Ayman A. El-Badawy
Department of Mechatronics Engineering
Faculty of Engineering and Material Science

FreeRTOS Arduino Example

- Blinking an LED on pin 2 and toggle another LED on pin 4 whenever a button on pin 3 is pressed:

- Blinking Task Definition:

```
// program page: 1
#include <Arduino_FreeRTOS.h>

void TaskBlink(void *pvParameters)  // A task that blinks an LED connected to pin 2.
{
  pinMode(2, OUTPUT); // set Pin No. 2 as output

  for (;;) // A Task shall never return or exit.
  {
    digitalWrite(2, HIGH);   // turn the LED on (HIGH is the voltage level)
    vTaskDelay( pdMS_TO_TICKS(1000) ); // wait for one second
    digitalWrite(2, LOW);    // turn the LED off by making the voltage LOW
    vTaskDelay( pdMS_TO_TICKS(1000) ); // wait for one second
  }
}
```

GUC German University in Cairo
Prof. Ayman A. El-Badawy
Department of Mechatronics Engineering
Faculty of Engineering and Material Science

FreeRTOS Arduino Example

- Toggle Task Definition:

```
// program page: 2
void TaskToggle(void *pvParameters)  // This task toggles an LED connected to pin 4
{                                    // whenever the button connected to pin 3 is pressed.
  char buttonState = 0; // Initialize variables and I/O
  char prevButtonState=0;
  char ledState = LOW;
  pinMode(3,INPUT);
  pinMode(4,OUTPUT);
  for (;;)//Infinite loop for task execution
  {
   /************* Read button state with debouncing *************/
   if(digitalRead(3))
   {
      vTaskDelay(pdMS_TO_TICKS(15)); // Task waits (block state) for 15 ms to perform debouncing
      if(digitalRead(3))
        buttonState=HIGH;
   }else
   {
      vTaskDelay(pdMS_TO_TICKS(15)); // Task waits (block state) for 15 ms to perform debouncing
      if(!digitalRead(3))
          buttonState=LOW;
   }
   /************* Check for a rising edge *************/
      if(!prevButtonState && buttonState)
      {
        ledState= !ledState; // toggle pin 4
        digitalWrite(4, ledState);
      }
}}
```

GUC German University in Cairo
Prof. Ayman A. El-Badawy
Department of Mechatronics Engineering
Faculty of Engineering and Material Science

FreeRTOS Arduino Example

- Toggle Task Definition:

```
// program page: 3
void setup() {

  // Now set up two tasks to run independently.
  xTaskCreate(TaskBlink, "Blink", 128, NULL, 2, NULL);// create blink task to toggle an LED every 1 second.

  xTaskCreate(TaskToggle, "Echo Task", 128, NULL, 1, NULL); // create led toggle task.

  // Now the task scheduler, which takes over control of scheduling individual tasks, is automatically
started.
}

void loop()
{
  // Empty infinite loop. Processing is done in the tasks.
}
```

GUC German University in Cairo
Prof. Ayman A. El-Badawy
Department of Mechatronics Engineering
Faculty of Engineering and Material Science

FreeRTOS AVR GCC Example

- A simple approach to start development using freeRTOS is to use the freeRTOS demo as a starting point, then remove un-necessary code and add the application specified code.

- Before writing code using the AVR GCC port, it is important to configure the freeRTOS kernel as required by the application :

- Most configuration flags can be found in the FreeRTOSConfig.h file, these flags include:

```
configCPU_CLOCK_HZ: define the MCU clock speed in Hz

configUSE_PREEMPTION: set this flag to 1 for preemptive multitasking or 0 for
cooperative

configMINIMAL_STACK_SIZE: minimum stack size used by a task

configTOTAL_HEAP_SIZE:  total heap size for dynamic memory allocation

configCHECK_FOR_STACK_OVERFLOW: detection of stack overflow in any of the
                running tasks
```

GUC German University in Cairo
Prof. Ayman A. El-Badawy
Department of Mechatronics Engineering
Faculty of Engineering and Material Science

FreeRTOS AVR GCC Example

- Blinking an LED on pin 2 and toggle another LED on pin 4 based whenever a button is pressed on pin 3

```
#include <avr/io.h>
/* Scheduler include files. */
#include "FreeRTOS.h"
#include "task.h"

/* Priority definitions for most of the tasks in the demo application. */
#define LED_BLINK_PRIORITY          ( tskIDLE_PRIORITY + 2 )
#define TOGGLE_PRIORITY ( tskIDLE_PRIORITY + 2 )


/* The task function for blinking an LED every two seconds. */
void BlinkLEDTask( void* pvParameters)
{
            DDRD |= 0b00000001; // set pin PD0 as output for LED
            /*Infinite loop toggle LED every second */
            for(;;){
            PORTD^=0b00000001;
            vTaskDelay( pdMS_TO_TICKS(1000)); // wait for one second
            }
}
```

GUC German University in Cairo
Prof. Ayman A. El-Badawy
Department of Mechatronics Engineering
Faculty of Engineering and Material Science

FreeRTOS AVR GCC Example

- Button press triggered LED toggle task:

```
void toggleTask( void* pvParameters){
    DDRD |= 0b00000010; // set pin PD1 as output
    DDRD &= 0b11111011; // set pin PD2 as input
    char buttonState = 0;
    char prevButtonState=0;
    for (;;)
    {
        /************* read button state with debouncing *************/
        if(PORTD & 0b00000100) // button pressed
        {
            vTaskDelay(pdMS_TO_TICKS(15)); // task waits (blocks) for 15 ms (debouncing)
            if(PORTD & 0b00000100)
                buttonState=1;
        }else
        {
            vTaskDelay(pdMS_TO_TICKS(15)); // task waits (blocks) for 15 ms (debouncing)
            if(!(PORTD & 0b00000100))
                buttonState=0;
        }
            /************* Check for a rising edge *************/
        if(!prevButtonState && buttonState)
        {
            PORTD^=0b00000010; //toggle PD1 LED
        }
    }
}
```

GUC German University in Cairo
Prof. Ayman A. El-Badawy
Department of Mechatronics Engineering
Faculty of Engineering and Material Science

FreeRTOS AVR GCC Example

- Button press triggered LED toggle task:

```
int main(void){

    xTaskCreate( BlinkLEDTask, "LED", 128, NULL, mainLED_BLINK_PRIORITY, NULL );
    xTaskCreate(toggleTask, "Toggle", 128, NULL, TOGGLE_PRIORITY, NULL);
    vTaskStartScheduler(); // start scheduler
}
```

GUC German University in Cairo
Prof. Ayman A. El-Badawy
Department of Mechatronics Engineering
Faculty of Engineering and Material Science