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

University of Chinese Academy of Sciences

Lecture 4 Virtual Memory

2021.11.08

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

Schedule

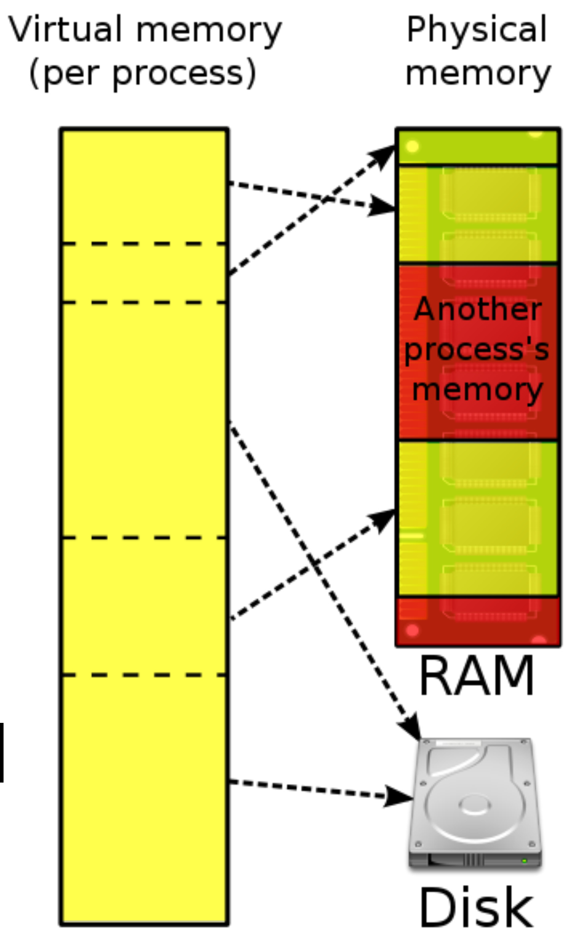
- Project 4 assignment
- Project 3 due

Project 4 Virtual Memory

- Requirement
  - Implement virtual memory management
    - Setup page tables for user-level processes
    - Handle page fault to support on demand paging assuming physical memory is enough
    - Handle page swapping assuming physical memory is limited
    - Support multi-threads running
    - Support shared memory

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, and is mapped into a physical page frame (e.g. 4KB)

Project 4 Virtual Memory

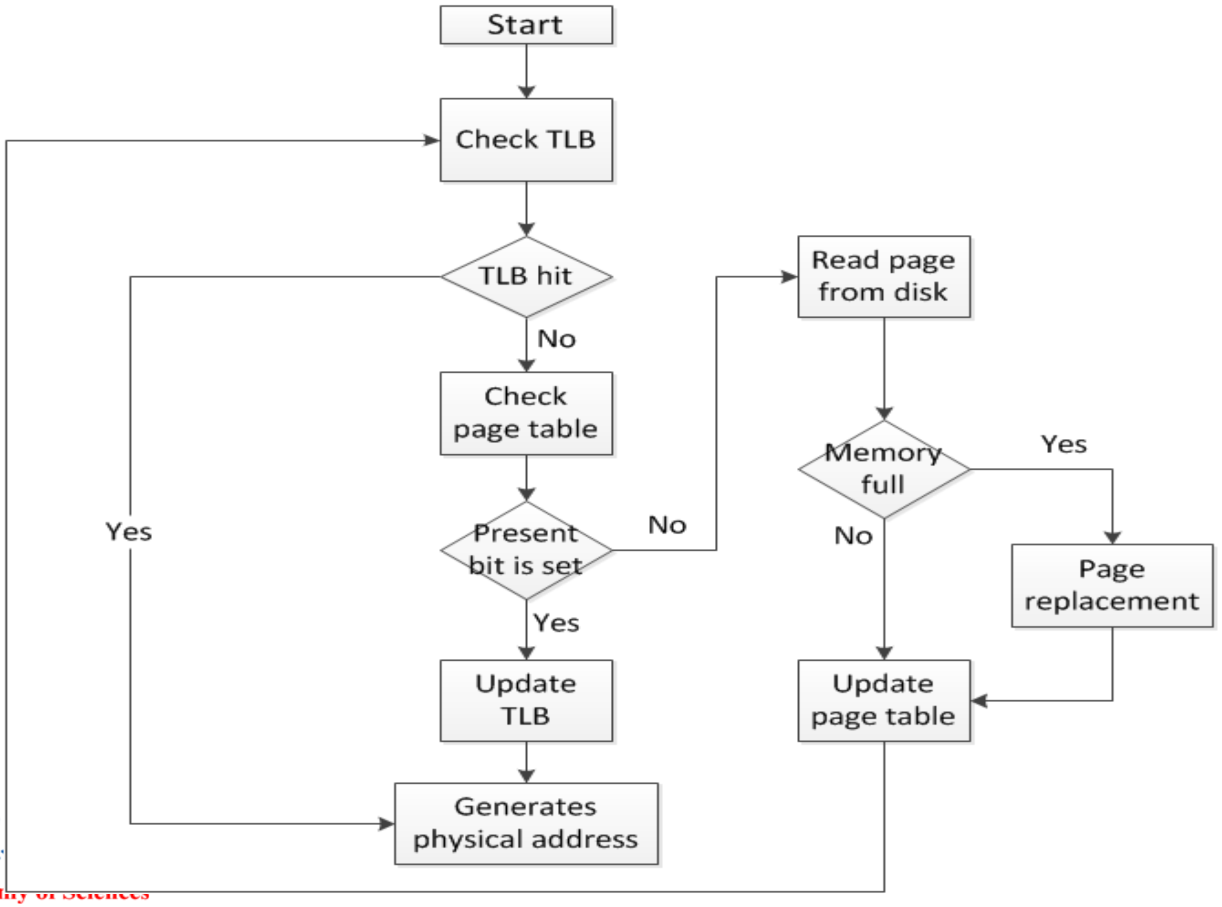
- Virtual memory
  - Page tables
    - The data structure to store the mapping between virtual addresses and physical addresses
    - Each mapping is a page table entry
  - Isolation among User Spaces and Kernel
    - Private Address Space
    - Process vs Thread
    - Shared Memory

Project 4 Virtual Memory

- Virtual memory
  - MMU and TLB
    - MMU stores a cache of recently used mappings from the page table, which is called translation lookaside buffer (TLB)
  - For RISC-V, TLB is handled by hardware
    - hardware page table walker

Project 4 Virtual Memory

- Virtual memory
  - Address translation



University of Chinese Academy of Sciences

Project 4 Virtual Memory

- Page table setup
  - After enabling virtual memory, both kernel and user-level processes use virtual addresses
  - Both kernel and user-level processes need to setup page tables
  - Pls. note that page table itself also needs page frames

Project 4 Virtual Memory

- Page table setup
  - Two-level page table is used for kernel, which is already setup in start code
  - You are required to use three-level page table for user-level process

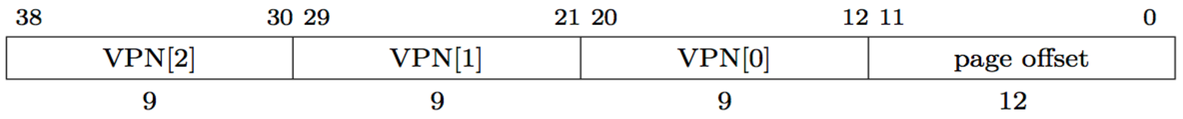


图 P4-3: Sv39 虚地址

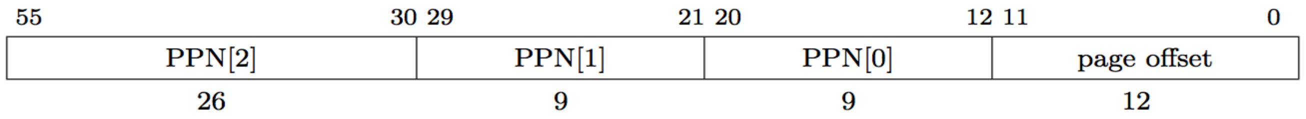


图 P4-4: Sv39 物理地址

Project 4 Virtual Memory

- Page table setup
  - Design the structure of page table entries (PTE)
    - Physical address
    - Valid
    - Access
    - Dirty

63 54	53 28	27 19	18 10	9 8	7	6	5	4	3	2	1	0
Reserved	PPN[2]	PPN[1]	PPN[0]	RSW	D	A	G	U	X	W	R	V
10	26	9	9	2	1	1	1	1	1	1	1	1

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

- Enabling virtual memory
  - Programmer need to set satp register
    - PPN is the physical page frame of page directory
    - Fill 8 in MODE to enable sv39 mode
    - ASID is used to distinguish different processes
    - Use *sfence.vma* to flush TLB

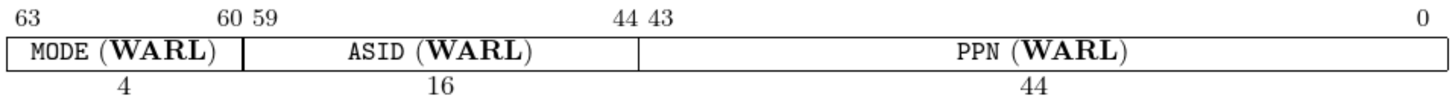


Figure 4.12: RV64 Supervisor address translation and protection register satp, for MODE values Sv39 and Sv48.

Project 4 Virtual Memory

- Handling page fault
  - Page fault: not matching PTE in the page table
    - Allocate a physical page
    - Update the page table to setup the mapping between virtual page and physical page
    - Flush TLB entry
  - Use *scause* register to identify interrupt type
  - Develop page fault handler
  - The address of virtual page causing page fault is stored in *stval* register

Project 4 Virtual Memory

- Handling swapping
  - Assuming you have limit memory space, you need to swap in-memory pages out to disk when there are no available pages.
  - When these swapped-out pages are reused again, swap the pages in from disk
  - Use sbi read/write functions we provided to read from and write to SD card

Project 4 Virtual Memory

- Process Address Space
  - Each Process has its private address space
    - No need to layout apps and stacks
    - Apps may start at same entrypoint and stack address
    - Seperated compilation and loading

  - All processes share kernel address space

Project 4 Virtual Memory

- Process vs. Thread
  - Threads like apps inside a process
    - entry points
    - stacks
    - schedule within a process： yield
  - All threads share process address space
    - single page table

Project 4 Virtual Memory

- S-Core (Task 1)
  - Statically setup page table for user-processes *shell* and *fly*
    - Initialize page table entries and set all pages valid
    - Filling kernel page table entries into user-level process page table
    - Use load_elf to load ELF image to virtual address, please refer to the guide book
    - Modify scheduler to allow switch page table among different processes
    - Reclaim memory space when executing *kill* and *exit*

Project 4 Virtual Memory

- A-Core
  - Task 2
    - Setup dynamic page table for user-level processes to support on-demand paging
    - Handling page fault for *rw.* Note that, *rw* runs when you can input different virtual addresses
    - You need support in-kernel locking as you did in Project 2. Note that, the lock resides in the shared kernel space

Project 4 Virtual Memory

- A-Core
  - Task 3
    - Limit the available memory space, and design a test case to access a range of memory addresses, which is larger than the available memory space
    - Supporting swapping with your own designed replacement policy

Project 4 Virtual Memory

- A-Core
  - Task 4
    - Implement multi-threads based mailbox as you implement in Project 3
    - Three processes use mailbox to communicate
    - For each process, recv_thread is created by *mthread_create* to receive msg from mailbox, and the main thread acts as send_thread to send msg to mailbox
    - send_thread send msg received by recv_thread by sharing a buffer with recv_thread

Project 4 Virtual Memory

- C-Core
  - Task 5
    - Refine fork to support copy on write
    - Child process shares the same page table with parent process until write happens
    - A new PTE is setup for the written page, and the original data is copied to the new page

Project 4 Virtual Memory

- C-Core
  - Task 6
    - Support shared memory between processes
    - Implement *shmpageget* and *shmpagedt*
    - *shmpageget(key):* the same key correspond to the same shared memory. Return a virtual address with the shared memory mapping
    - *shmpagedt(void* addr):* unmapping the virtual address and the shared memory

Project 4 Virtual Memory

- C-Core
  - Task 7
    - Support multi-threads with spin-lock in the same process address space
    - Compared multi-processes, multi-threads are expected to have better performance when executing number adding

Project 4 Virtual Memory

- Requirements for design review (40 points)
  - Show the data structure of your page table and page table entry. Where do you place the page table in physical memory? How large is your page table?
  - How do you handle page fault? pls. provides workflow or pseudo code
  - What is your replacement policy?

Project 4 Virtual Memory

- Requirements for design review (40 points)
  - What objects should be created to setup a thread? How to start/stop your thread ? How to schedule a thread?
  - Please ask any question you do not know

Project 4 Virtual Memory

- P4 schedule
  - Design review: 15th Nov.
  - Due: 22nd Nov.