Multitasking Basics

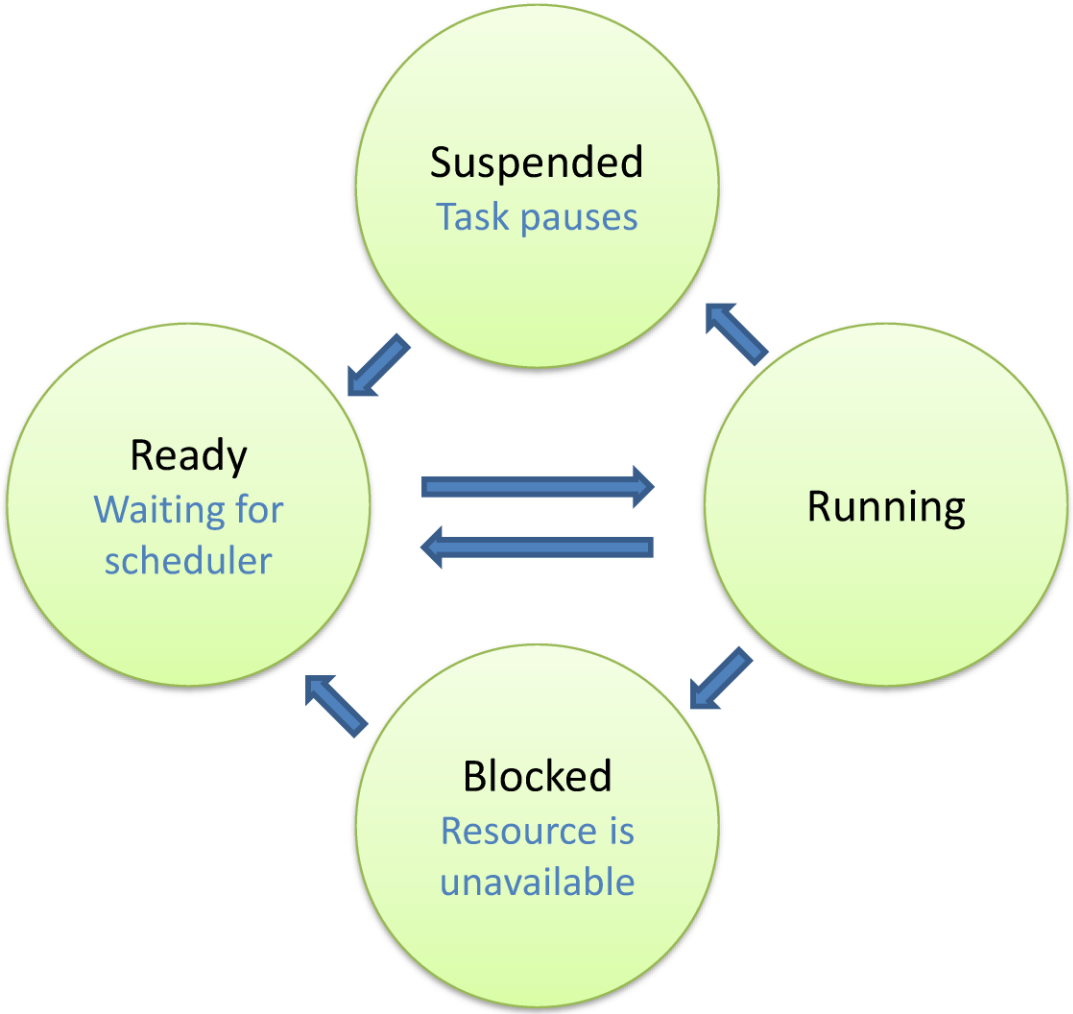
Networks and Embedded Software

Module 6.2.1

by Wolfgang Neff

Scheduler (1)

- Task states
  - Ready
  - Running
  - Suspended
    - Dormant
  - Blocked
    - Waiting



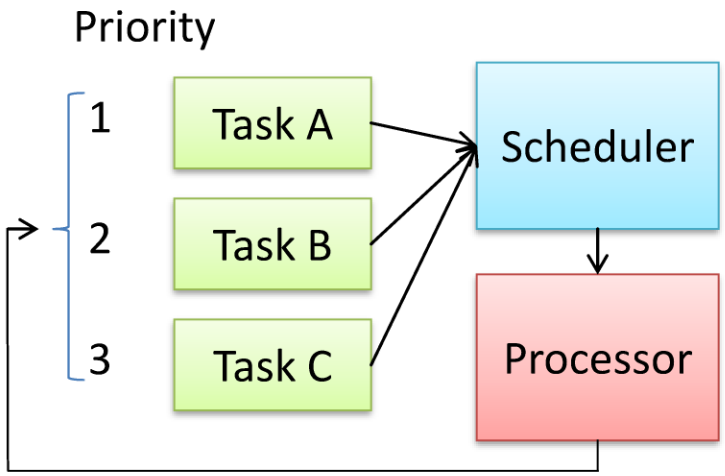
Scheduler (2)

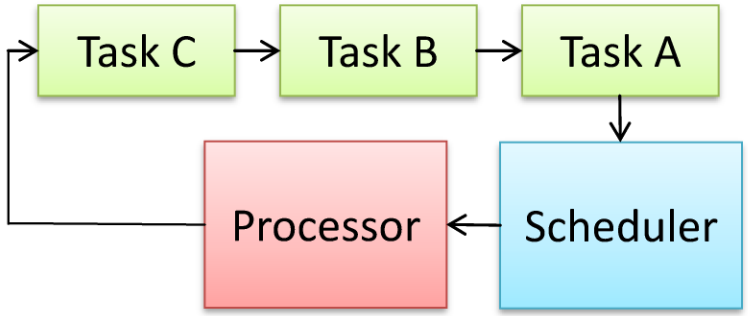
- Task control: FreeRTOS example
  - Task control
    - `void vTaskSuspend(TaskHandle_t xTaskToSuspend);`
    - `void vTaskResume(TaskHandle_t xTaskToResume);`
    - `void vTaskDelay(const TickType_t xTicksToDelay);`
      - Example

```
void vTask (void* pvParameters) {
    while (1) {
        …
        // Suspend for 500 ms
        vTaskDelay(pdMS_TO_TICKS(500));
    }
}
```

Scheduler (3)

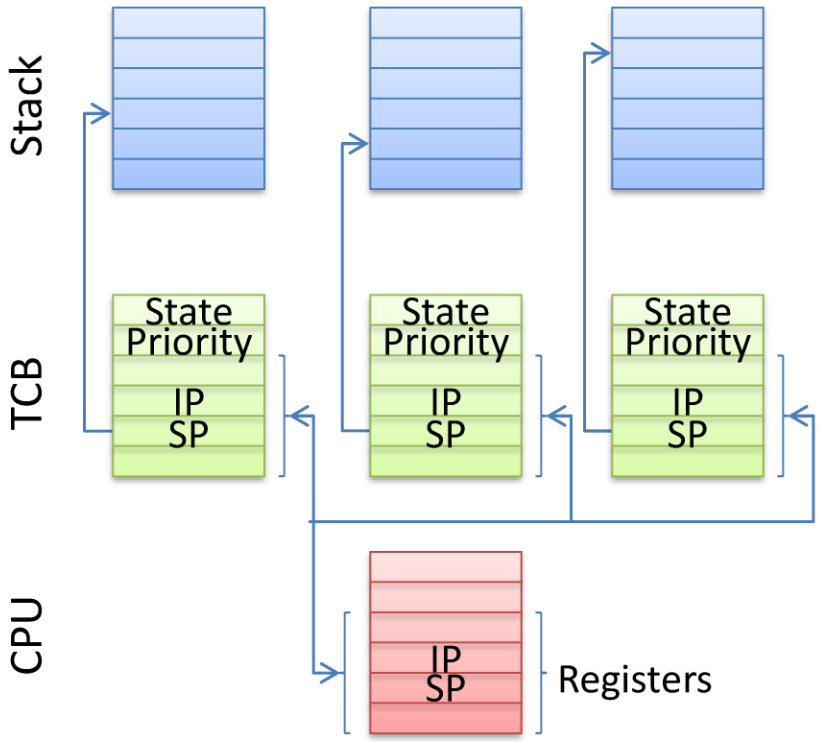
- Enables multitasking
- Assigns time slices
- Selects next task to run
- Basic algorithms
  - Priority based
  - Round-robin





Scheduler (4)

- Context Switch
  - Context must be saved
    - Registers
      - Instruction Pointer
      - Stack Pointer
    - Task State
    - Task Priority
  - Context is saved in Task Control Block



Scheduler (5)

- Timing
  - Timing of tasks is unknown
    - Implementation of scheduler is unknown
    - Interrupts can influence scheduling
    - Tasks may be blocked by external devices
  - No assumption on timing must be made
  - Algorithms must not depend on timing
    - Result of the algorithm must not depend on the execution sequence of the different tasks

Race Conditions (1)

- Pre-emption
  - Tasks can be inter-rupted at any time
  - Interruption may corrupt data
  - Software must be thread-safe

```
int temp;
void swap(int *a, int *b)
{   // not thread-save
    temp = *a;
    *a = *b;
    *b = temp;
}
```

```
void swap(int *a, int *b)
{   // thread-save
    int temp;
    temp = *a;
    *a = *b;
    *b = temp;
}
```

Race Conditions (2)

- Basic concepts
  - Unexpected results
  - Reason
    - Result depends on the timing of tasks
    - The timing of tasks is unknown
  - Precondition
    - Multitasking (at least two concurring tasks)
    - Common resource (data, hardware)
    - Modification of the resource (write access)

Race Conditions (3)

- Example 1 (common resource is hardware)

Tick	Task 1	Printer	Task 2
	`print("abc");`		`print("123");`
0	`out('a');`	a	
1	`out('b');`	ab	
2		ab1	`out('1');`
3	`out('c');`	ab1c	
4		ab1c2	`out('2');`
5		ab1c23	`out('3');`

Pre-emption

Pre-emption

Unexpected result

Race Conditions (4)

- Example 2 (common resource is data: data race)

Tick	Task 1			Common	Task 2		
	`swap(4,5);`	a=4	b=5	temp=?	a=1	b=2	`swap(1,2);`
0	`temp = a;`	a=4	b=5	temp=4	a=1	b=2	
1	`a = b;`	a=5	b=5	temp=4	a=1	b=2	
2		a=5	b=5	temp=1	a=1	b=2	`temp = a;`
3		a=5	b=5	temp=1	a=2	b=2	`a = b;`
4		a=5	b=5	temp=1	a=2	b=1	`b = temp;`
5	`b = temp;`	a=5	b=1	temp=1			
Task 1 lost the race: unexpected result!					Task 2 won the race!		

Pre-emption

Pre-emption

Race Conditions (5)

- Remedies
  - No common resources
    - Not always possible
      - Hardware
      - Communication
  - No multitasking
    - Prevent pre-emption
    - Performance issues
    - Must be very small

```
int temp;
void swap(int *a, int *b)
{
    // now thread-save
    taskENTER_CRITICAL();
    temp = *a;
    *a = *b;
    *b = temp;
    taskEXIT_CRITICAL();
}
```