Lecture 2 A Simple Kernel

2025.10.15

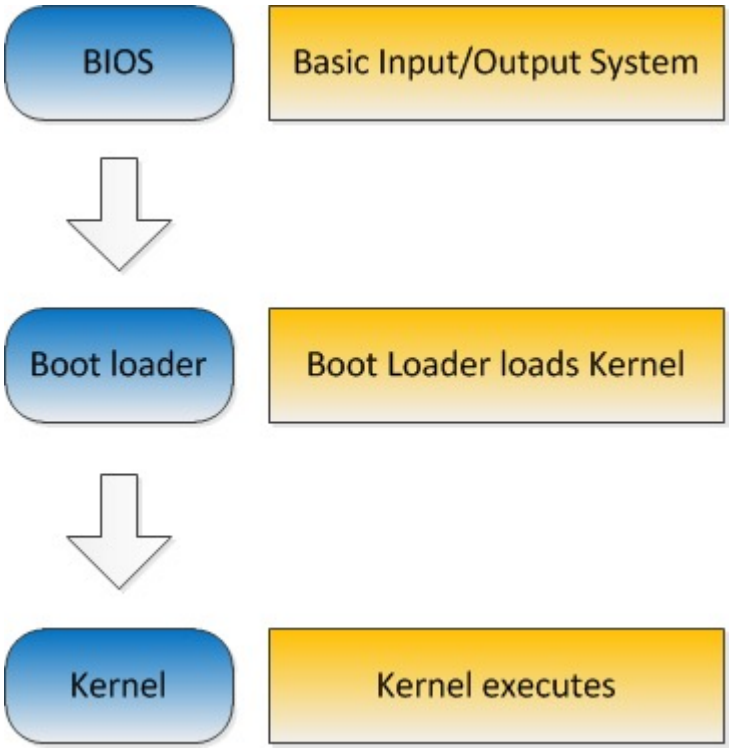
中国科学院大学
University of Chinese Academy of Sciences

Schedule

- Project 1 due
- Project 2-1 assignment

Project 2 – A Simple Kernel

- Booting procedure



Project 2 – A Simple Kernel

- Requirement I
  - Write a simple kernel (non-preemptive)
    - Process management
      - Start a set of processes
      - Provide context switches between processes with non-preemptive scheduling
      - Support basic mutex locking to enable BLOCK state of processes

Project 2 – A Simple Kernel

- A set of user processes
  - print1, print2, flyer
  - Please first read the codes of different tasks to understand what they do

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
  - Register context
  - You may need to add new fields to PCB in subsequent projects

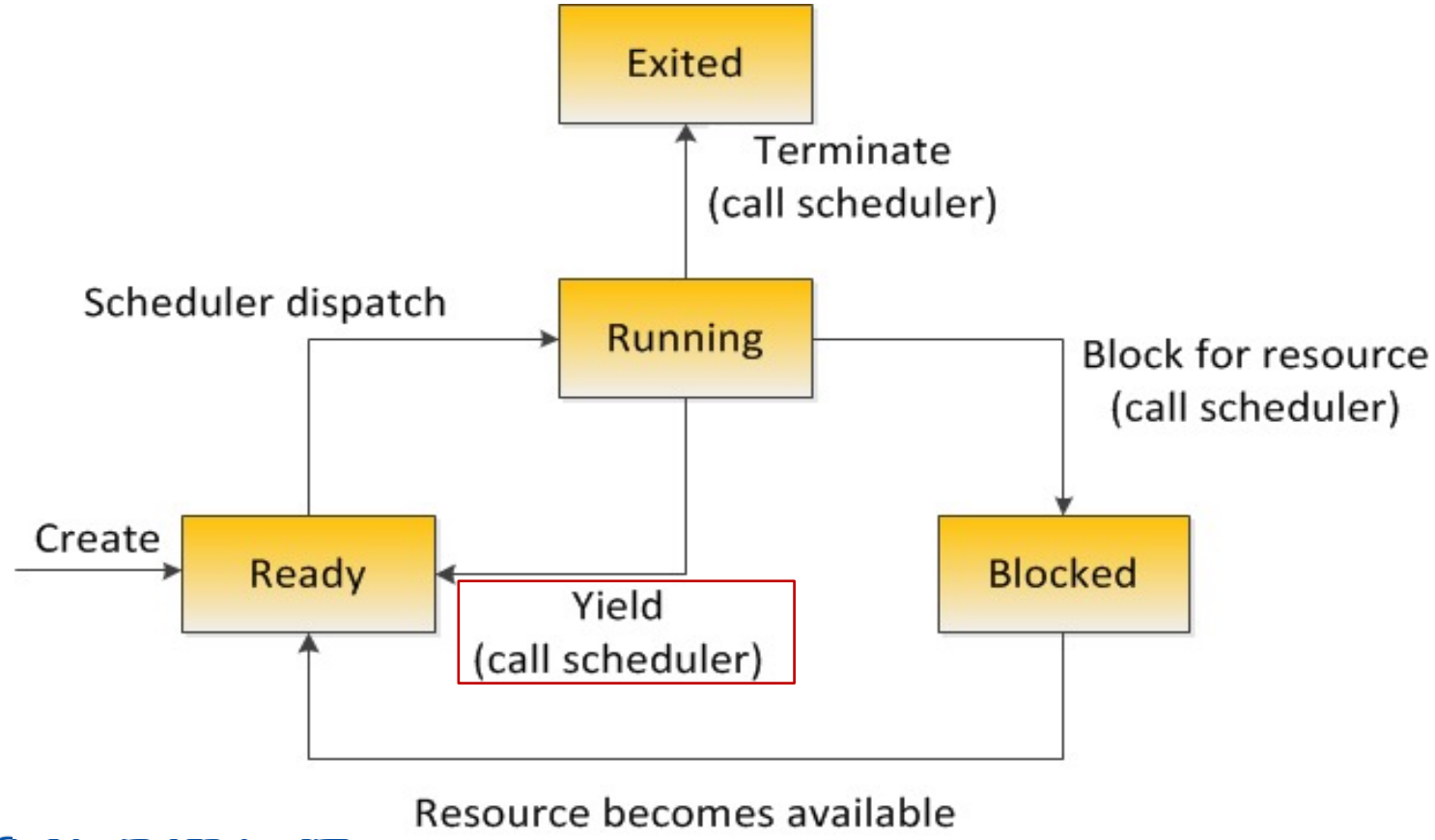
```c
/* Process Control Block */
typedef struct pcb
{
    /* register context */
    // NOTE: this order must be preserved, which is defined in regs.h!!
    reg_t kernel_sp;
    reg_t user_sp;

    /* 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 can be organized as a linked list
  - Allocate memory for kernel stack and user stack
    - Please use allocKernelPage/allocUserPage functions (*kernel/mm/mm.c*)

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
  - kernel/sched/sched.c *do_scheduler()*

```
void do_scheduler(void)
{
    // TODO: [p2-task3] Check sleep queue to wake up PCBs

    /************************************************************/
    /* Do not touch this comment. Reserved for future projects. */
    /************************************************************/

    // TODO: [p2-task1] Modify the current_running pointer.


    // TODO: [p2-task1] switch_to current_running

}
```

Project 2 – A Simple Kernel

- Context switch
  - Save context
    - CPU registers → kernel stack (in memory)
  - Restore context
    - kernel stack → CPU registers

Project 2 – A Simple Kernel

- Kernel stack initialization
  - Kernel stack is used to save and restore the context of a process
  - Pls. refer to *switchto_context_t*

```
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));

}
```

Project 2 – A Simple Kernel

- *switch_to* function
  - Work like a function call, but after the call the return address is another process
  - In switch_to, first save the process context of the previous process, and then restore the process context of the next process
  - Where to place the process context?
    - Kernel stack

Project 2 – A Simple Kernel

- *switch_to* function

```
// NOTE: the address of previous pcb in a0
// NOTE: the address of next pcb in a1
ENTRY(switch_to)
  addi sp, sp, -(SWITCH_TO_SIZE)

  /* TODO: [p2-task1] save all callee save registers on kernel stack,
   * see the definition of `struct switchto_context` in sched.h*/

  /* TODO: [p2-task1] restore all callee save registers from kernel stack,
   * see the definition of `struct switchto_context` in sched.h*/

  addi sp, sp, SWITCH_TO_SIZE
  jr ra
ENDPROC(switch_to)
```

新进程的入口地址

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

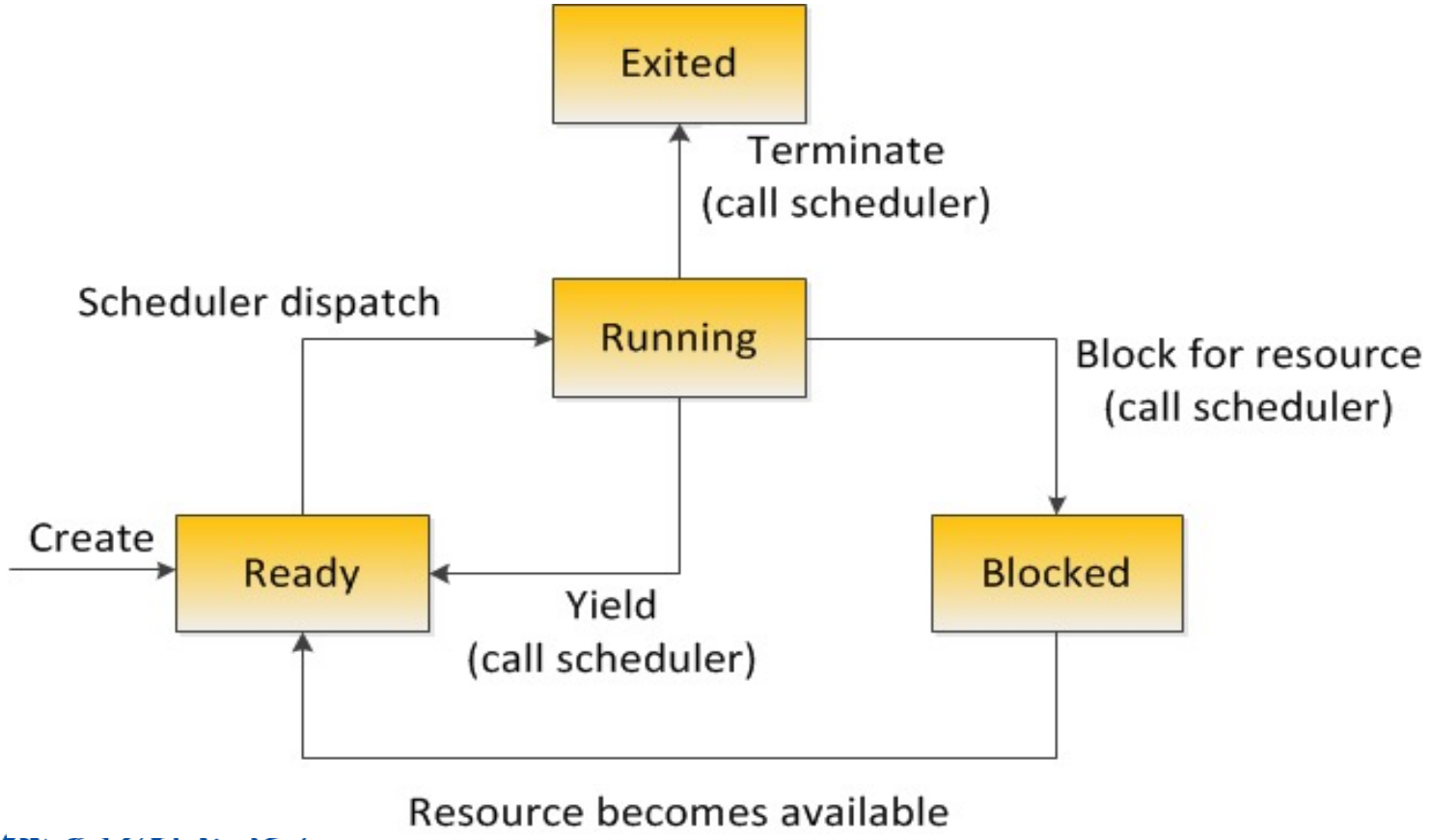
- Start a process
  - You need to choose a new process to run in *do_scheduler*
    - FIFO is a preferred scheduling algorithm
    - In *do_scheduler()*, switch_to is called to perform context switch between the current running process and the new one

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

- BLOCK state of a process



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-Core

Core type	Task requirements
S-Core	Tasks 1, 2

Project 2 – A Simple Kernel

- P2 schedule
  - 22nd Oct.
    - P2-1 review
    - P2-2 assign
  - 29th Oct.
    - P2-1 due