µC/OS

Motivation

- So far we were programming on **bare metal** hardware, building everything we needed by outself

- As an application gets more complex you need to manage a lot of different tasks

→ consider using an **operating system**

Motivation

An operating system is useful for:

- running multiple processes or functions in parallel (maybe even with different priorities)
- memory management
- using standard protocols (TCP/IP, HTTP)
- file system management
- portability (running on different processors)

Motivation

- most MPUs are too small to run a standard OS like Windows or even Linux
- moreover those are not suitable for real-time applications

µC/OS-II™ The Real-Time Kernel

Micriμm Embedded Software

µC/OS

- very small real-time kernel (~20kB)
- allows defining several functions in C which execute independently in parallel
- supports many platforms (ARM, AVR, Altera, Xilinx, etc.)
- used in different areas (avionics, automotive, medical equipment, consumer electronics...)



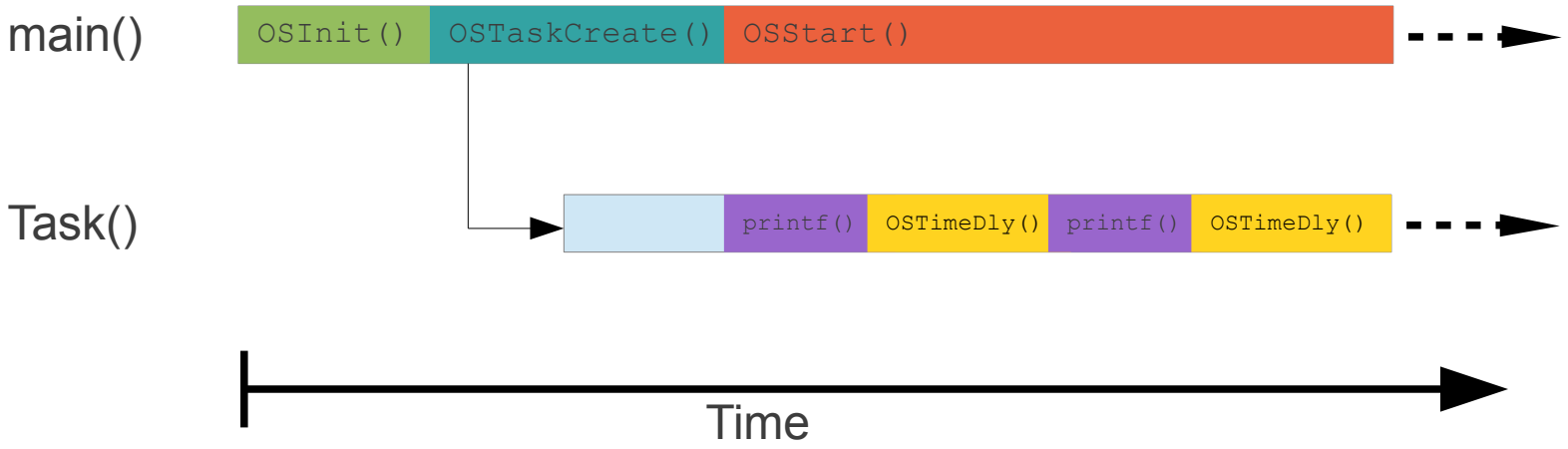
µC/OS Example

```
OS_STK  Task1Stk[1024];        //task's stack

void main(void)
{
    OSInit();                  //Initialize µC/OS-II

    OSTaskCreate(              //Create a new task
                 Task,             //task function
                 (void *)0,        //pointer to args
                 &Task1Stk[1023], //pointer to stack
                 25);              //task priority

    OSStart();                 //Start Multitasking
    //should never reach here
}
```

µC/OS Example

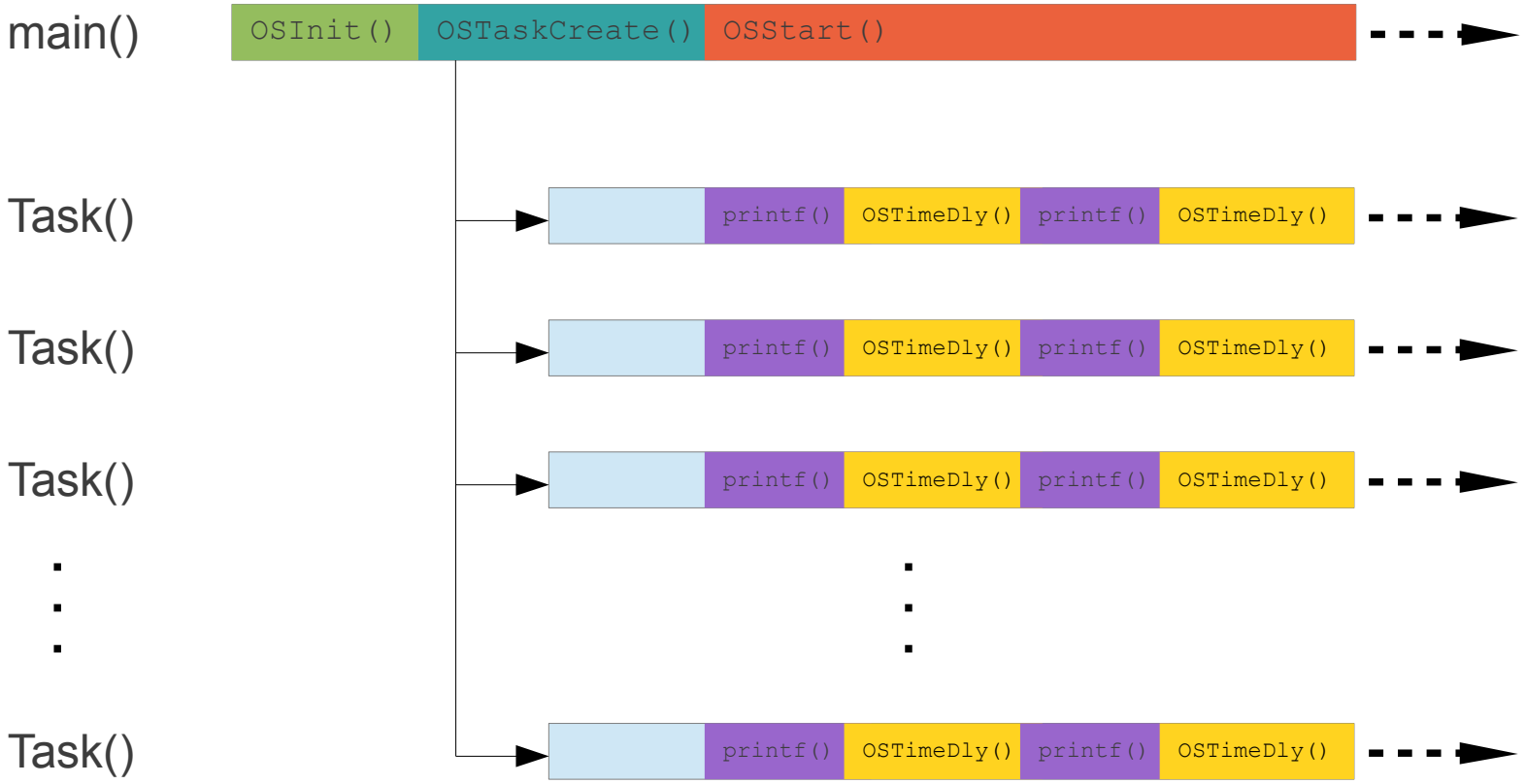
```
void Task (void* pdata)
{
    while(1)    //Task body, always an infinite loop
    {
        printf("Hello World from Task!\n");

        OSTimeDlyHMSM(    //sleep
                      0,      //hours
                      0,      //minutes
                      1,      //seconds
                      0);     //milliseconds
    }
}
```

µC/OS Example



µC/OS Example

How to run multiple tasks in parallel?



µC/OS Example

```
#define N_TASKS 5
OS_STK  TaskStk[N_TASKS][1024];//one stack per task

void main(void)
{
    int i;
    OSInit();
    for(i=0; i<N_TASKS; i++){
        OSTaskCreate(
                Task2,
                (void *)i,     //argument to Task2()
                &TaskStk[i][1023],  //unique stack!
                i); //task priority must be unique!
    }
    OSStart();
}
```

µC/OS Example

```
void Task2 (void* pdata)
{
    int i=(int)pdata;//same i as assigned in main()
    while(1)
    {
        printf("Hello World from Task #%d!\n", i);

        OSTimeDlyHMSM(0,0,1,0);
    }
}
```

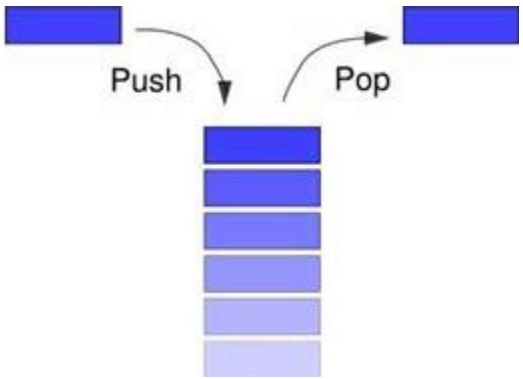
µC/OS Example

Output:

```
Hello World from Task #0!
Hello World from Task #1!
Hello World from Task #2!
Hello World from Task #3!
Hello World from Task #4!
Hello World from Task #5!
Hello World from Task #0!
Hello World from Task #1!
```

Stacks

- A task stack contains the stack's local variables and registers (e.g. program counter)
- **Each task needs an own stack!**
- Make sure it is large enough to contain all variables



Priority

- Tasks with a lower **priority value** will run before tasks with a higher value (Rate Monotonic Scheduling)
- A task's priority is also its **identifier**, i.e it has to be unique!

Sleeping

- With **OSTimeDlyHMSM()** a task can delay its execution and allows other tasks to run in the meantime

- e.g remove busy waiting from ADC:

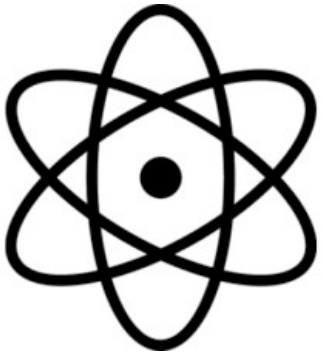
```
while(!done)
{
  spi_response=IORD(ADC_SPI_READ_BASE, 0);
  done=(spi_response & DONE_FLAG);
  OSTimeDlyHMSM(0,0,0,1);
}
```

Synchronization

- Sometimes multiple tasks need to access the same resource
- Since tasks run in parallel this leads to **race conditions** and **nondeterministic behavior**
- Use **semaphores** for synchronization

Semaphores

A semaphore is basically an **atomic integer counter** with two main oper



- **OSSemPend()**: decrements (––) the c
  and **blocks** if the counter is negative

- **OSSemPost():** increments (++) the counter and **wakes up** a task blocked by OSSemPend() if counter is >= 0

Semaphores

- You have to initialize a semaphore with **OSSemCreate()**

```
OS_EVENT* sem = OSSemCreate(5); //starting value: 5
//up to 5 processes can access the resource at a time
```

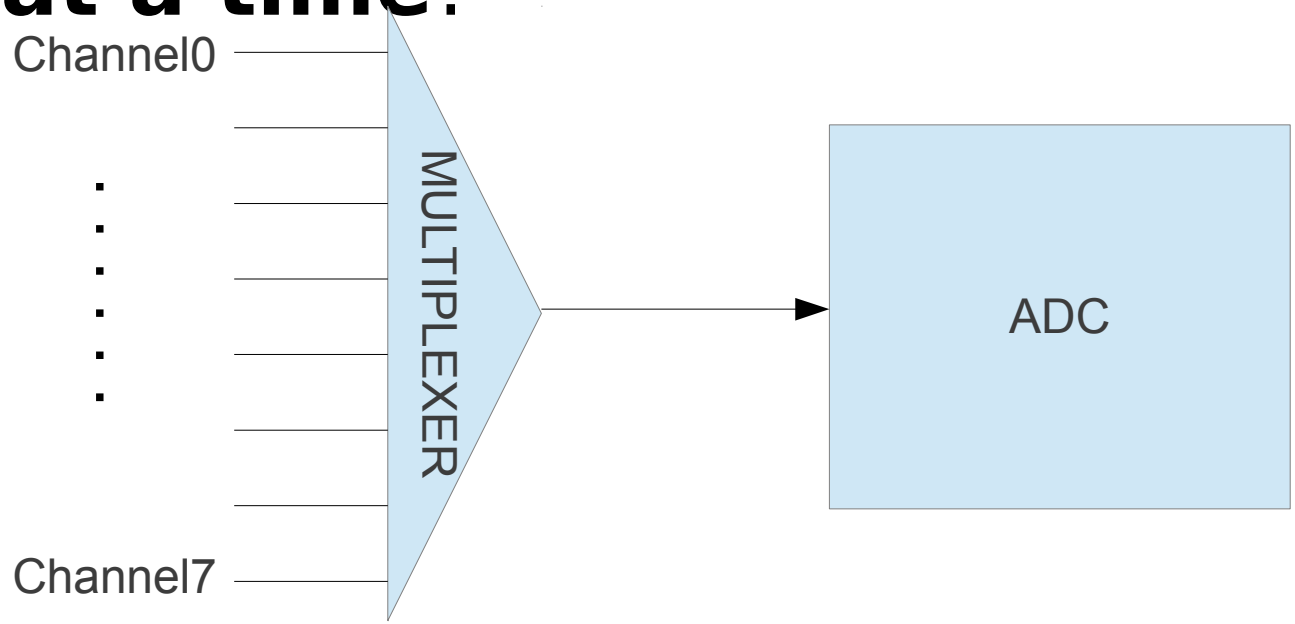
- You can assign a timeout to the blocking operation in **OSSemPend()**

```
INT8U status;
OSSemPend(sem, 10000, &status); //10000 clock ticks
if(status == OS_ERR_TIMEOUT){ printf("Timeout!"); }
```

Example: ADC

The ADC on the de0-nano is a limited resource.

It can read from **only one channel at a time**!



Example: ADC

- If you change the channel while a conversion is in process you will get incorrect values!

- To read from multiple channels from different tasks use a shared semaphore

Example: ADC

- in main():

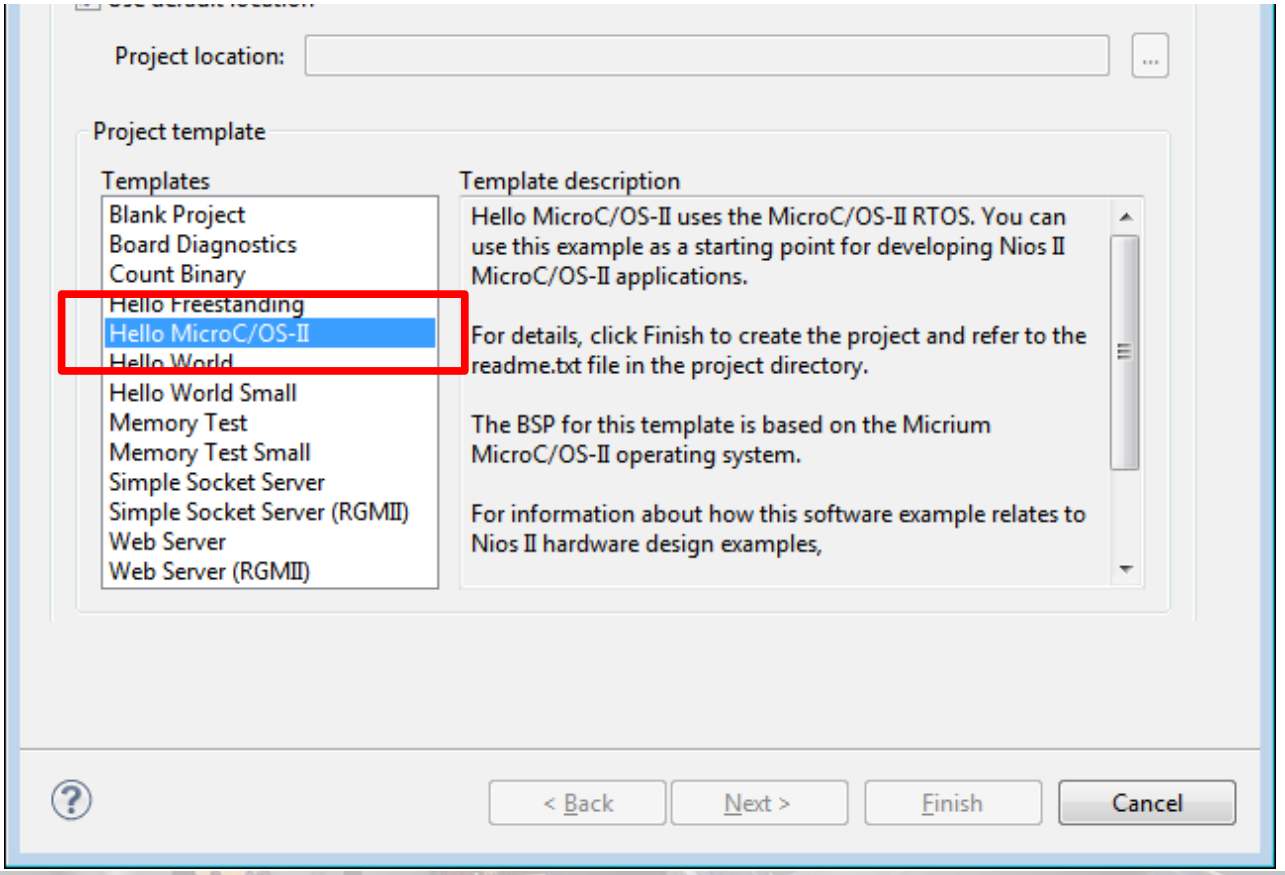
```
sem=OSSemCreate(1); //starting value 1
                    //only one access at a time
```

- in Task():

```
INT8U status;
OSSemPend(sem, 0, &status);//timeout 0:wait forever

read_from_adc();

OSSemPost(sem);
```

Template

- Select “Hello MicroC/OS-II” when creating a new project



Exercise 1: LED Tasks

Create one task for each LED and turn them on and off at different speeds.

Use only one task function.

Exercise 2: Synchronization

- Take a look at the exercise template.
- Two simple tasks each print slowly a message but due to parallelism the messages are not clear
- Use semaphores for synchronization to find out what they are printing
- Make sure that one task finishes printing, before the other starts

Exercise 3: Number of tasks

Find out the maximum number of tasks you can run

Hint: check the return value of OSCreateTask() function

Exercise

UNTIL TOMMORROW AFTERNOON

- Program your car to avoid obstacles!
- Program your car to follow you!