Lab 8 Real-time OS - 1

Speaker: Hao-Yun Chin

Advisor: Prof. Tian-Sheuan Chang

Apr 27, 2004

Outline

- *Introduction to Real-time Operation System (RTOS)*
- Introduction to μC/OS-II
  - Features
  - Task & task scheduling
  - Start μC/OS-II
  - Port application

Real-time OS

- Real-time OS (RTOS) is an intermediate layer between hardware devices and software programming
- "Real-time" means keeping deadlines, not speed
- Advantages of RTOS in SoC design
  - Shorter development time
  - Less porting efforts
  - Better reusability
- Disadvantages
  - More system resources needed
  - Future development confined to the chosen RTOS

Soft and Hard Real Time

- Soft real-time
  - Tasks are performed by the system as fast as possible, but tasks don't have to finish by specific times
  - Priority scheduling
  - Multimedia streaming
- Hard real-time
  - Tasks have to be performed correctly and on time
  - Deadline scheduling
  - Aircraft controller, Nuclear reactor controller

Outline

- Introduction to RTOS
- ***Introduction to μC/OS-II***
  - ***Features***
  - ***Task & task scheduling***
  - ***Start μC/OS-II***
  - ***Port application***

μ C/OS-II

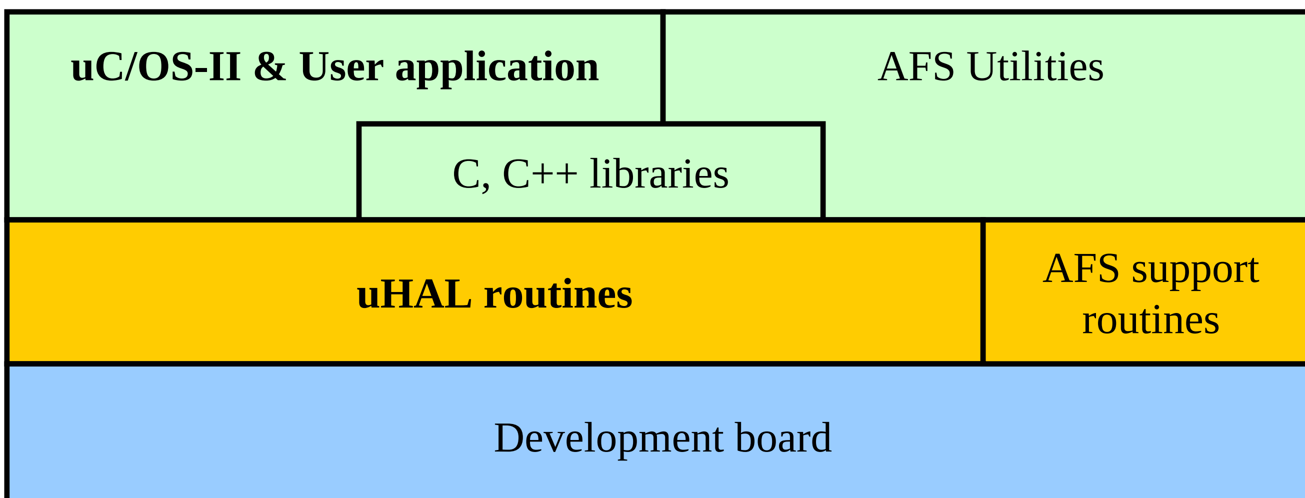
- Written by Jean J. Labrosse in ANSI C
- A portable, ROMable, scalable, preemptive, real-time, multitasking kernel
- Used in hundreds of products since its introduction in 1992
- Certified by the FAA for use in commercial aircraft
- Available in ARM Firmware Suite (AFS)
- Over 90 ports for free download
- http://www.ucos-ii.com

μ C/OS-II Features

- Portable — runs on architectures ranging from 8-bit to 64-bit
- ROMable — small memory footprint
- Scalable — select features at compile time
- Multitasking — preemptive scheduling, up to 64 tasks

μ C/OS-II vs. μ HAL

- uHAL (pronounced *Micro-HAL*) is the ARM Hardware Abstraction Layer that is the basis of the ARM Firmware Suite
- uHAL is a basic library that enables simple application to run on a variety of ARM-based development systems
- uC/OS-II use uHAL to access ARM-based hardware



Task

- Task is an instance of program
- Task thinks that it has the CPU all to itself
- Task is assigned a unique priority
- Task has its own set of stack
- Task has its own set of CPU registers (backup in its stack)
- Task is the basic unit for scheduling
- Task status are stored in Task Control Block (TCB)

Task Structure

Task structure:

- An infinite loop
- An self-delete function

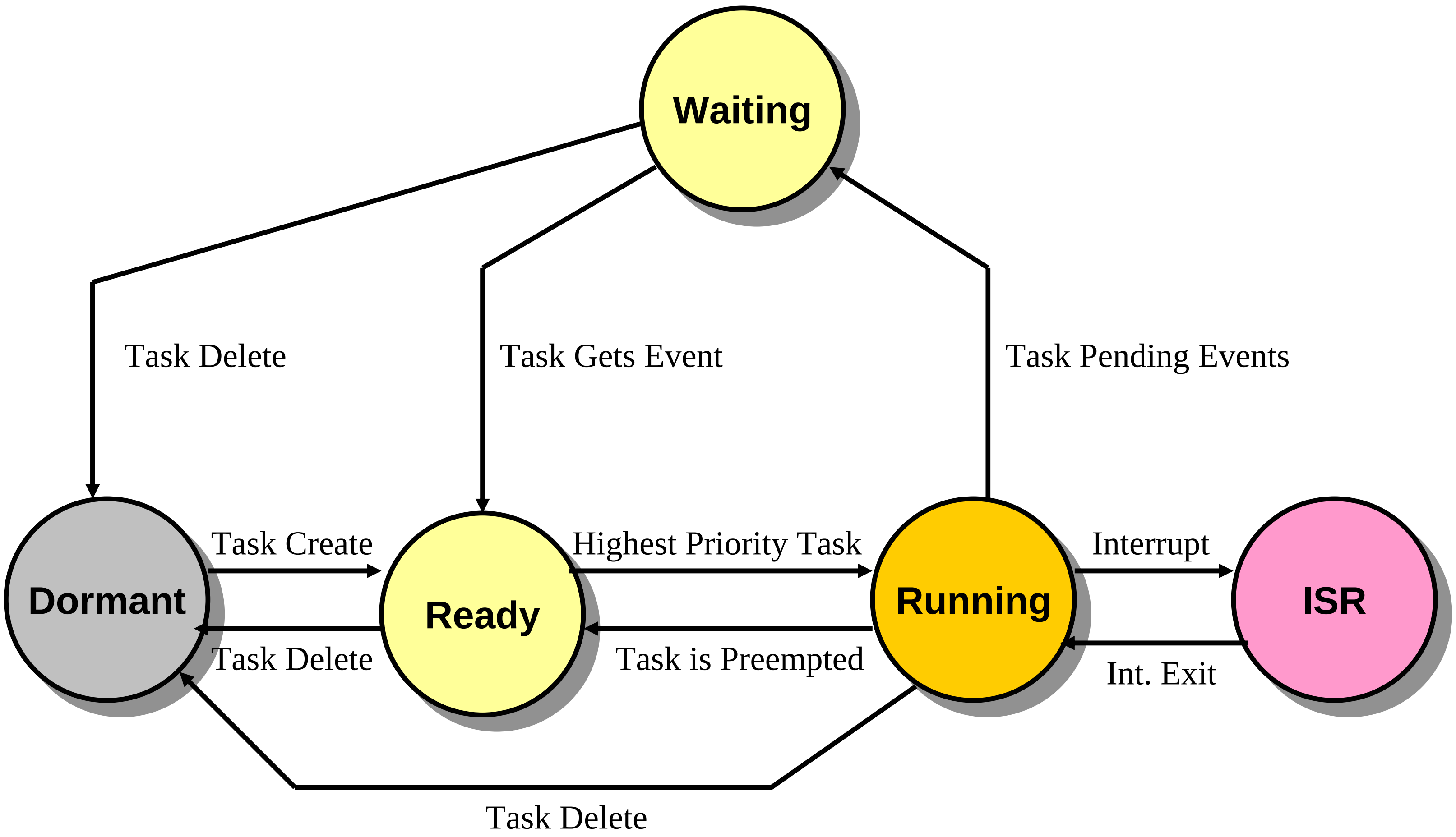
Task with infinite loop structure

```
void ExampleTask(void *pdata)
{
        for(;;) {
            /* User Code */
            /* System Call */
            /* User Code */
        }
}
```

Task that delete itself

```
void ExampleTask(void *pdata)
{
        /* User Code */
        OSTaskDel(PRIO_SELF);
}
```

Task States



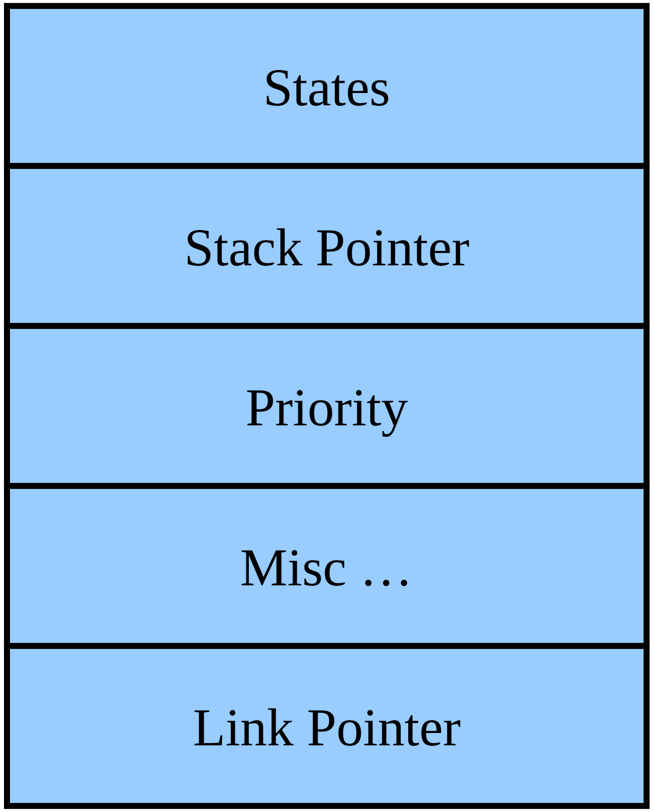
Waiting
Task Delete
Task Gets Event
Task Pending Events
Dormant
Task Create
Task Delete
Ready
Highest Priority Task
Task is Preempted
Running
Interrupt
Int. Exit
ISR
Task Delete

Task Priority

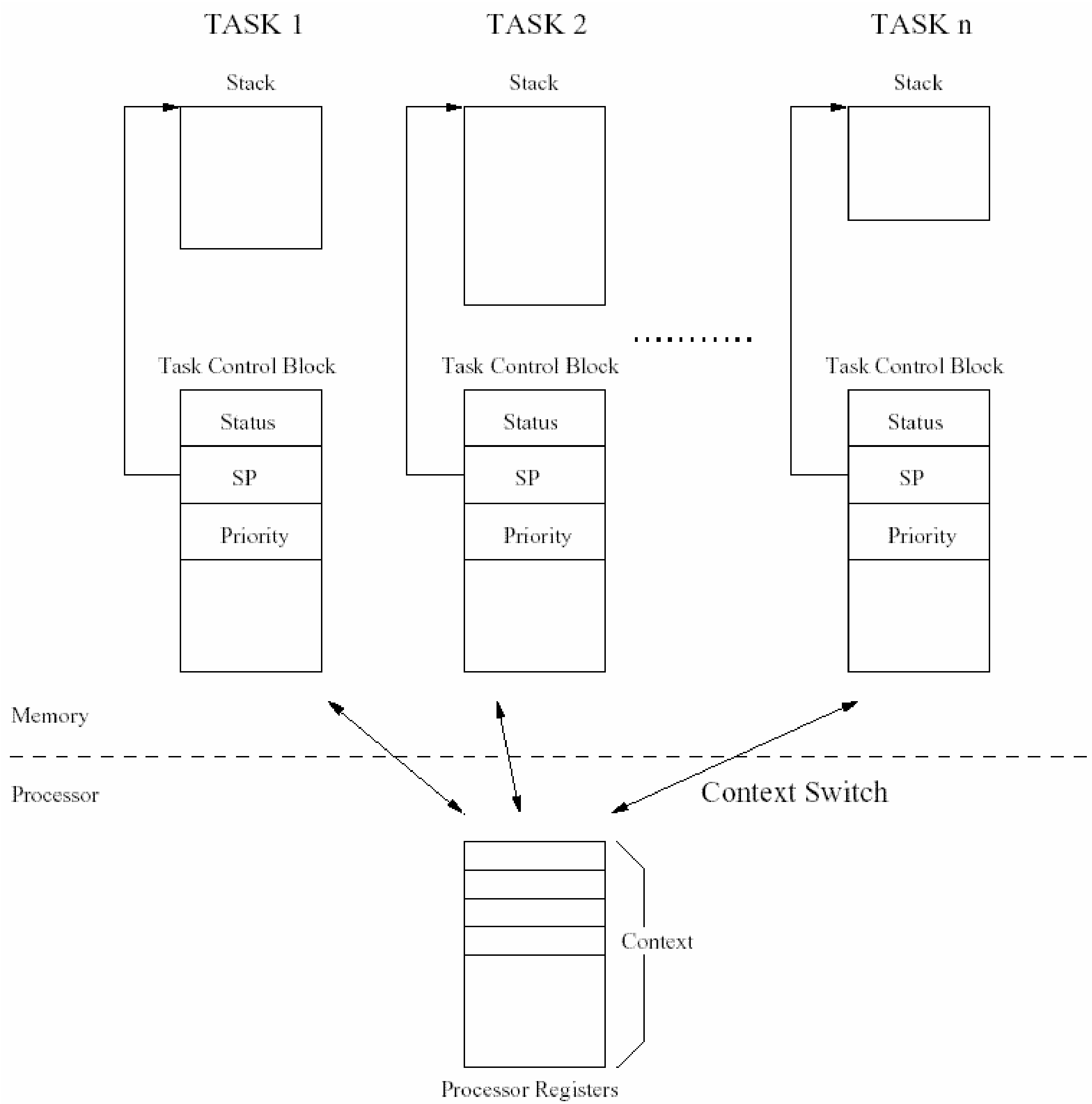
- Unique priority (also used as task identifiers)
- 64 priorities max (8 reserved)
- Always run the highest priority task that is READY
- Allow dynamically change priority

Task Control Block

uC/OS-II use TCB to keep record of each task



Task Control Block(cont.)



Exchanging CPU Control

Control returns from task to OS when Kernel API is called

```
void ExampleTask(void *pdata)
{
        for(;;) {
            /* User Code */
            /*System Call */
            /* User Code */
        }
}
```

uC/OS-II Kernel API

OSMboxPend();

OSQPend();

OSSemPend();

OSTaskSuspend();

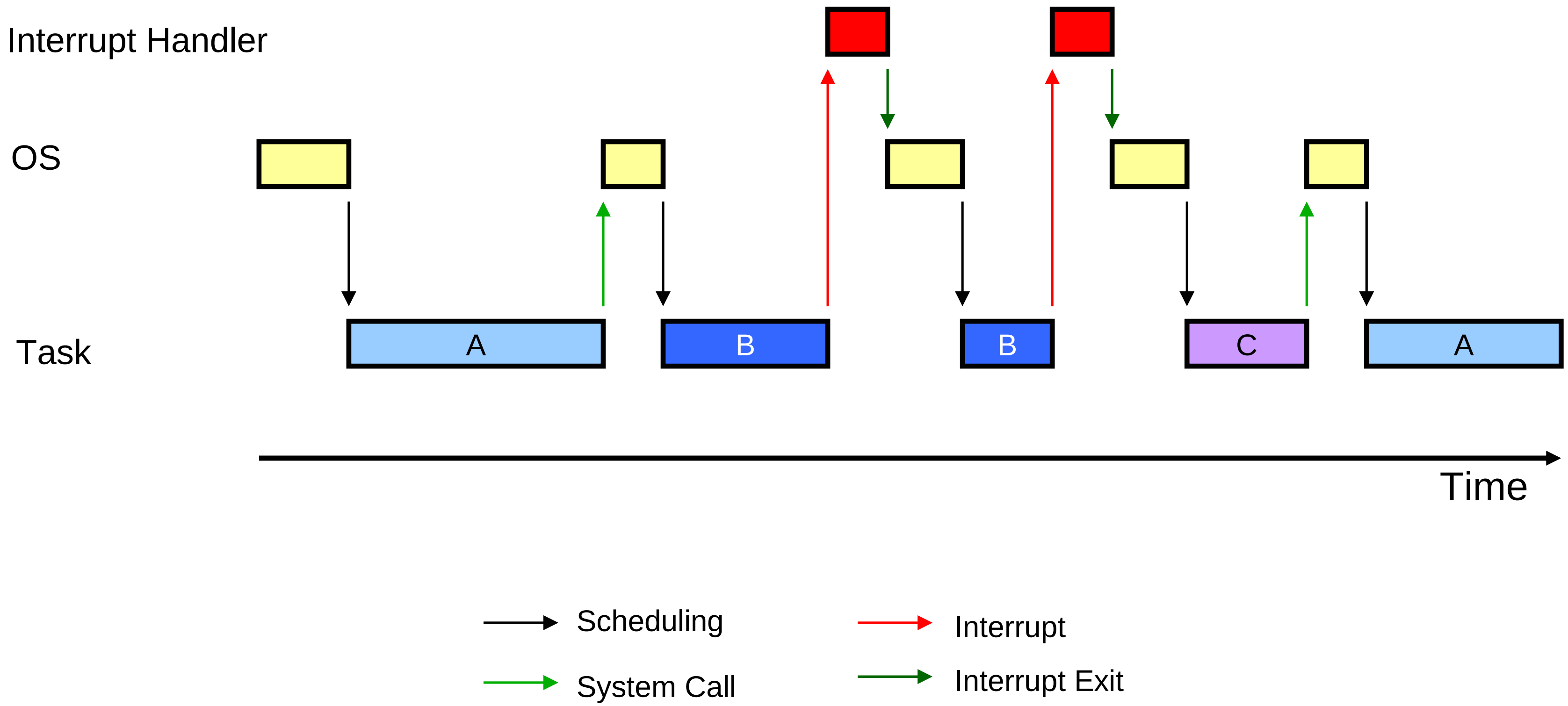
OSTimeDly();

OSTimeDlyHMSM();

More…

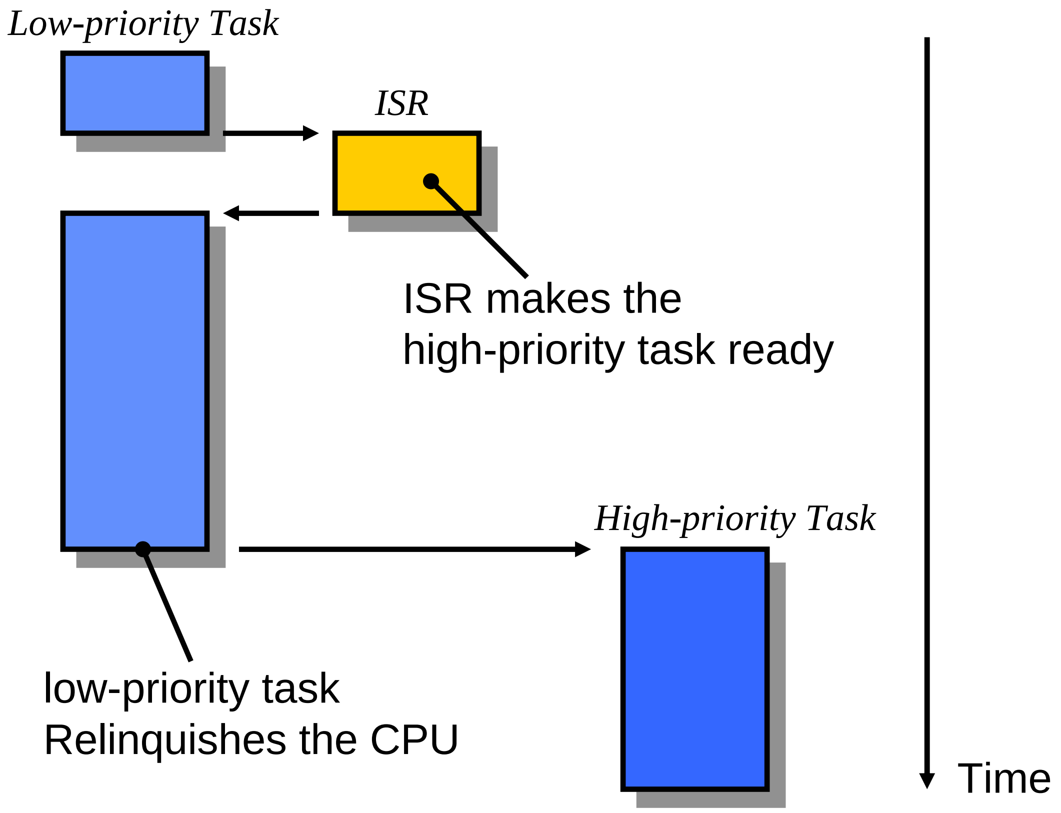
Exchanging CPU Control

Only one of OS, Task, Interrupt Handler gets CPU control at a time



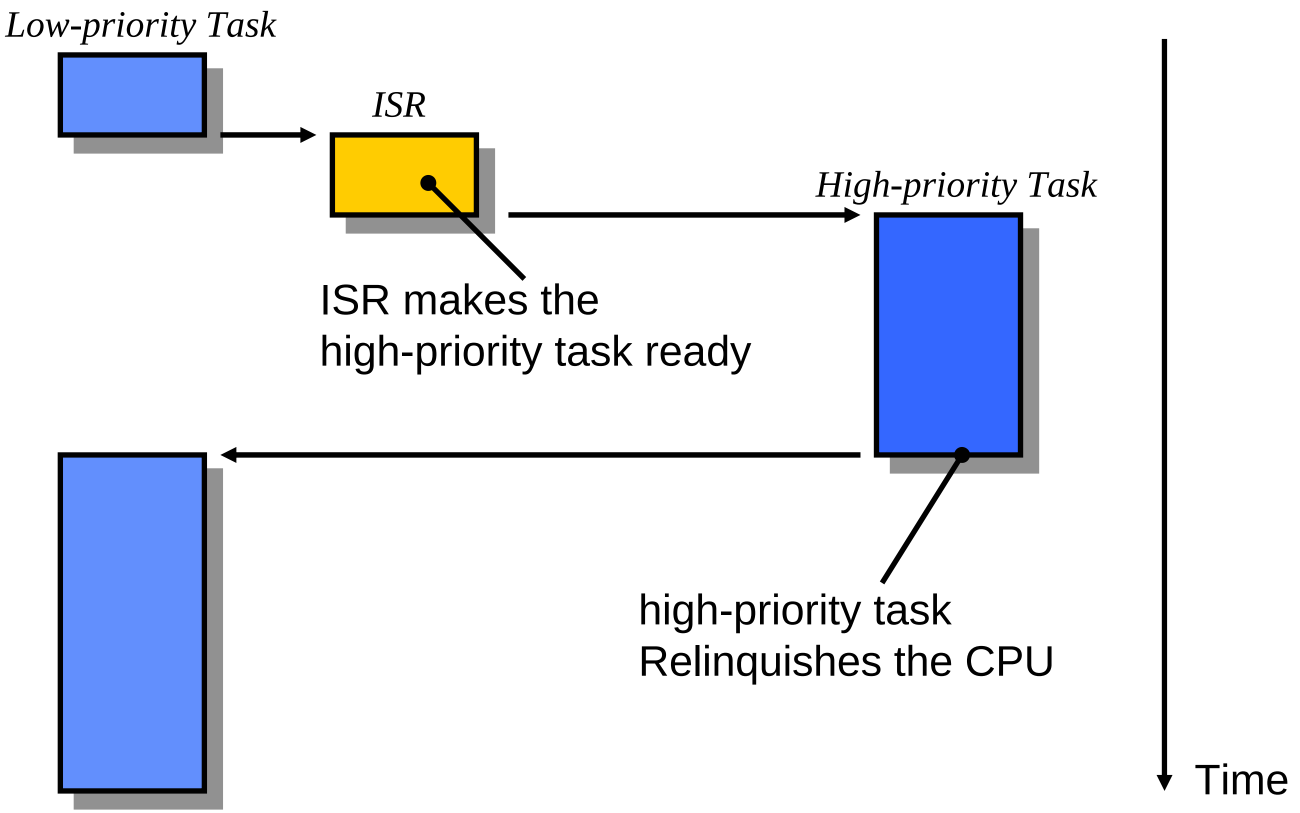
Task Scheduling

- Non-preemptive



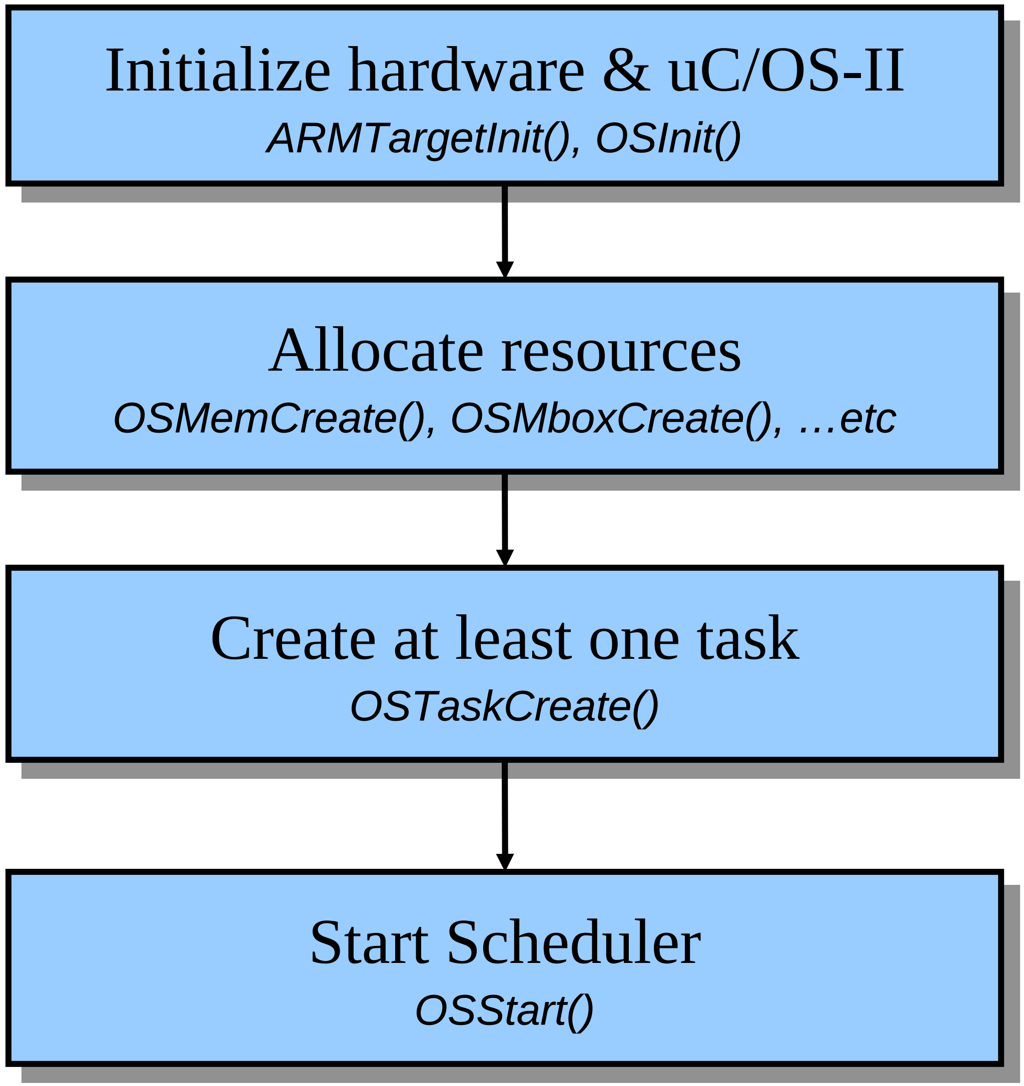
Task Scheduling

- Preemptive



uC/OS-II adopts preemptive scheduling

Starting μC/OS-II



Porting Application

Necessary coding changes

- variables
  - use local variables for preemption
  - use semaphore to protect global variables (resources)
- data transfer
  - arguments => mailbox/queue
- memory allocation
  - malloc() => OSMemCreate()
    OSMemGet()

Porting Application

assign task priorities

- unique priority level in uC/OS-II
  - only 56 levels available
  - priority can be change dynamically
- call OSTimeDly() in infinite loop task
  - ensure lower priority task get a chance to run

    MUST: if lower priority task is pending data from higher priority task

Lab 7:Real-time OS - 1

- Goal
  - A guide to use RTOS and port programs to it
- Principles
  - Basic concepts and capabilities of RTOS
    - Task, task scheduling
  - Coding guideline for a program running on the embedded RTOS
  - Setting up the ARMulator
- Guidance
- Steps
  - Building μC/OS-II
  - Porting Program to μC/OS-II
  - Building Program with μC/OS-II
- Requirements and Exercises
  - Write an embedded software for ID checking engine (single task)
- Discussion
  - What are the advantages and disadvantages of using RTOS in SOC design?

References

[1] AFS_Reference_Guide.pdf