操作系统研讨课
Course: B0911011Y

蒋德钧

Fall Term 2023-2024

email: jiangdejun@ict.ac.cn
office phone: 62601007

University of Chinese Academy of Sciences

Lecture 2 A Simple Kernel
（Part I）

2023.09.25

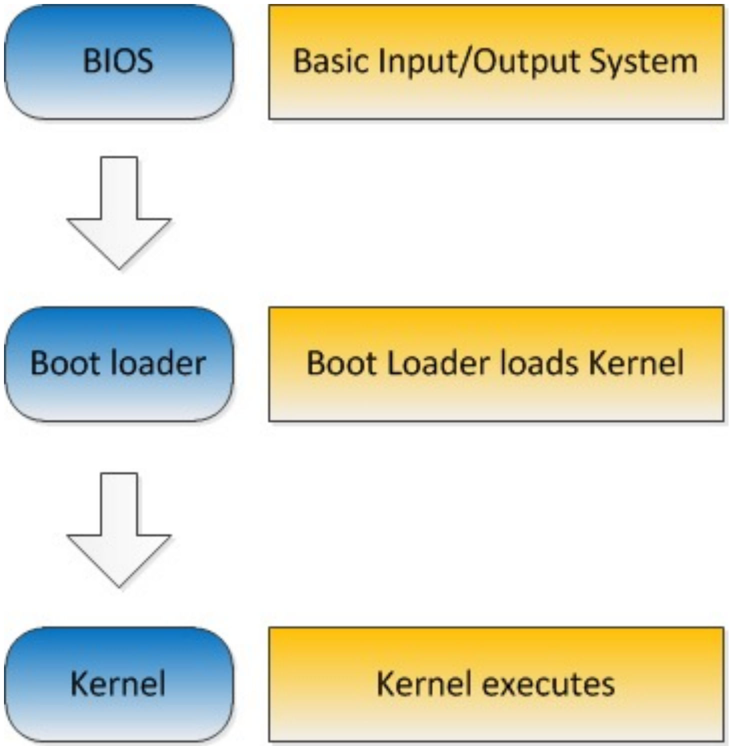
中国科学院大学
University of Chinese Academy of Sciences

Schedule

- Project 1 due
- Project 2 assignment

Project 2 – A Simple Kernel

- Booting procedure



Project 2 – A Simple Kernel

- Requirements (part I)
  - Write a simple kernel (non-preemptive)
    - Start a set of processes
    - Perform context switches between processes
    - Provide non-preemptive kernel support with context switch
    - Support basic mutex to allow BLOCK state of processes

Project 2 – A Simple Kernel

- A set of user processes
  - Program codes under the *test/test_project2* directory in start code
  - STRONGLY suggest to first read the codes of different tasks to understand what they do
  - You need to initialize these tasks in init_task_info(*main.c)* as you did in Project 1

Project 2 – A Simple Kernel

- Process Control Block (PCB)
  - A in-memory data structure in OS kernel containing the information to manage a process
  - Please refer to the given definition pcb_t (*include/os/sched.h*)

Project 2 – A Simple Kernel

- Process Control Block (PCB)
  - Process ID
  - Process status
  - Kernel stack pointer
  - User stack pointer
  - You may need to add new fields to PCB in subsequent projects

```
/* Process Control Block */
typedef struct pcb
{
    /* register context */
    // NOTE: this order must be preserved, which is defined in regs.h!!
    reg_t kernel_sp;
    reg_t user_sp;

    /* previous, next pointer */
    list_node_t list;

    /* process id */
    pid_t pid;

    /* BLOCK | READY | RUNNING */
    task_status_t status;

    /* cursor position */
    int cursor_x;
    int cursor_y;

    /* time(seconds) to wake up sleeping PCB */
    uint64_t wakeup_time;

} pcb_t;
```

Project 2 – A Simple Kernel

- Initialize PCB
  - Initialize PCBs for tested processes
    - Assign ID, set status/cursor_x/cursor_y
    - PCBs are organized as a linked list
  - Allocate memory for kernel stack and user stack
    - Please use allocKernelPage/allocUserPage functions (*kernel/mm/mm.c*)

地址范围	建议用途
0x50000000-0x50200000	BBL 代码及其运行所需的内存
0x50200000-0x50500000	Kernel 的数据段/代码段等
0x50500000-0x52000000	供内核动态分配使用的内存
0x52000000-0x52500000	用户程序的数据段/代码段等
0x52500000-0x60000000	供用户动态分配使用的内存

University of Chinese Ac

Project 2 – A Simple Kernel

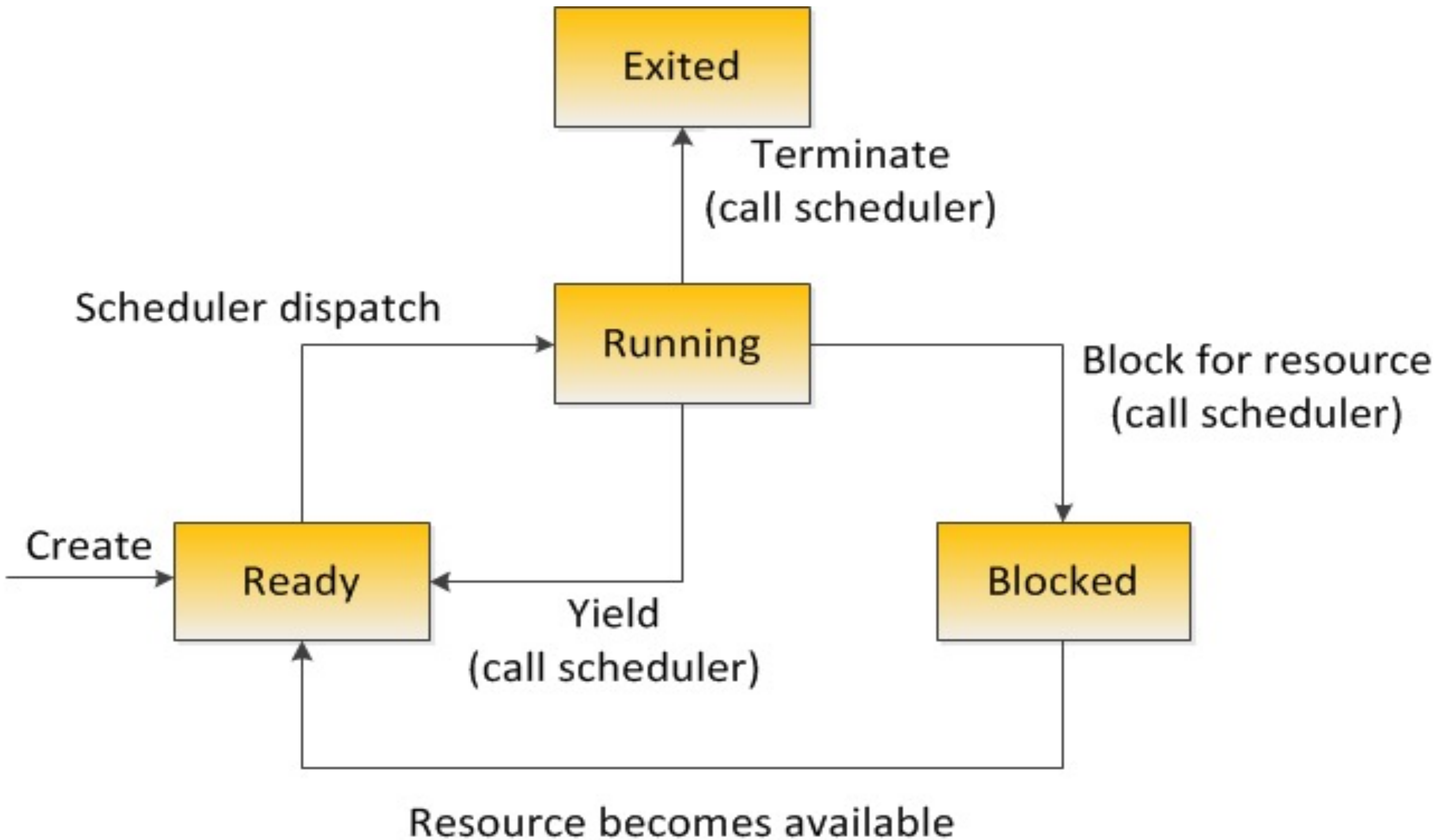
- Initialize kernel stack
  - Kernel stack is used to save and restore the context of a process

```
static void init_pcb_stack(
    ptr_t kernel_stack, ptr_t user_stack, ptr_t entry_point,
    pcb_t *pcb)
{
     /* TODO: [p2-task3] initialization of registers on kernel stack
      * HINT: sp, ra, sepc, sstatus
      * NOTE: To run the task in user mode, you should set corresponding bits
      *     of sstatus(SPP, SPIE, etc.).
      */
    regs_context_t *pt_regs =
        (regs_context_t *)(kernel_stack - sizeof(regs_context_t));


    /* TODO: [p2-task1] set sp to simulate just returning from switch_to
     * NOTE: you should prepare a stack, and push some values to
     * simulate a callee-saved context.
     */
    switchto_context_t *pt_switchto =
        (switchto_context_t *)((ptr_t)pt_regs - sizeof(switchto_context_t));

}
```

中国科学院大学
University of Chinese Academy

Project 2 – A Simple Kernel

- Scheduler (non-preemptive kernel)



Project 2 – A Simple Kernel

- Yield
  - A process itself releases the control of CPU by calling kernel scheduler
  - Kernel scheduler places the current running process to the end of the ready queue, and chooses another process to run
  - Please refer to kernel_yield() function

Project 2 – A Simple Kernel

- Context switch
  - Kernel scheduler executes context switch to run new process
  - Save context
    - Registers → Memory
  - Restore context
    - Memory → Registers

Project 2 – A Simple Kernel

- switch_to asm function
  - Work like a function call, but after the call the return address is another process
  - You need to save/restore process context in switch_to
  - Where to place the process context?
    - Kernel stack
  - Please refer to switchto_context_t (*include/os/sched.h*) for initializing kernel stack and performing context switch

Project 2 – A Simple Kernel

- Start a process
  - After initialization, the kernel calls do_scheduler

```
while (1)
{
    // If you do non-preemptive scheduling, it's used to surrender control
    do_scheduler();

    // If you do preemptive scheduling, they're used to enable CSR_SIE and wfi
    // enable_preempt();
    // asm volatile("wfi");
}
```

Project 2 – A Simple Kernel

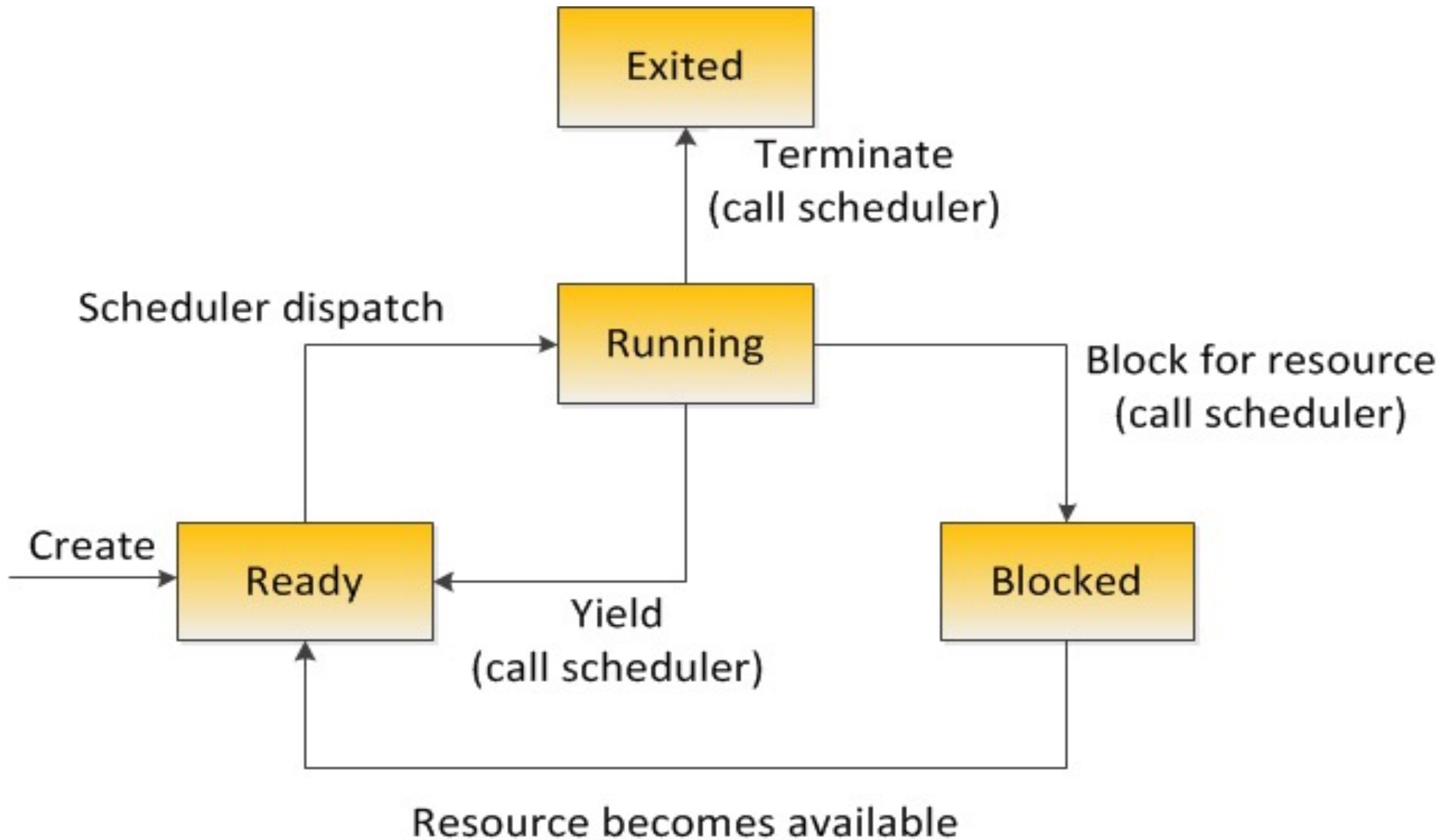
- Start a process
  - You need to choose a new process to run in do_scheduler
    - Round robin is a preferred scheduling algorithm
    - switch_to is called to perform context switch between the current running process and the new one

Project 2 – A Simple Kernel

- Start a process
  - Note that, when you start the first process, there is no current running user process, how do you handle this?
    - Please refer to pid0_pcb (*include/os/sched.h*)
    - It can be used for store context of kernel process (main.c)

Project 2 – A Simple Kernel

- Mutex lock



Project 2 – A Simple Kernel

- Mutex lock
  - What if no process currently holds the lock?
    - Acquire the lock
  - What if the lock is currently held?
    - Wait
  - Implement lock-related functions
  - Manage processes that do not acquire the lock
    - Ready queue vs. wait queue?

Project 2 A Simple Kernel

- Step by step – Task 1
  - Initialize PCBs for tested processes
  - Implement do_scheduler and switch_to
  - Start tested processes and support context switch among these processes as a non-preemptive kernel

Project 2 A Simple Kernel

- Step by step – Task 2
  - Implement mutex lock to support BLOCK state

Project 2 A Simple Kernel

- Requirement for design review
  - *switch_to*函数的工作机制是怎样的？是如何支持两个进程的上下文切换的?
  - 你在初始化PCB和内核栈时需要做哪些事情？
  - 你在switchto_context_t中会保存哪些内容?
  - 请介绍你设计的内核调度器(do_scheduler)的工作流程

Project 2 A Simple Kernel

- Requirement for design review
  - 当一个进程被阻塞时，内核会进行如何的处理？
    - 当一个进程被阻塞或获得资源时，内核会将进程的PCB放置在哪里？
    - 你设计的互斥锁机制能否支持一个进程请求多把锁？

Project 2 – A Simple Kernel

- Requirement for S/A/C-Core

Core type	Task requirements
S-Core	Tasks 1, 2
A-Core	NA
C-Core	NA

Project 2 – A Simple Kernel

- P2 schedule
  - 9th Oct.
    - P2 part I design review
    - Assign P2 part II
  - 16th Oct.
    - P2 part I due
    - P2 part II design review if it was assigned