操作系统研讨课

蒋德钧

Fall Term 2018-2019

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

University of Chinese Academy of Sciences

Lecture 2 A Simple Kernel

2018.09.26

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

Schedule

- Project 1 due
- Project 2 assignment

Project 1 Due

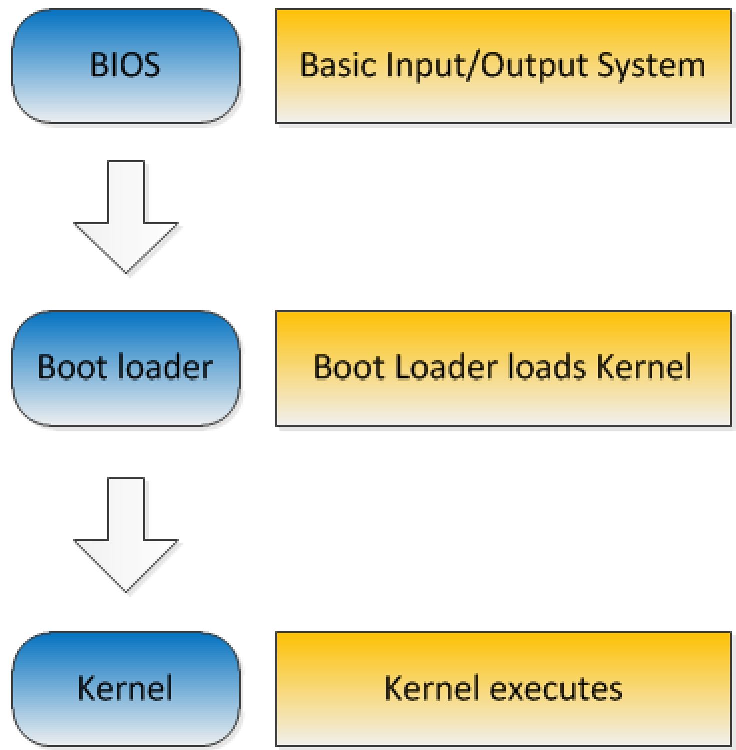
- P1 due
  - We test Tasks 2 and 3 for P1 due
  - Please compile your code, running the code on your board, and show the results to TA
  - Answer any questions we may ask

Project 1 Due

- P1 submission
  - Submit a compressed package named as "StudentNo.-YourName-P1"
  - Please includes
    - Source code
    - README to simply describe your code, e.g. which file is your work or how to run your code
    - Design document covering the questions in the design document template
  - Do not forget to submit before 23:55 TONIGHT

Project 2 – A Simple Kernel

- Booting procedure



Project 2 – A Simple Kernel

- Requirements
  - 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 (Cont.)
  - 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

- Process Control Block (PCB/TCB)
  - Process status
    - Status of a process when it is suspended
    - Contents of registers, stack pointers etc.
  - Process scheduling info
    - e.g. priority

Project 2 – A Simple Kernel

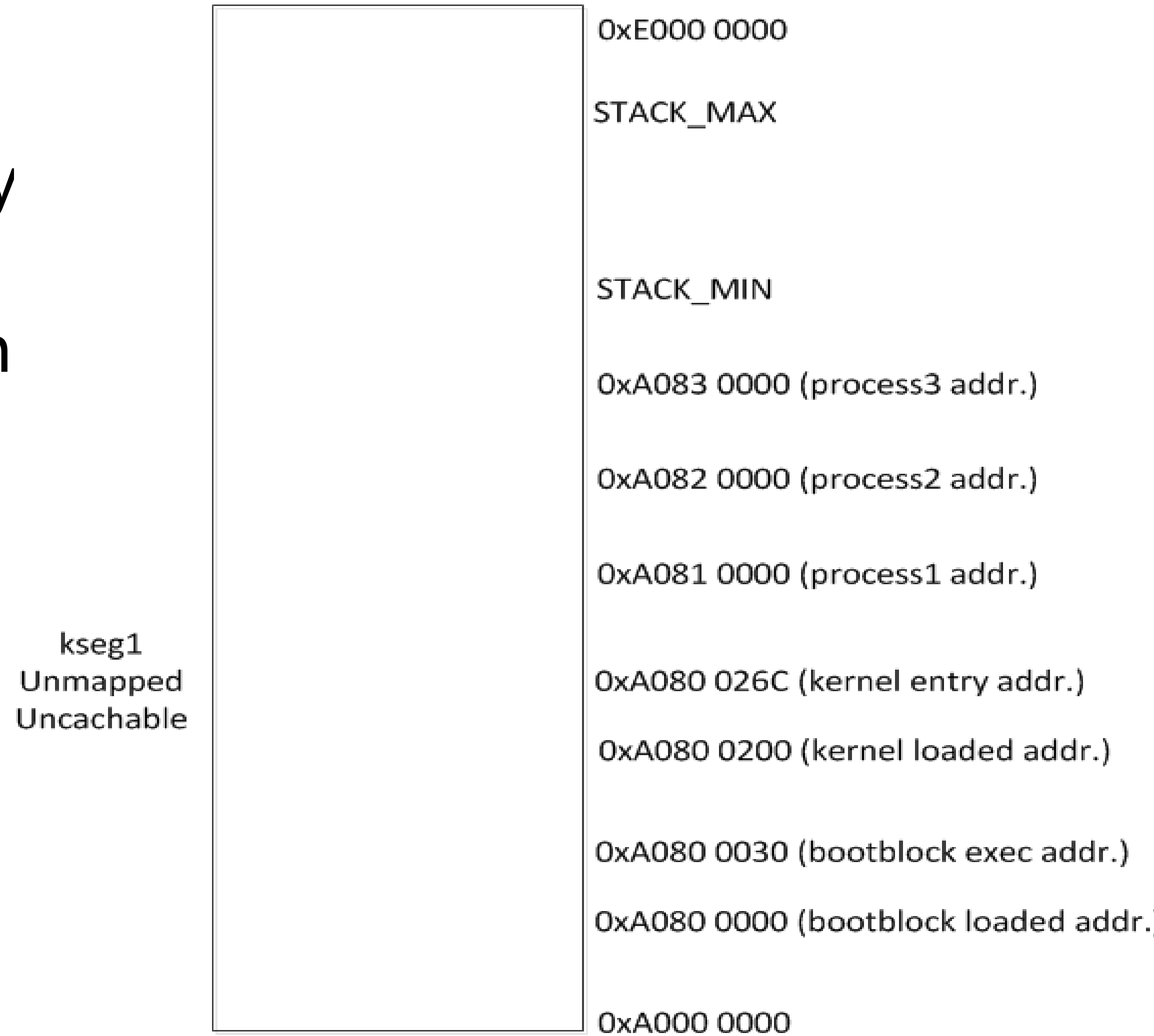
- Start a task(process/thread)
  - Task type
    - User Process
      - Use faked SYSCALL to call kernel functions
      - But, in this project compiled with the kernel and share the same address with the kernel
    - Kernel thread
  - Task entry point
    - Function addresses
  - Please refer to *task_info* structure in start-code

Project 2 – A Simple Kernel

- Start a task (process/thread)
  - Each task is associated with a PCB
  - Initialize PCB
    - Which registers should be set?
    - Where is the task located?
    - How to setup stack? Stack size?
    - Where is the PCB located?

Project 2 – A Simple Kernel

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



Project 2 – A Simple Kernel

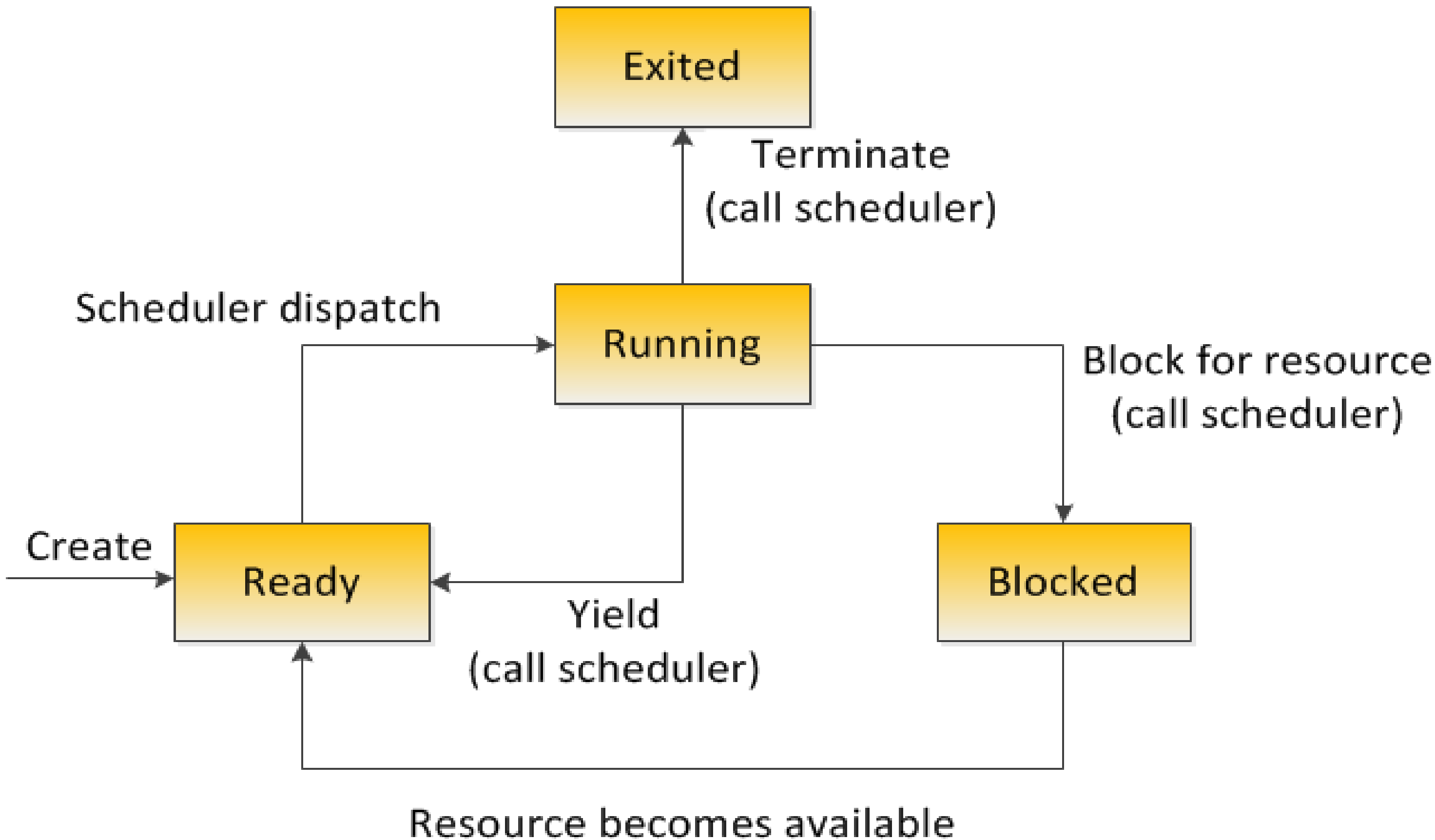
- Start a task
  - Scheduler: single task vs. multi-tasks
    - How to organize tasks being scheduled?
    - How to select the next task?
      - FIFO

Project 2 – A Simple Kernel

- Start a task
  - Scheduler: first task vs. the following ones
    - do_scheduler()
      - Locate the PCB of the first task
      - Restore PCB
    - When the scheduler is invoked?
      - Non-preemptive kernel
      - Preemptive kernel

Project 2 – A Simple Kernel

- Scheduler (non-preemptive kernel)



Project 2 – A Simple Kernel

- Yield
  - An action to force a processor to release control of the current running thread
  - Place the current running thread to the end of the running queue
  - In this project, we call do_scheduler() to execute yield

Project 2 – A Simple Kernel

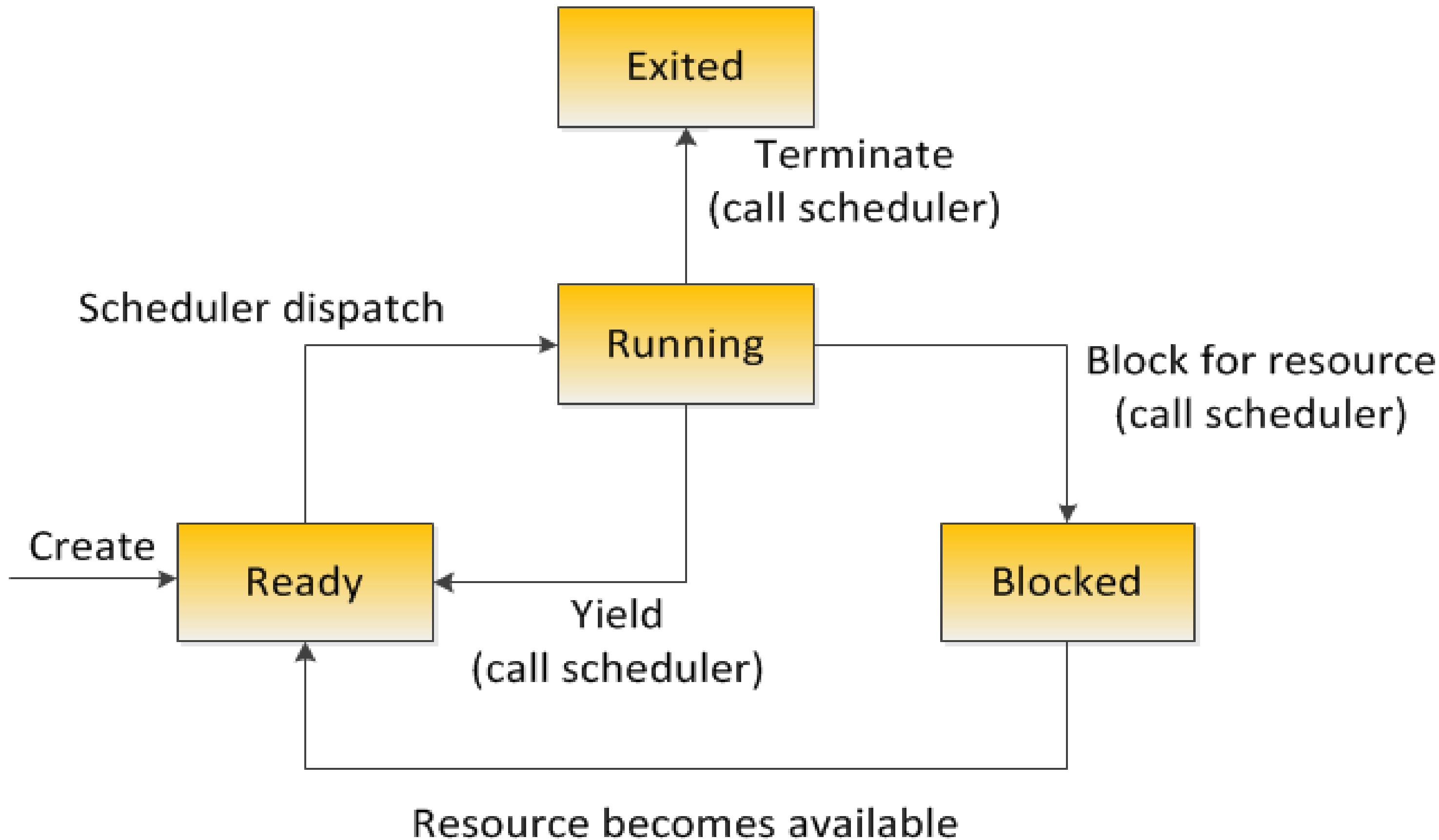
- Context switch
  - Save PCB
    - What kind of things to save
    - Registers → Memory
  - Restore PCB
    - Memory → Registers
    - do_scheduler()

Project 2 – A Simple Kernel

- Create machine image
  - Combine binary of kernel, tasks together into a machine image using createimage in your P1

Project 2 – A Simple Kernel

- Mutex lock

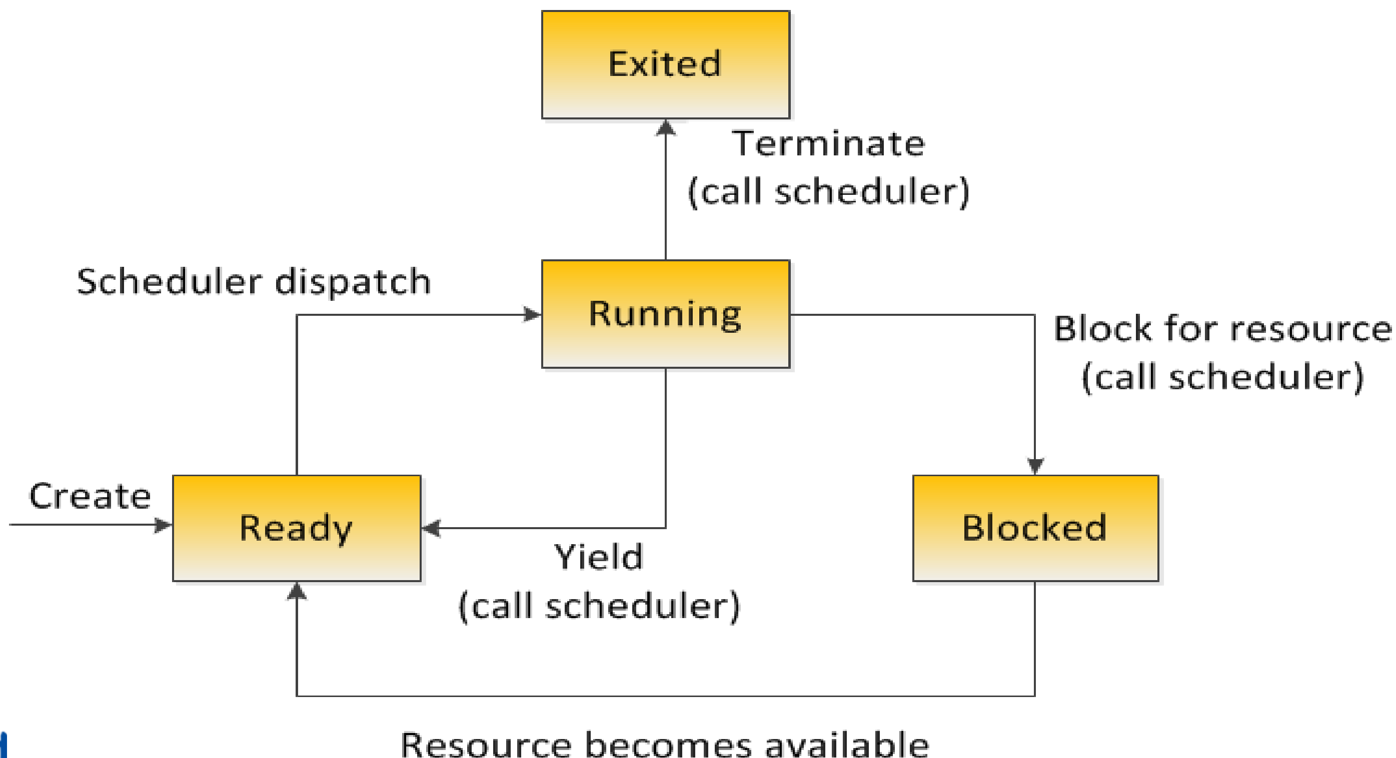


Project 2 – A Simple Kernel

- Mutex lock
  - What if no thread currently holds the lock?
  - What if the lock is currently held?
  - Implement lock-related functions
  - Manage tasks that do not acquire the lock
    - Same queue vs. different queues?

Project 2 A Simple Kernel

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

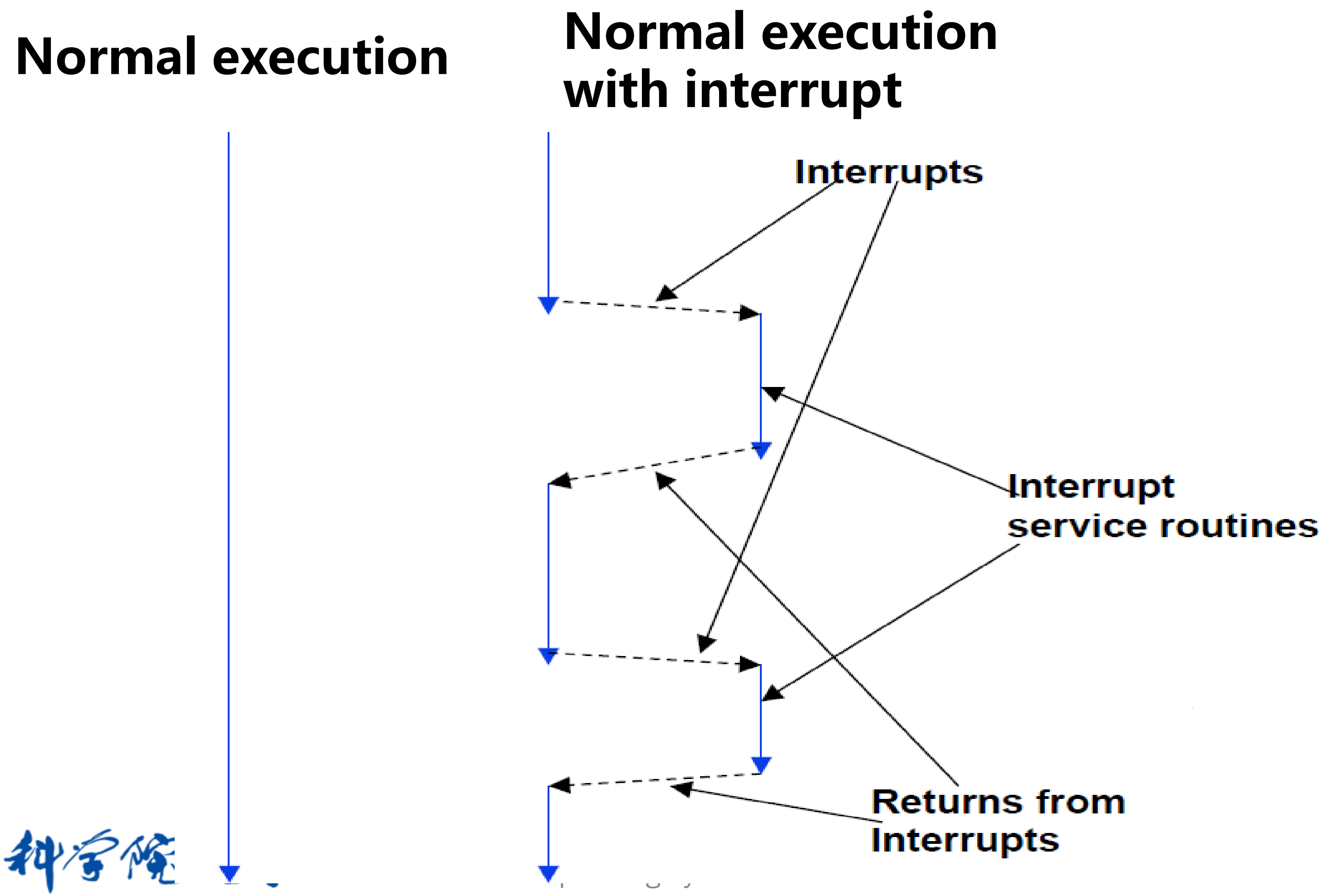


Project 2 A Simple Kernel

- 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

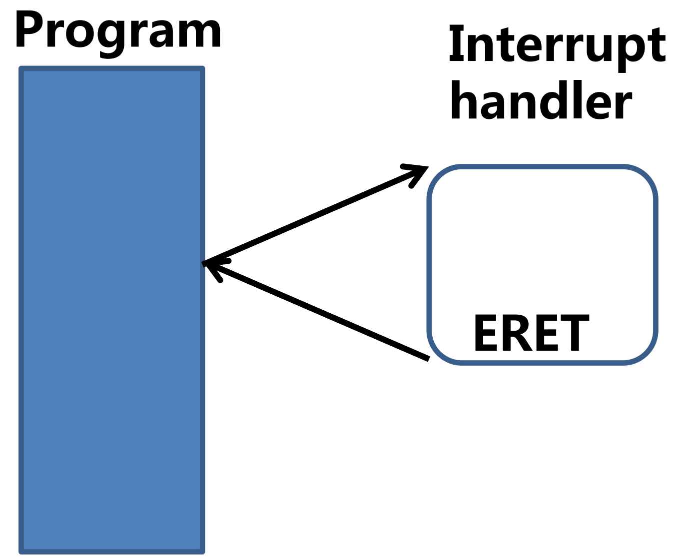


Project 2 A Simple Kernel

- 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

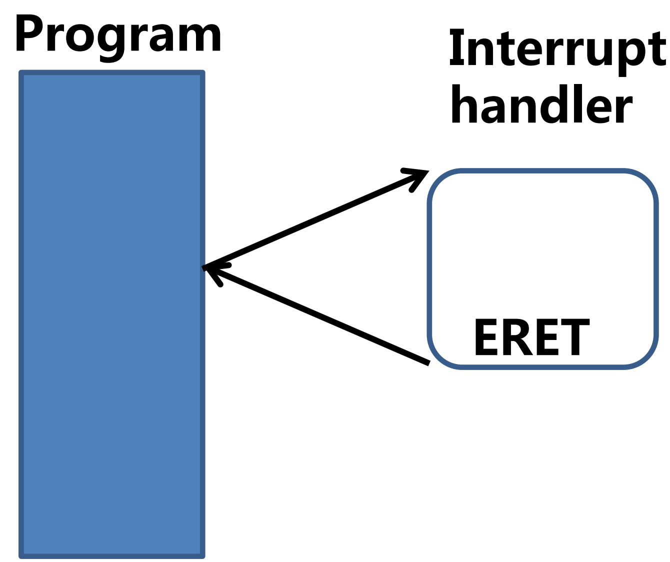


Project 2 A Simple Kernel

- 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?
    - How to deal with sleeping tasks?

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 0: PLEASE read the guiding book and start code CAREFULLY
  - Task 1: start a set of kernel threads and support context switch as a non-preemptive kernel
  - Task 2: implement mutex lock to support BLOCK state

Project 2 A Simple Kernel

- Step by step
  - Task 3: implement a clock interrupt handler and priority based scheduler
  - Task 4: support syscalls and implement sys_sleep

Project 2 A Simple Kernel

- Requirement for design review (40 points)
  - What is PCB? What are included in PCB?
  - What need to be done for initializing tasks?
  - When is context switching in this project? What need to be done for context switching?
  - How do you manage blocked tasks?

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)
  - Start tasks and set PCBs: 5
  - Execute context switch without errors: 20
  - Implement the mutex lock: 5
  - Implement clock interrupt handler and syscall handler 20
  - Implement blocking sleep 5
  - Implement a priority-based scheduler 5

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
  - P2 design review: 10th Oct.
  - No class on 17th Oct.
  - P2 due: 24th Oct.
- Final reminder
  - 重要的事情说三遍
  - Start early
  - Start early
  - Start early