操作系统研讨课

蒋德钧

Fall Term 2019-2020

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

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

Lecture 2 A Simple Kernel (Part II)

2019.09.23

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

Schedule

- Project 2 part I design review
- Project 2 part II assignment

Project 2 – A Simple Kernel

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

Project 2 – A Simple Kernel

- Requirements (Part II)
  - Write a simple kernel (preemptive)
    - Provide preemptive kernel support with clock interrupt handler and priority based scheduling
    - Support system calls

Project 2 – A Simple Kernel

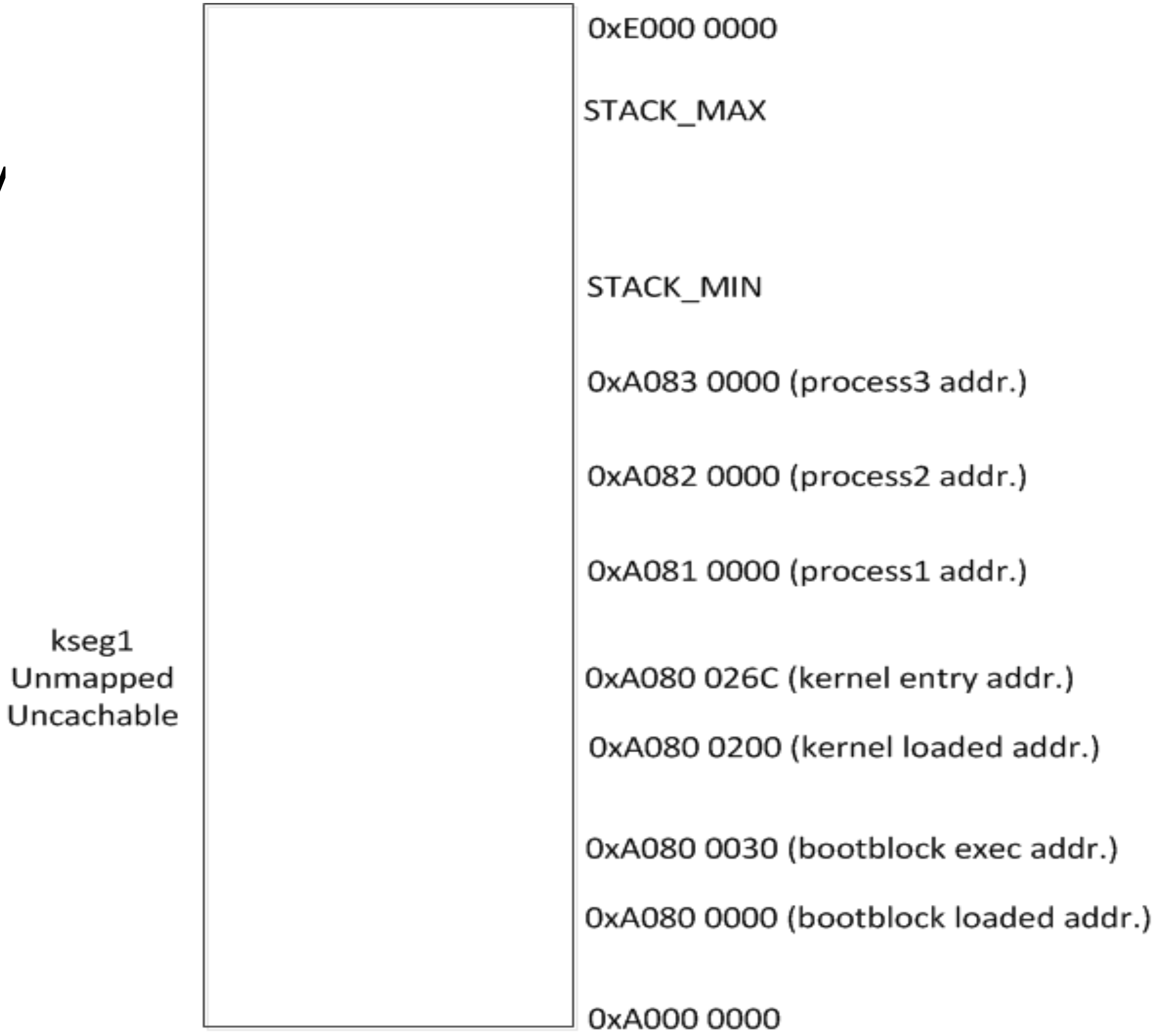
- A set of multiple tasks
  - Program codes under the *test* directory in start-code
  - Please refer to *test.c* for different groups of tasks
  - Fixed number of tasks for each test group
  - Allocate per-task state statically in main.c
  - STRONGLY suggest to first read the codes of different tasks to understand what they do

Project 2 – A Simple Kernel

- Process Control Block (PCB/TCB)
  - A data structure in OS kernel containing the information to manage a particular process/thread
  - Normally, kept in an area of memory protected from normal user access

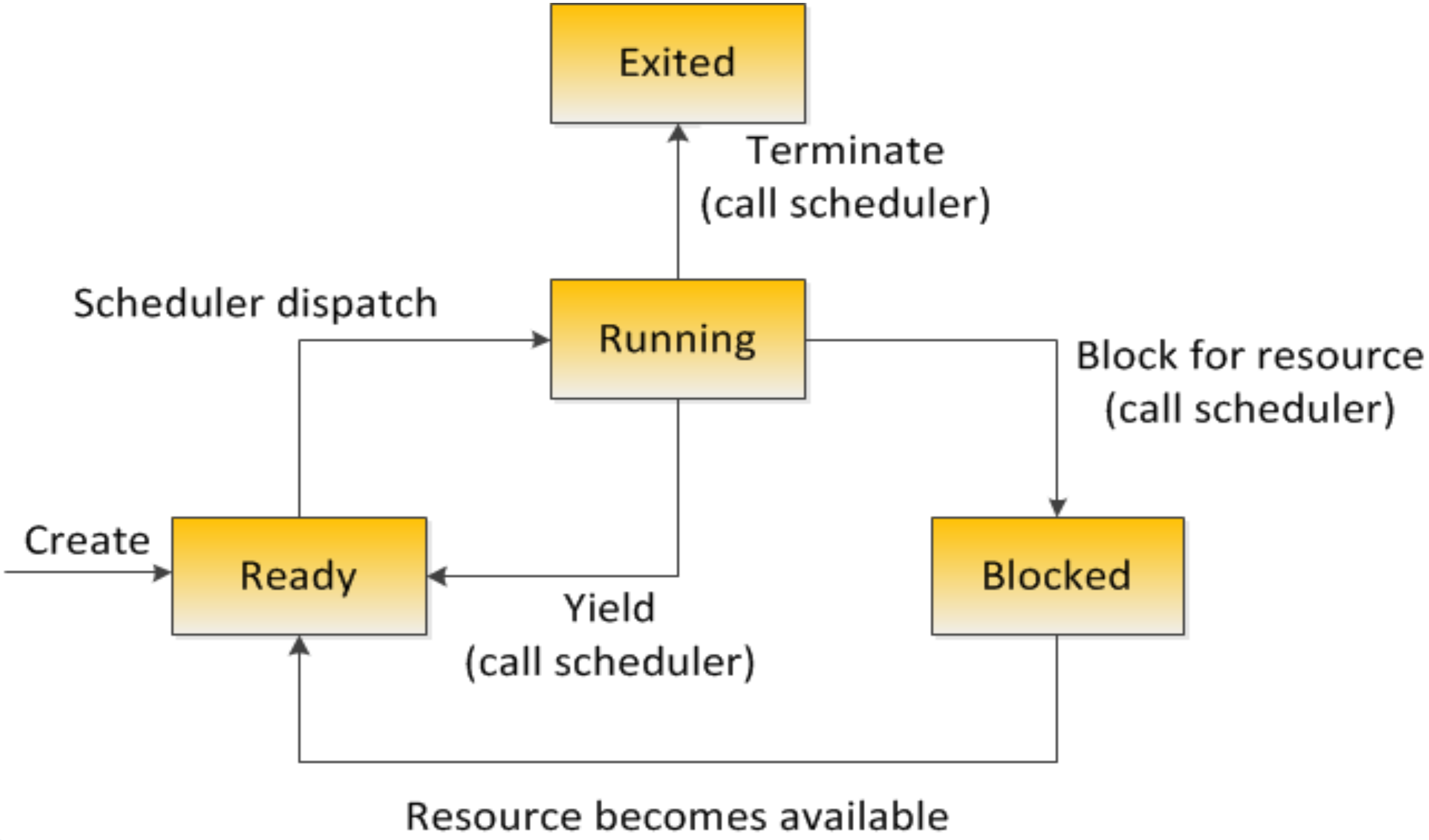
Project 2 – A Simple Kernel

- Start a task
  - Possible memory layout
  - Decide your own STACK_MIN and STACK_MAX


0xE000 0000
STACK_MAX
STACK_MIN
0xA083 0000 (process3 addr.)
0xA082 0000 (process2 addr.)
0xA081 0000 (process1 addr.)
kseg1
Unmapped
Uncachable
0xA080 026C (kernel entry addr.)
0xA080 0200 (kernel loaded addr.)
0xA080 0030 (bootblock exec addr.)
0xA080 0000 (bootblock loaded addr.)
0xA000 0000

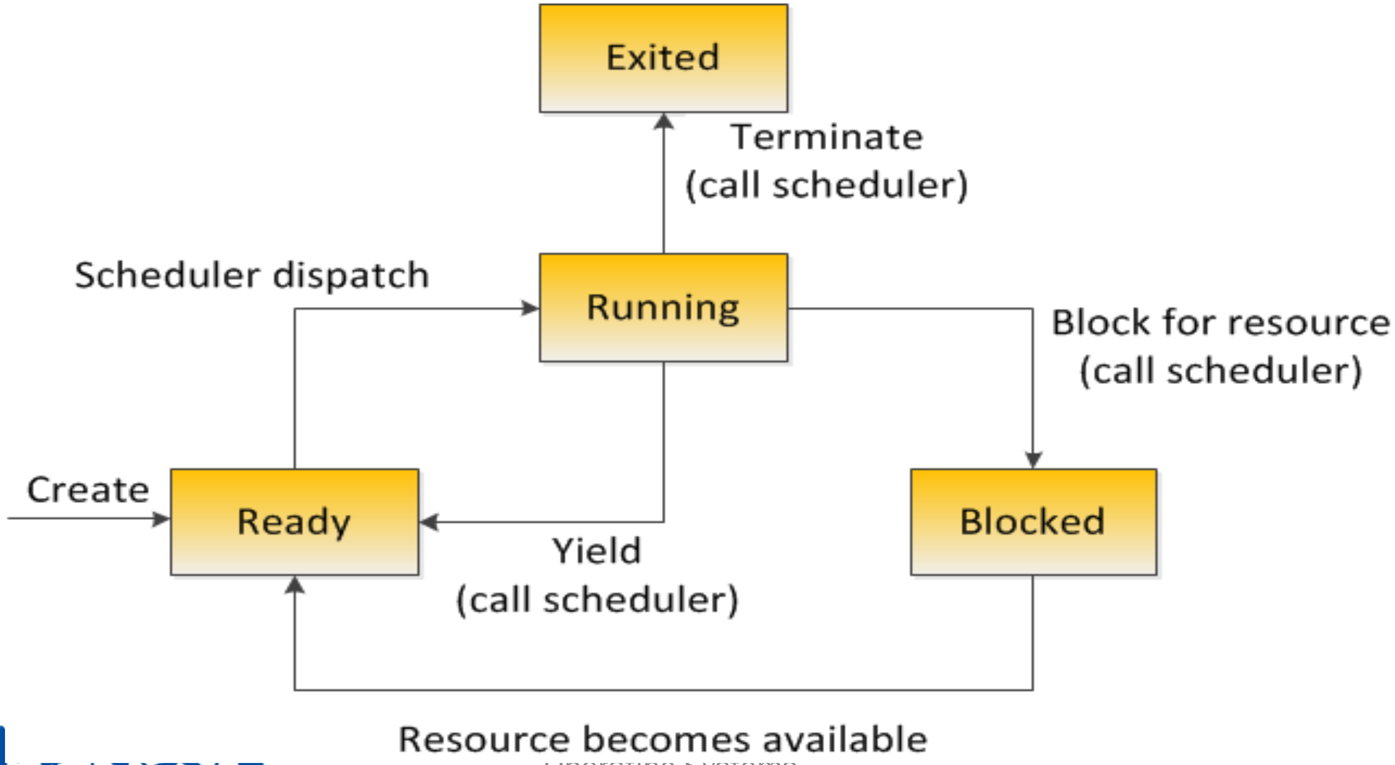
Project 2 – A Simple Kernel

- Scheduler (non-preemptive kernel)



Project 2 A Simple Kernel

- Problems of Non-Preemptive kernel
  - How about the throughput of the operating system?



Project 2 A Simple Kernel

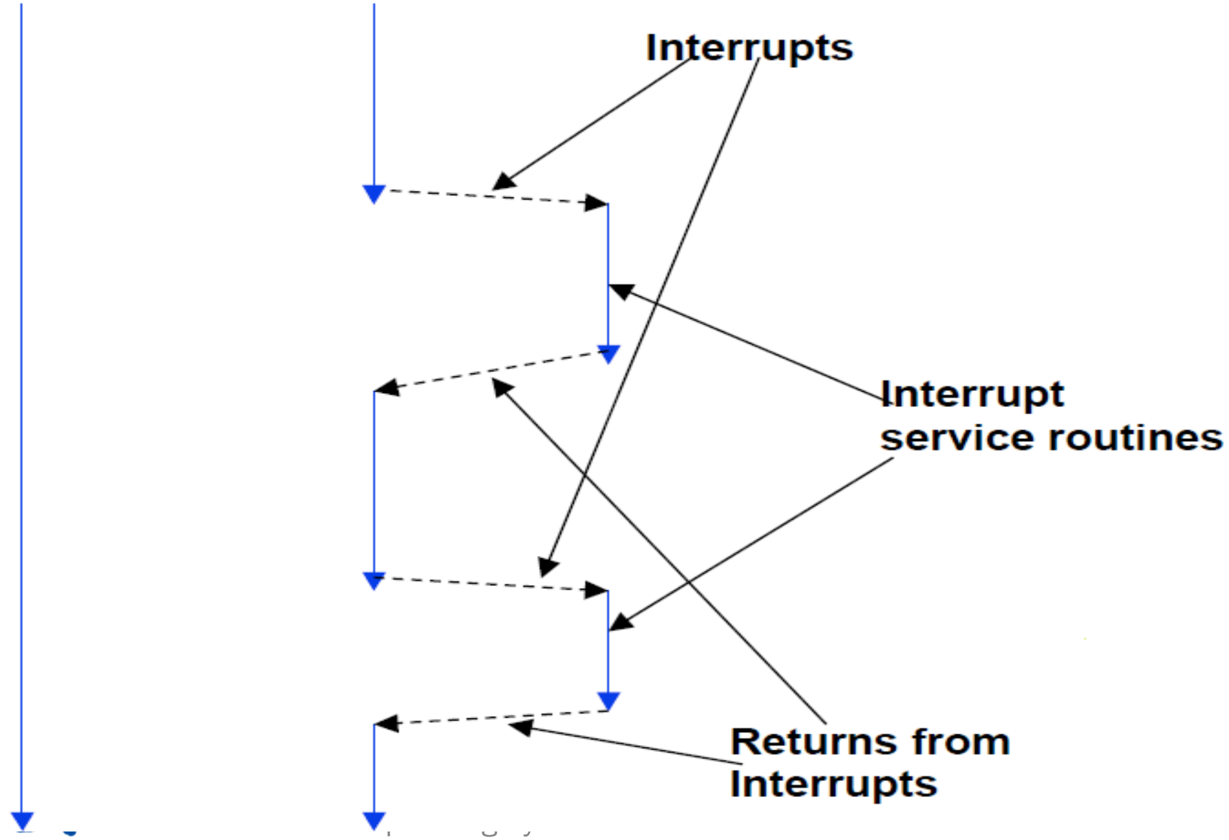
- Interrupt & Exception
  - A signal to the processor emitted by HW or SW indicating an event requiring immediate process
  - The processor responds by suspending its current running task, saving its state, and executing interrupt/exception handler
  - After the interrupt/exception handler finishes, the processor resumes normal activities

Project 2 A Simple Kernel

- Interrupt & Exception

Normal execution

Normal execution with interrupt



Interrupts

Interrupt service routines

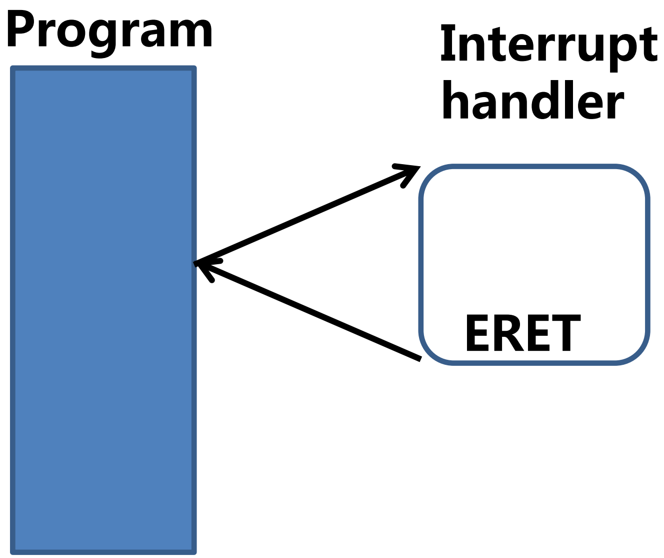
Returns from Interrupts

Project 2 A Simple Kernel

- Hardware interrupt
  - A change in execution caused by an external event outside the processor
    - Clock for timesharing
    - I/O devices: disk, network, keyboard etc.
- Software interrupt
  - System calls: a user-programmed interrupt
- We do not handle other exceptions here
  - Segment fault, Overflow, Page fault etc.

Project 2 A Simple Kernel

- Handling interrupt
  - Save context
  - Determine what caused interrupt
  - Invoke specific routine based on type of interrupt
  - Restore context
  - Return from interrupt

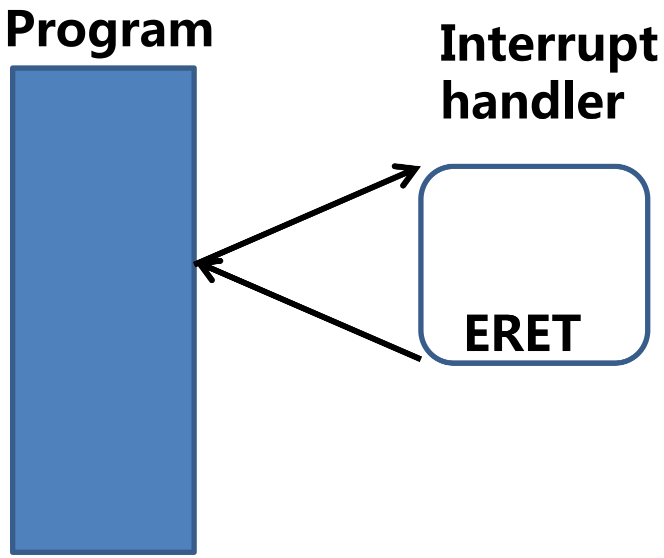


Project 2 A Simple Kernel

- Handling interrupt
  - What if interrupt occurs while in interrupt handler?
    - ENTER_CRITICAL
      - Disable interrupt before actually handling the interrupt
      - Setting a label to indicate the interrupt is disabled
    - LEAVE_CRITICAL
      - Do not forget to re-enable interrupt after handling the interrupt

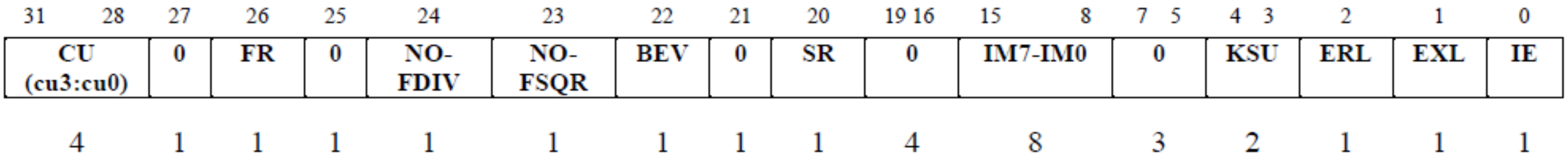
Project 2 A Simple Kernel

- Handling interrupt
  - ENTER_CRITICAL
  - Save context
  - Determine what caused interrupt
  - Invoke specific routine based on type of interrupt
  - LEAVE_CRITICAL
  - Restore context
  - Return from interrupt



Project 2 A Simple Kernel

- Tips on implementing interrupt handler
  - Coprocessor 0 (CP0) registers
    - Status register
    - IE bit: 1 enable interrupt; 0 disable
    - IM7 ~ IM0: control enable/disable different interrupts
    - IM7 bit: 1: enable Clock interrupt; 0: disable



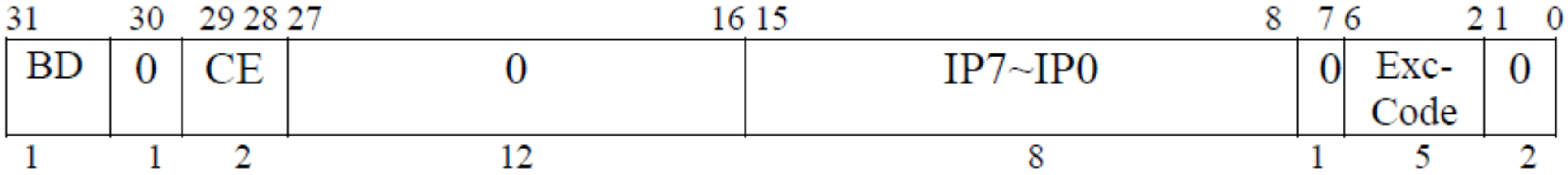
31 28	27	26	25	24	23	22	21	20	19 16	15 8	7 5	4 3	2	1	0
CU (cu3:cu0)	0	FR	0	NO- FDIV	NO- FSQR	BEV	0	SR	0	IM7-IM0	0	KSU	ERL	EXL	IE
4	1	1	1	1	1	1	1	1	4	8	3	2	1	1	1

Project 2 A Simple Kernel

- Tips on implementing interrupt handler
  - Coprocessor 0 (CP0) registers
    - Status register
    - Refer to the initialization of interrupts in start code
    - Pay attention to the initialization of the register in YOUR PCB
    - We handle clock interrupt and syscall in this project

Project 2 A Simple Kernel

- Tips on implementing interrupt handler
  - Coprocessor 0 (CP0) registers
    - Cause register
    - EXCCODE: Exception type
      - Hardware interrupt / software interrupt
    - IP7 ~ IP0: indicating the interrupt type
    - IP7 bit: 1: Clock interrupt



31	30	29 28	27 16	15 8	7	6 2	1 0
BD	0	CE	0	IP7~IP0	0	Exc-Code	0
1	1	2	12	8	1	5	2

Project 2 A Simple Kernel

- Tips on implementing interrupt handler
  - Operate CP0 registers
    - mfc0: move from Coprocessor 0
      - Loads data from a CP0 register into a CPU register
    - mtc0: move to Coprocessor 0
      - Stores data into a CP0 register
  - Use MIPS registers
    - $k0, $k1

Project 2 A Simple Kernel

- Tips on implementing interrupt handler
  - Coprocessor 0 (CP0): read-modify-write
    - Read the cp0 register into a CPU register
    - Modify the contents of the CPU register
    - Write the modified value back to the cp0 register
    - e.g.
      ```
      mfc0    k0, CP0_STATUS
      mtc0    k0, CP0_STATUS
      ```

Project 2 A Simple Kernel

- Tips on implementing interrupt handler
  - What do you do in the clock interrupt handler?
    - How to deal with normal tasks?
      - Schedule based on your scheduling policy
    - How to deal with sleeping tasks?
      - Check whether waking up the task

Project 2 A Simple Kernel

- Process sleep()
  - Blocking sleep
    - Block the task when it calls sleep()
    - Use a separate queue to keep sleeping tasks
  - Wake up the task
    - When the timing reaches sleeping threshold of the task
  - About timing
    - A global counter recording the number of ticks
    - The counter increases in each clock interrupt
    - We provide wall time calculation functions in time.c

Project 2 A Simple Kernel

- Scheduler
  - Round robin
  - Priority based
  - Fairness
  - Think about what kind of information should be included in PCB if you want to do the above scheduler

Project 2 A Simple Kernel

- Step by step
  - Task 1: implement a clock interrupt handler and priority based scheduler
  - Task 2: support syscalls and implement sys_sleep

Project 2 A Simple Kernel

- Requirement for design review (40 points, cont.)
  - What is the workflow for handling interrupt (for both clock interrupt and syscalls)?
  - How do you implement the priority based scheduler?
  - When do you wake up the sleeping task?

Project 2 – A Simple Kernel

- Requirement for developing (60 points)
  - Implement clock interrupt handler and syscall handler 40
  - Implement blocking sleep 10
  - Implement a priority-based scheduler 10

Project 2 – A Simple Kernel

- Bonus (1 point)
  - Support one thread to acquire multiple locks (at least two)
  - Support more than two threads to acquire a single lock
  - Design your own test cases and show the results

Project 2 – A Simple Kernel

- P2 schedule
  - 23rd Sep.
    - P2 part I design review:
    - P2 part II assignment
  - 30th Sep.
    - P2 part I due
    - P2 part II design review
  - 9th Oct. (optional)
    - P2 part II design review
  - 14th Oct.
    - P2 part II due