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 4 Virtual Memory

2019.11.18

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

Schedule

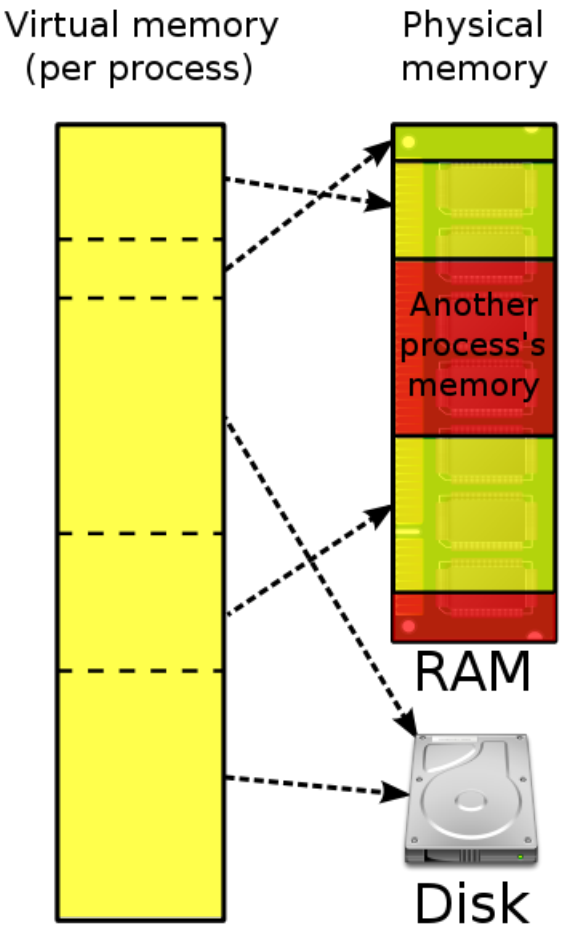
- Project 4 assignment
- Project 3 due

Project 4 Virtual Memory

- Requirement
  - Implement virtual memory management for user process
    - Setup page table and TLB entries
    - Handle TLB exception to support TLB refill and invalid TLB exceptions
    - Handle page fault to support on demand paging assuming physical memory is enough

Project 4 Virtual Memory

- Virtual memory
  - Each process sees a contiguous and linear address space, which is called virtual addresses
  - Virtual addresses are mapped into physical addresses by both HW and SW



[From Wikipedia]

Project 4 Virtual Memory

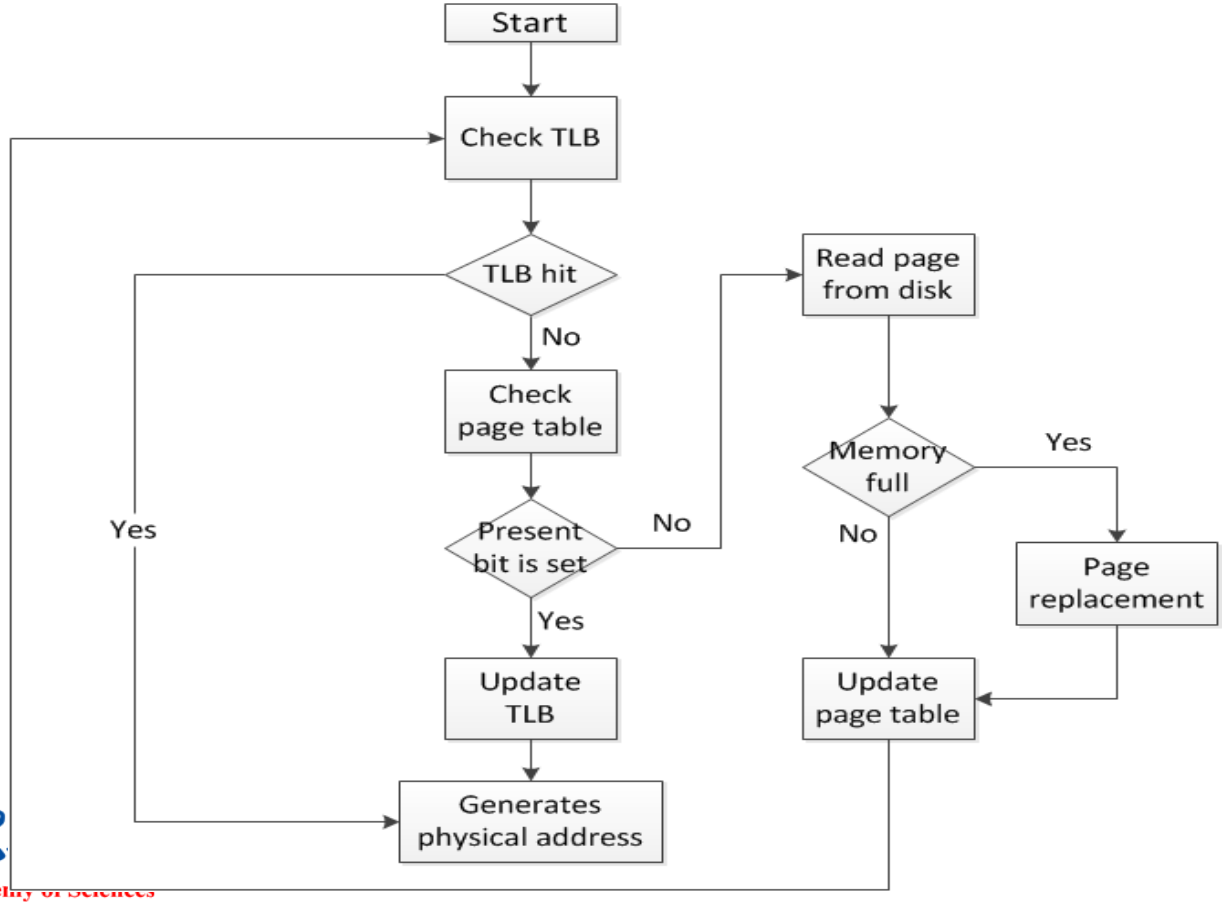
- Virtual memory
  - Virtual address space is divided into pages, which are blocks of contiguous virtual memory addresses
    - e.g. 4KB pages
  - Each page has a virtual address

Project 4 Virtual Memory

- Virtual memory
  - Page tables
    - The data structure to store the mapping between virtual addresses and physical addresses
    - Each mapping is a page table entry
  - MMU and TLB
    - MMU stores a cache of recently used mappings from the page table, which is called translation lookaside buffer (TLB)

Project 4 Virtual Memory

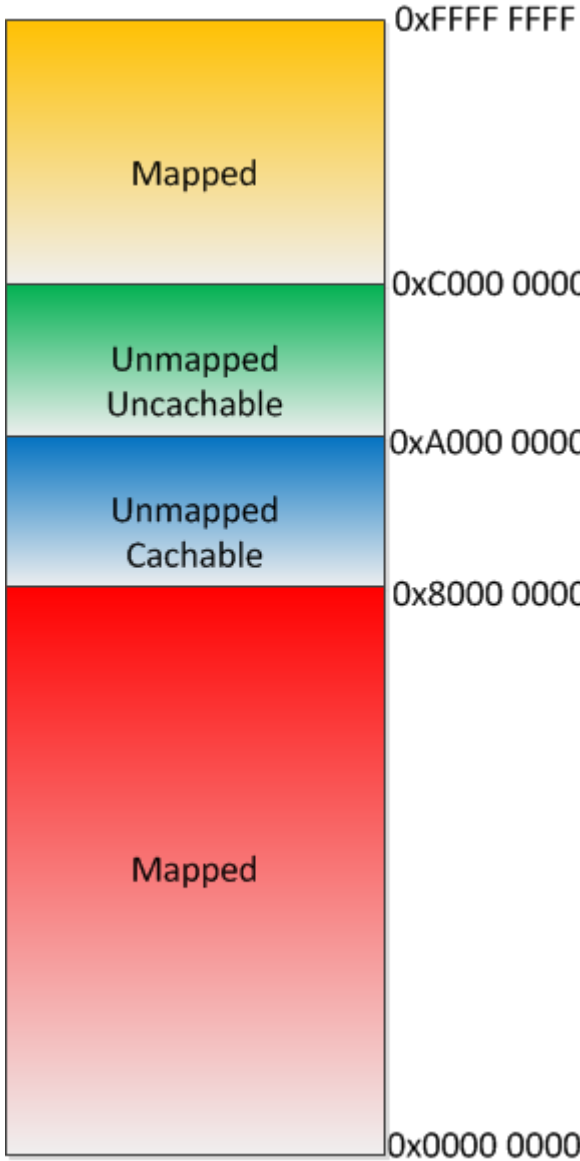
- Virtual memory
  - Address translation



Project 4 Virtual Memory

- Implementing virtual memory
  - MIPS virtual memory layout

Address range	Capacity (GB)	Mapping approach	Cacheable
0xFFFFFFFF -- 0xC0000000	1	TLB lookup	Yes
0xBFFFFFFF -- 0xA0000000	0.5	Base + offset	No
0x9FFFFFFF -- 0x80000000	0.5	Base + offset	Yes
0x7FFFFFFF -- 0x00000000	2	TLB lookup	Yes



Project 4 Virtual Memory

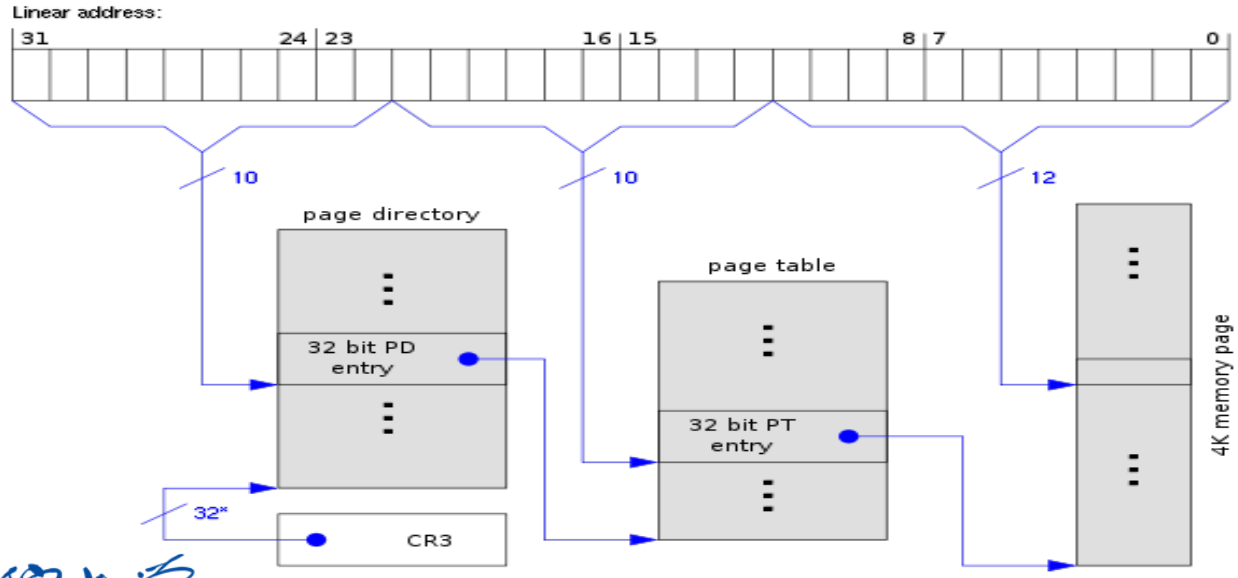
- Considering physical page frames
  - The size of physical page frames
    - Normally 4KB
    - You are free to use large page frame, e.g. 64KB,
  - The total number of page frames
    - 0x00000000 ~ 0x7FFFFFFF
    - 0xC0000000 ~ 0xFFFFFFFF

Project 4 Virtual Memory

- Page table setup
  - How many page frames for page table of the tested process?
    - Set a fixed sized page table
    - Note that: we provide a test process which requires you to input a random virtual address.
    - Please consider the page table setup for this situation. E.g. in task 1, the page tale should map all input virtual addresses

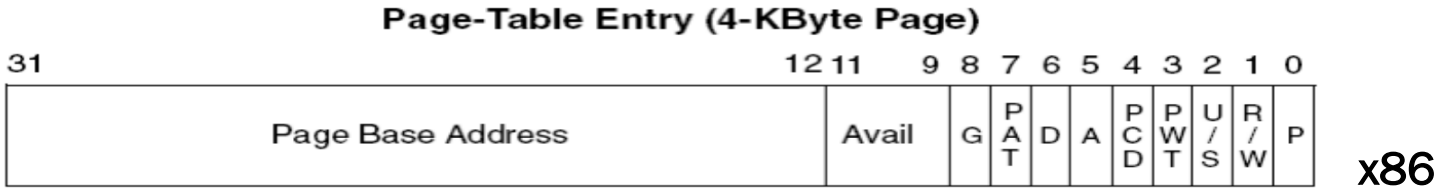
Project 4 Virtual Memory

- Page table setup
  - Allocate page frames for page table itself
    - Where to place the page table?
    - You are free to build single-level or two-level page table



Project 4 Virtual Memory

- Page table setup
  - Design the structure of page table entries (PTE)
    - Virtual address, physical address, valid, dirty
    - Any more?

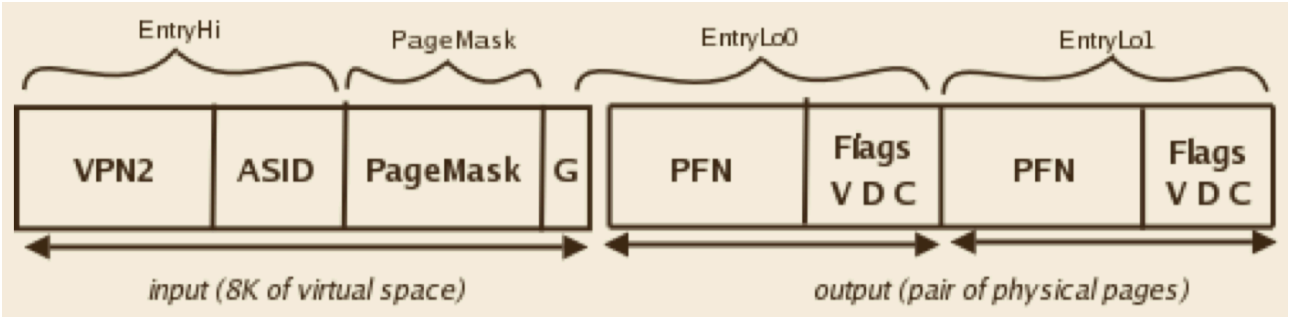


Project 4 Virtual Memory

- Page table setup
  - Statically fill page table
    - Fill paired VA to PA mappings
    - How many PTEs will you initialize?
  - On-demand paging
    - An empty page table
    - Fill the page table when page fault occurs

Project 4 Virtual Memory

- Handling TLB
  - Structure of TLB entry
    - Each entry includes the mapping for two continuous virtual addresses
  - Four registers:
    - EntryHi, PageMask, EntryLo0, EntryLo1
  - TLB entries: 32



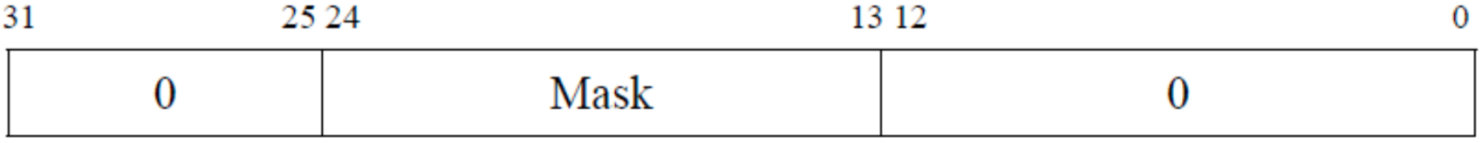
Project 4 Virtual Memory

- Registers for handling TLB
  - EntryHi (*CP0_ENHI*)
    - ASID: process ID
    - VPN (VPN2): refers to a pair of continuous virtual pages, e.g. virtual pages 0/1, 2/3

  - EntryLo0, EntryLo1 (*CP0_ENLO0, CP0_ENLO1*)
    - C: cachable or not, set to 010 in all tasks
    - D: writable or not (although it is called *dirty*)
    - V: valid or not

Project 4 Virtual Memory

- Registers for handling TLB
  - PageMask register (*CP0_PAGEMASK*)
    - Mask field (bit 13 - 24) indicates the size of a page frame
    - Choose the right value for Mask field



页大小	位											
	24	23	22	21	20	19	18	17	16	15	14	13
4Kbytes	0	0	0	0	0	0	0	0	0	0	0	0
16 Kbytes	0	0	0	0	0	0	0	0	0	0	1	1
64 Kbytes	0	0	0	0	0	0	0	0	1	1	1	1

Project 4 Virtual Memory

- Handling TLB
  - Instructions for manipulating TLB entry
    - tlbp: tlb lookup
    - tlbr: read TLB entry at index
    - tlbwi: write tlb entry at index
    - tlbwr: write tlb entry selected by random
    - Set the corresponding registers and then execute the above instructions
      - *CP0_ENHI, CP0_ENLO0, CP0_ENLO1, CP0_PAGEMASK, CP0_INDEX*
    - Coprocessor operations
      - mfc0, mtc0

Project 4 Virtual Memory

- Handling TLB
  - Instructions for manipulating TLB entry
  - Sample code

```
li k1, (vpn2<<13)|(asid & 0xff)
mtc0 k1, C0_ENHI

li k1, 0
mtc0 k1,C0_PAGEMASK

li k1, index_of_some_entry
mtc0 k1, C0_INDEX
tlbwi
```

Project 4 Virtual Memory

- Handling TLB exceptions
  - TLB refill: no matching entry in TLB
  - Invalid TLB: the matching TLB entry is invalid
  - TLB modification (not required in P4)

Project 4 Virtual Memory

- Handling TLB exception
  - TLB exception handler
    - Use *ExcCode* (2 or 3) to judge the exception type
    - Exception handling entry address 0x8000 0000, and place your exception handler at this address

Project 4 Virtual Memory

- Handling TLB refill
  - BadVAddr、Context、EntryHi registers all store the virtual address(VA) which triggers the TLB refill exception
  - Search this VA in page table and find the corresponding physical address
  - Insert the TLB entry

Project 4 Virtual Memory

- Handling invalid TLB
  - BadVAddr、Context、EntryHi register all store the virtual address(VA) which triggers the TLB invalid exception
  - Find the matching entry in page table, and update the corresponding TLB entry
  - Note that, you need to use *TLBP* instruction to locate the entry index, where invalid TLB exception occurs

Project 4 Virtual Memory

- Handling invalid TLB
  - Execute the following instruction, and the location of the entry is then stored in *CP0_INDEX* register

```
mtc0 a1, CP0_EntryHi
tlbp
```

Project 4 Virtual Memory

- Handling page fault
  - Page fault: not matching PTE in the page table
    - Allocate a physical page
    - Update the page table
    - Flush TLB entry
  - Note that, handling page fault is part of TLB exception handler

Project 4 Virtual Memory

- Step by step

Tasks	Assumption
Task1	You have all valid page table entries as well as all valid TLB entries
Task2	You have all valid page table entries but missing or invalid TLB entries
Task3	You have nothing ☹

Project 4 Virtual Memory

- Step by step
  - Task 1: setup page tables for process2.c
    - Statically setup page tables
      - Initialize page table entries and set all pages valid
      - Fill TLB entries for your page table entries
      - Note that, in this task, you need to
        - » Get familiar with page table, PTE, and TLB entries
        - » Design the PTEs in your page table
        - » Learn how to manipulate TLB entries

Project 4 Virtual Memory

- Step by step
  - Task 2: handling TLB exceptions for process2.c
    - Statically setup page (the same as task1)
    - Leave TLB entries empty
    - Implement TLB exception handlers for TLB refill exception and invalid TLB exception
    - Setup user-space stack for the tested process instead of using unmapped memory space as previous projects (e.g. using 0x00000000~0x7FFFFFFF)

Project 4 Virtual Memory

- Step by step
  - Task 3: handling page fault for process2.c
    - Setup an empty page table for triggering page fault
    - Add handling page fault into your TLB exception handler
    - Allocate physical page frames for missed virtual address
    - Need to distinguish the same virtual address from different processes

Project 4 Virtual Memory

- Requirements for design review (40 points)
  - What is the virtual memory range of the test process mapped by the test process?
  - Show the data structure of your page table. Where do you place the page table?
  - What are the initialized values for PTEs in tasks 1 and 2 respectively? How many initialized PTEs in both tasks 1 and 2?

Project 4 Virtual Memory

- Requirements for design review (40 points)
  - How do you handle TLB invalid exception? Please show the sample code/pseudo code for this process
  - Considering your page fault handler, where do you allocate physical page frames

Project 4 Virtual Memory

- Requirements of developing (60 points)
  - Implement static page table and TLB entries without TLB miss nor page fault(25)
  - Implement static page table and TLB exception handler without page fault (20)
  - Implement page fault handler with TLB miss and page fault, capable of isolating different processes(15)

Project 4 Virtual Memory

- Bonus: handle page in/out (4 points)
  - Select one page to page out to disk if there is no free physical pages
  - Which one to select?
    - Use FIFO
  - How to read/write pages?
    - Swap pages to disk using provided SD card read/write functions
  - Note that
    - Pinning pages: page directory, page tables, stack page table, kernel pages

Project 4 Virtual Memory

- Bonus: handle page in/out (4 points)
  - Limit the size of available physical memory, or you may need to increase the process image size by designing the test cases
  - Implement the page replacement when the required memory size exceeds the physical memory size
  - Choose an algorithm for the page replacement rather than FIFO

Project 4 Virtual Memory

- P4 schedule
  - design review: 25th Nov.
  - due: 2nd Dec.