操作系统研讨课
Course: B0911011Y

陈明宇、蒋德钧

Fall Term 2021-2022

email: cmy@ict.ac.cn / 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)

2021.10.04

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

Schedule

- Project 2 part I due
- Project 2 part II assignment

Project 2 – A Simple Kernel

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

Project 2 – A Simple Kernel

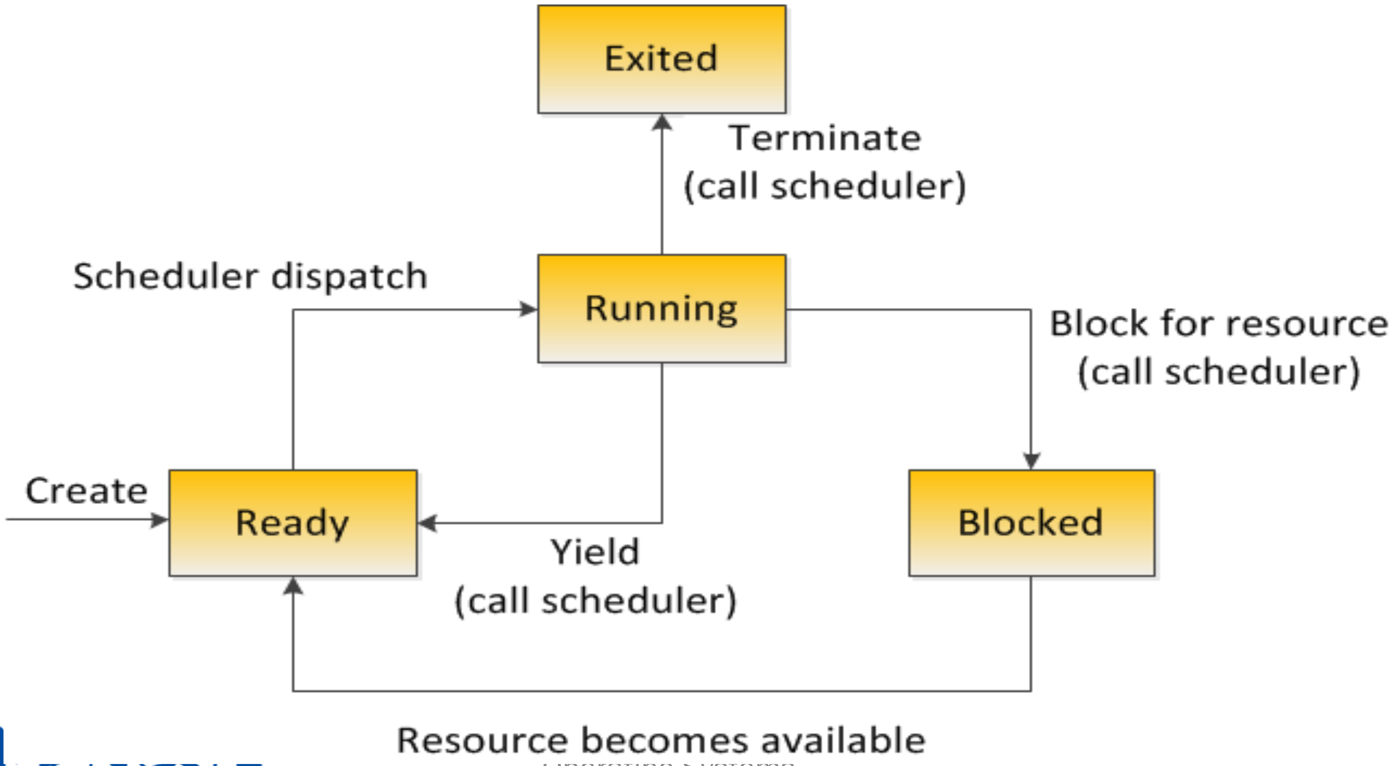
- Requirements (Part II)
  - Write a simple kernel (preemptive)
    - Provide preemptive kernel supporting exception handler, including
      - System call handler
      - Clock interrupt handler
    - Round-robin scheduler
    - fork and priority-based scheduler (optional for C-Core)

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
    - sched2_tasks、timer_tasks、lock2_tasks
  - 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

- Non-Preemptive kernel
  - A task runs until it exits/yields or is blocked
  - Other tasks need to wait for the running task

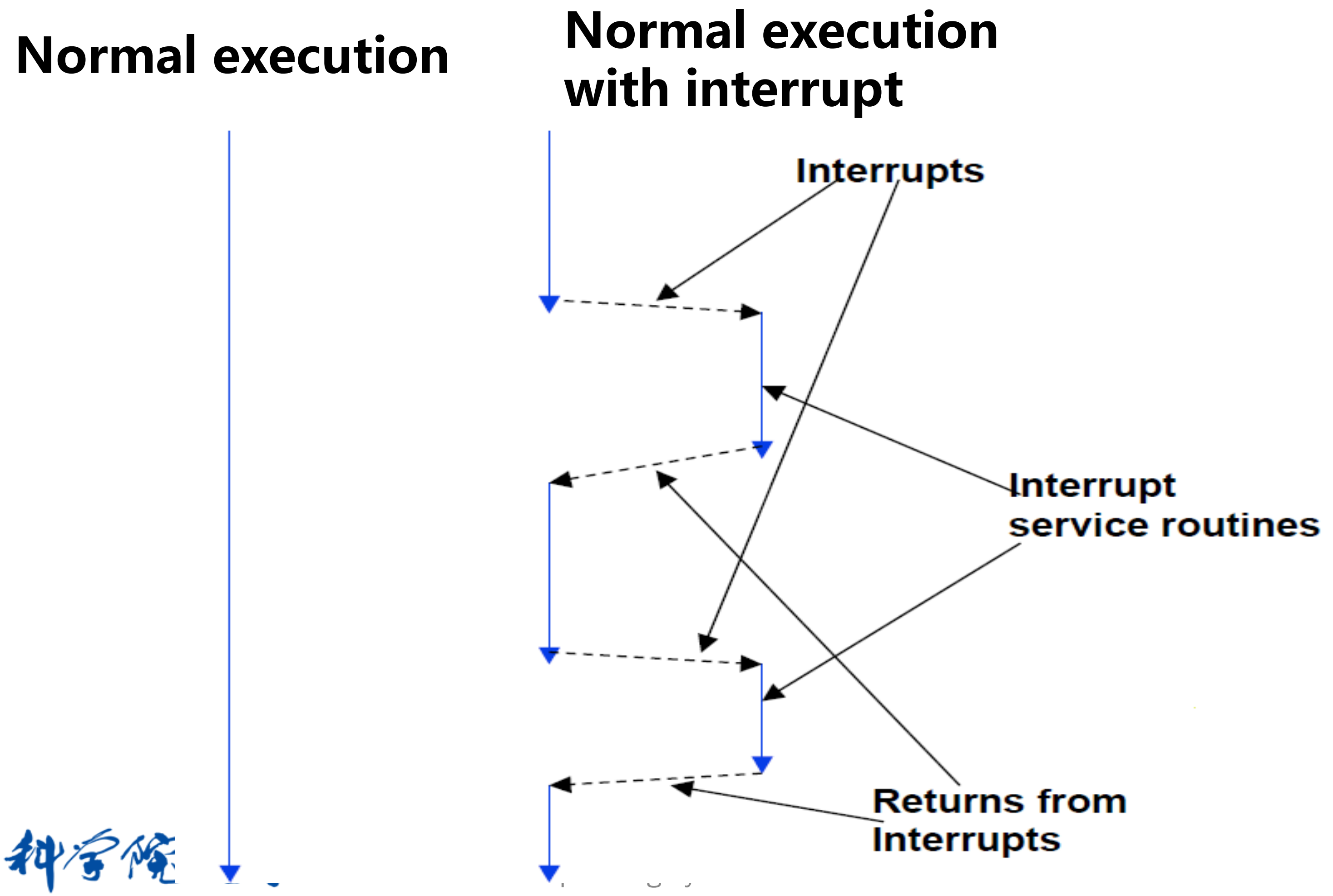


Project 2 A Simple Kernel

- Interrupt
  - 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 handler
  - After the interrupt handler finishes, the processor resumes normal activities

Project 2 A Simple Kernel

- Interrupt

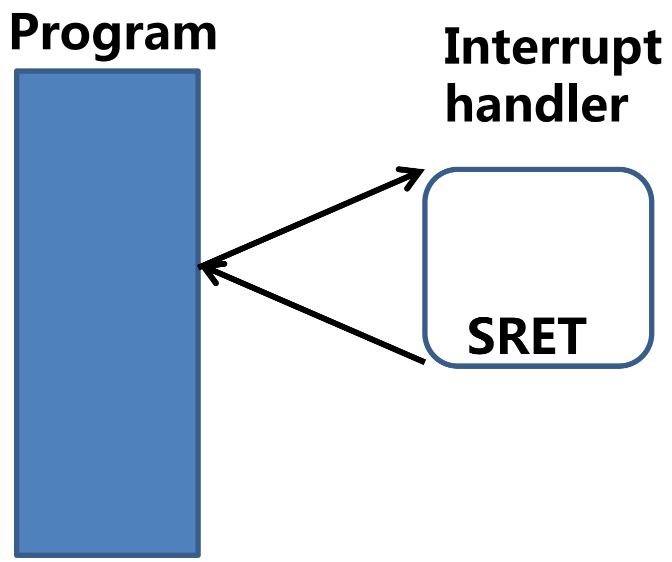


Project 2 A Simple Kernel

- Programmed interrupt
  - System calls: a user-programmed interrupt
- Hardware interrupt
  - A change in execution caused by a hardware event
    - Clock interrupt for timesharing
    - I/O device interrupt: disk, network, keyboard etc.
- We do not handle other interrupts here
  - Page fault, TLB miss etc.

Project 2 A Simple Kernel

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



Project 2 A Simple Kernel

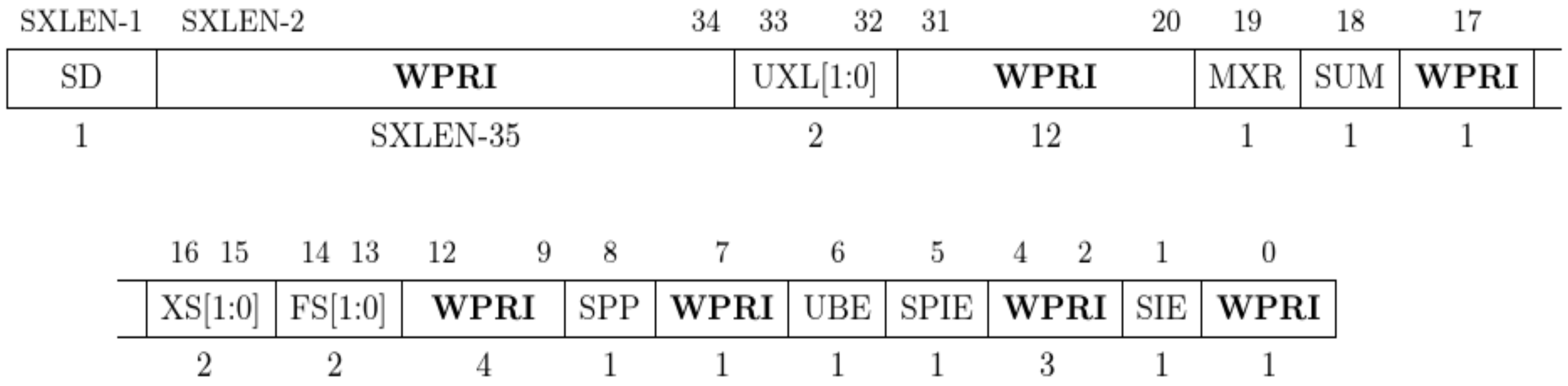
- Tips on implementing interrupt handler
  - What if interrupt occurs while in interrupt handler?
    - The processor automatically disables all interrupts when an interrupt occurs
    - The processor automatically re-enables all interrupts after sret is called

Project 2 A Simple Kernel

- Tips on implementing interrupt handler
  - Control Status Registers(CSRs)
    - We need to manipulate CSRs when implementing interrupt handler
  - Two types of CSRs
    - One type is for setting up interrupt
    - The other is for handling interrupt
    - Pls. refer to Table P2-3 in guidebook

Project 2 A Simple Kernel

- Tips on implementing interrupt handler
  - Enable/disable interrupt
    - sstatus register



SXLEN-1	SXLEN-2 34	33 32	31 20	19	18	17
SD	**WPRI**	UXL[1:0]	**WPRI**	MXR	SUM	**WPRI**
1	SXLEN-35	2	12	1	1	1

16 15	14 13	12 9	8	7	6	5	4 2	1	0
XS[1:0]	FS[1:0]	**WPRI**	SPP	**WPRI**	UBE	SPIE	**WPRI**	SIE	**WPRI**
2	2	4	1	1	1	1	3	1	1

Project 2 A Simple Kernel

- Tips on implementing interrupt handler
  - Enable/disable interrupt
    - SIE bit enables or disables all interrupts in supervisor mode
    - SPIE bit indicates whether supervisor interrupts were enabled prior to trapping into supervisor mode.
    - You need to correctly initialize SPIE in sstatus register

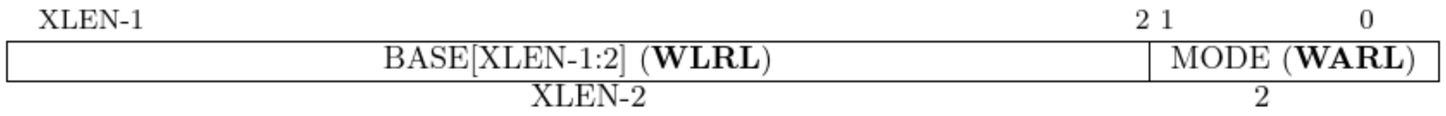
Project 2 A Simple Kernel

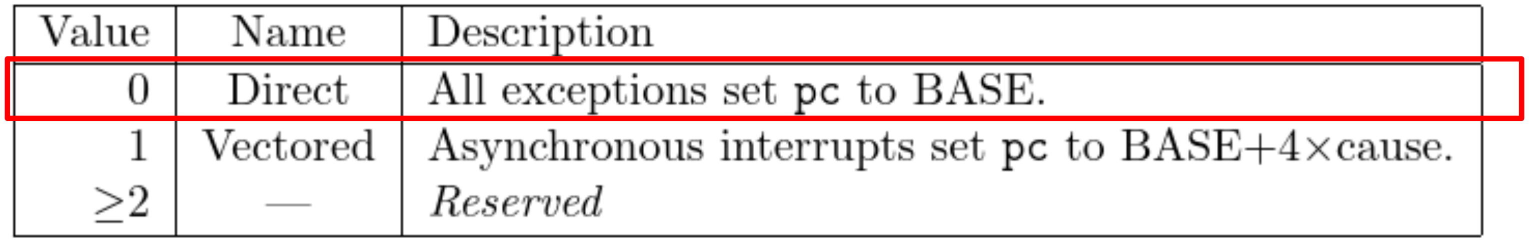
- Tips on implementing interrupt handler
  - Enable interrupt
    - sie register
    - STIE(Supervisor-level timer interupt enable)
    - Set STIE when setting up clock interrupt

SXLEN-1 10	9	8 6	5	4 2	1	0
WPRI	SEIE	**WPRI**	STIE	**WPRI**	SSIE	**WPRI**
SXLEN-10	1	3	1	3	1	1

Project 2 A Simple Kernel

- Tips on implementing interrupt handler
  - Entry to handle interrupt
    - stvec register
    - BASE field hold the interrupt handler’ s address
    - Pls. refer to exception_handler_entry in entry.S

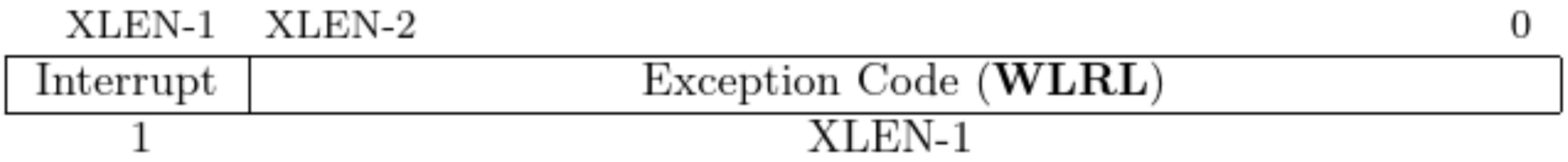




Value	Name	Description
0	Direct	All exceptions set pc to BASE.
1	Vectored	Asynchronous interrupts set pc to BASE+4×cause.
≥2	—	*Reserved*

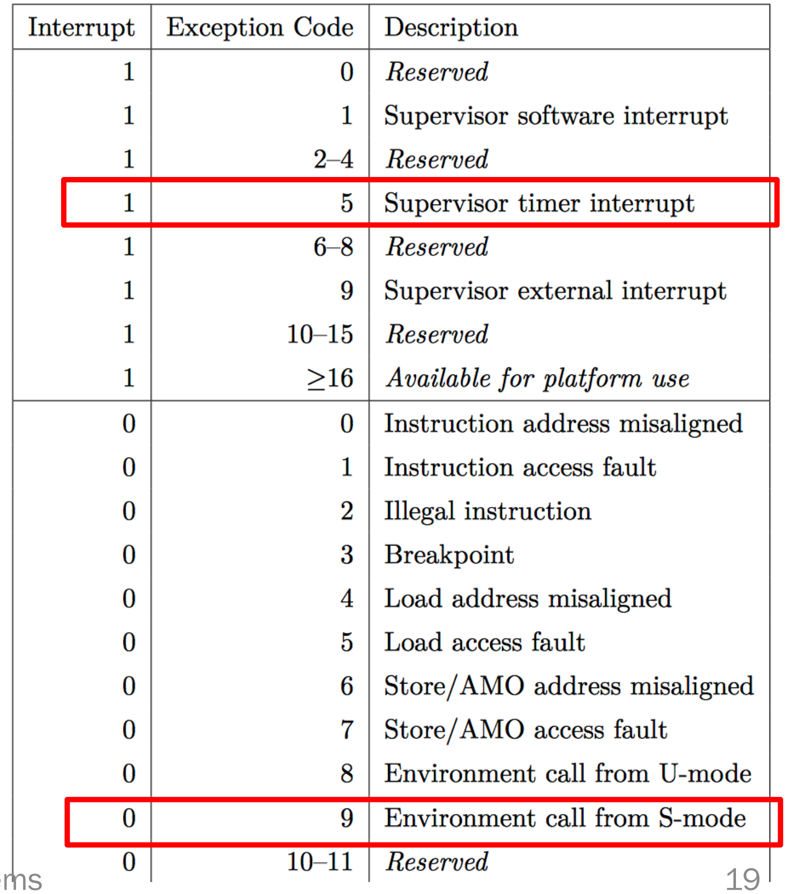
Project 2 A Simple Kernel

- Tips on implementing interrupt handler
  - Identifying interrupt type
    - scause register



Project 2 A Simple Kernel

- Tips on implementing interrupt handler
  - Identifying interrupt type
  - pls. refer to Table P2-4



Interrupt	Exception Code	Description
1	0	*Reserved*
1	1	Supervisor software interrupt
1	2–4	*Reserved*
1	5	Supervisor timer interrupt
1	6–8	*Reserved*
1	9	Supervisor external interrupt
1	10–15	*Reserved*
1	≥16	*Available for platform use*
0	0	Instruction address misaligned
0	1	Instruction access fault
0	2	Illegal instruction
0	3	Breakpoint
0	4	Load address misaligned
0	5	Load access fault
0	6	Store/AMO address misaligned
0	7	Store/AMO access fault
0	8	Environment call from U-mode
0	9	Environment call from S-mode
0	10–11	*Reserved*

Project 2 A Simple Kernel

- Tips on implementing interrupt handler
  - Manipulate CSR registers
    - csrr: read csr
      - Loads data from a CSR register into a CPU register
    - csrw: write csr
      - Stores data into a CSR register
    - csrc/csrs：clear/set a CSR register’s corresponding bits
  - Examples
    - csrr a0, sip
    - csrw stvec, t0

Project 2 A Simple Kernel

- Syscall to support 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

- 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
    - Reset timer

Project 2 A Simple Kernel

- Scheduler
  - Round robin
  - Priority based
  - 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 3 in guidebook
    - Implement syscalls
    - Support do_sleep
    - Modify sys_yield, printf, and mutex-related functions to syscalls

Project 2 A Simple Kernel

- Step by step
  - Task 4 in guidebook
    - Implement clock interrupt handler and round robin based scheduler

Project 2 A Simple Kernel

- Requirement for design review
  - How do you handle clock interrupt and syscalls? Pls. list things you will do to handle clock interrupt, or possible functions you will implement
  - What are user-level and kernel-level stacks used for? How are they used during handling interrupt、syscalls and task scheduling?
  - When do you wake up the sleeping task?
  - Any questions you are not clear enough

Project 2 – A Simple Kernel

- Requirement for developing
  - S-Core
    - Implement syscalls, but syscall handler only needs to print error information, including
    - PC address where exception occurs (sepc)
    - bad address (stval)
    - exception causes (scause)

Project 2 – A Simple Kernel

- Requirement for developing
  - A-Core
    - Implement syscall handler, supporting blocking sleep, printf, sys_yield, and mutex-related syscalls
    - Implement clock interrupt handler

Project 2 – A Simple Kernel

- Requirement for developing
  - C-Core
    - All functions of A-Core
    - fork syscall, pls. refer to task 5 in guidebook
    - priori syscall and priority-based scheduler

Project 2 – A Simple Kernel

- P2 schedule
  - 4th Oct.
    - P2 part I due
    - P2 part II assignment
  - 11th Oct.
    - P2 part II design review
  - 18th Oct.
    - Hopefully P2 part II due