Assignment6

11911839 聂雨荷

Q1

[25 pts] Read Chapter 15 of “Three Easy Pieces” (https://pages.cs.wisc.edu/~remzi/OSTEP/vm -mechanism.pdf) and explain how do the CPU hardware and the operating system cooperate in the procedure of address translation.

CPU hardware has the following requirements

Hardware Requirements	Notes
Privileged mode	Needed to prevent user-mode processes from executing privileged operations
Base/bounds registers	Need pair of registers per CPU to support address translation and bounds checks
Ability to translate virtual addresses and check if within bounds	Circuitry to do translations and check limits; in this case, quite simple
Privileged instruction(s) to update base/bounds	OS must be able to set these values before letting a user program run
Privileged instruction(s) to register exception handlers	OS must be able to tell hardware what code to run if exception occurs
Ability to raise exceptions	When processes try to access privileged instructions or out-of-bounds memory

- Hardware must also provide the base and bounds registers. When a user program is running, the hardware will translate each address, by adding the base value to the virtual address generated by the user program.
- The hardware must also be able to check whether the address is valid, which is accomplished by using the bounds register and some circuitry within the CPU.
- The hardware should provide special instructions to modify the base and bounds registers, allowing the OS to change them when different processes run.

- The CPU must be able to generate exceptions in situations where a user program tries to access memory illegally

OS has the following requirements

OS Requirements	Notes
Memory management	Need to allocate memory for new processes; Reclaim memory from terminated processes; Generally manage memory via free list
Base/bounds management	Must set base/bounds properly upon context switch
Exception handling	Code to run when exceptions arise; likely action is to terminate offending process

- First, OS must take action when a process is created, finding space for its address space in memory. OS will have to search a data structure (often called a free list) to find room for the new address space and then mark it used
- Second, the OS must do some work when a process is terminated, reclaiming all of its memory for use in other processes or the OS. Upon termination of a process, the OS thus puts its memory back on the free list, and cleans up any associated data structures as need be.
- Third, the OS must also perform a few additional steps when a context switch occurs. OS must save and restore the base-and-bounds pair when it switches between processes. Specifically, when the OS decides to stop running a process, it must save the values of the base and bounds registers to memory, in some per-process structure such as the process structure or process control block (PCB).
- Fourth, the OS must provide exception handlers, or functions to be called, as discussed above. the CPU will raise an exception; the OS must be prepared to take action when such an exception arises

The cooperation between CPU hardware and OS is listed in below:

OS @ boot (kernel mode)	Hardware	
initialize trap table		
	remember addresses of...	
	system call handler	
	timer handler	
	illegal mem-access handler	
	illegal instruction handler	
start interrupt timer		
	start timer; interrupt after X ms	
initialize process table		
initialize free list		

OS @ run (kernel mode)	Hardware	Program (user mode)
To start process A:		
allocate entry in process table		
allocate memory for process		
set base/bounds registers		
return-from-trap (into A)		
	restore registers of A	
	move to **user mode**	
	jump to A's (initial) PC	
		Process A runs
		Fetch instruction
	Translate virtual address	
	and perform fetch	
		Execute instruction
	If explicit load/store:	
	Ensure address is in-bounds;	
	Translate virtual address	
	and perform load/store	
		...
	Timer interrupt	
	move to **kernel mode**	
	Jump to interrupt handler	
Handle the trap		
Call `switch()` routine		
save regs(A) to proc-struct(A)		
(including base/bounds)		
restore regs(B) from proc-struct(B)		
(including base/bounds)		
return-from-trap (into B)		
	restore registers of B	
	move to **user mode**	
	jump to B's PC	
		Process B runs
		Execute bad load
	Load is out-of-bounds;	
	move to **kernel mode**	
	jump to trap handler	
Handle the trap		
Decide to terminate process B		
de-allocate B's memory		
free B's entry in process table		

Figure 15.5: **Limited Direct Execution Protocol (Dynamic Relocation)**

Q2

[25 pts] Read Chapter 16 "(https://pages.cs.wisc.edu/~remzi/OSTEP/vm-segmentation.pdf) and chapter 18 (https://pages.cs.wisc.edu/~remzi/OSTEP/vm-paging.pdf) of "Three Easy Pieces" and compare segmentation and paging. Your answer should cover all aspects (e.g., size of chunks, management of free space, context switch overhead, fragmentation, status bits and protection bits, etc.) and compare them side-by-side.

Aspects	Segmentation	Paging
size of chunks	variable-sized chunks	fixed-sized chunks

Aspects	Segmentation	Paging
management of free space	compact physical memory by rearranging the existing segments use a free-list management algorithm that tries to keep large extents of memory available for allocation (best-fit, worst-fit, first-fit or buddy algorithm)	OS keeps a free list of all free pages for this, and just grabs the first four free pages off of this list
context switch overhead	high OS saves and restores segment registers OS finds space in physical memory for its segments	low paging requires us to perform one extra memory reference in order to first fetch the translation from the page table Extra memory references are costly, and in this case will likely slow down the process by a factor of two or more
fragmentation	external fragmentation	internal fragmentation

Aspects	Segmentation	Paging
status bits	**protection bits**: indicating whether or not a program can read or write a segment **a grows positive bit**: the hardware also needs to know which way the segment grows (1-segment grows in the positive direction, 0-negative)	**a valid bit**: indicate whether the particular translation is valid **protection bits**: indicating whether the page could be read from, written to, or executed from **a present bit**: indicates whether this page is in physical memory or on disk **a dirty bit**: indicating whether the page has been modified since it was brought into memory **a reference bit**: is sometimes used to track whether a page has been accessed etc.
protection bits	Segmentation uses it to indicate whether or not a program can read or write a segment, or perhaps execute code that lies within the segment.	Paging uses protection bits to indicate whether the page could be read from, written to, or executed from.

Q3

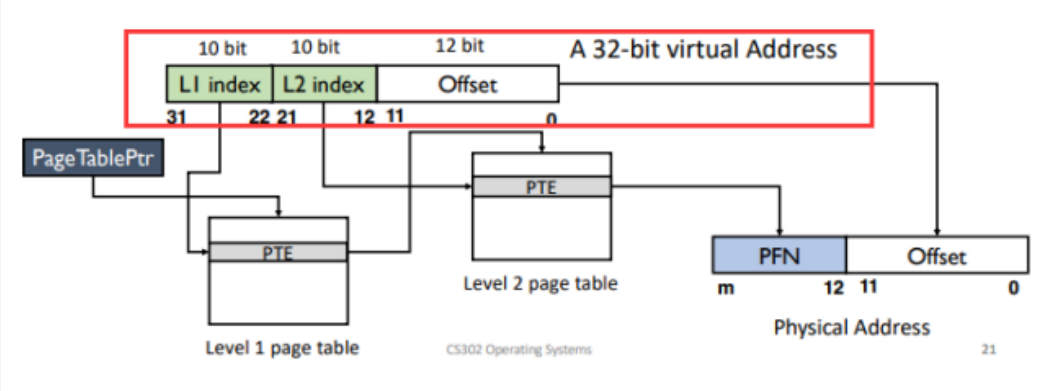
[25 pts] Consider a system with the following specifications:

- 46-bit virtual address space
- Page size of 8 K Bytes
- Page table entry size of 4 Bytes

- Every page table is required to fit into a single page

How many levels of page tables would be required to map the entire virtual address space? Please document the format of a virtual address under this translation scheme. Briefly explain your rationale.

Hint: Here is the example of the format of a 32-bit virtual address in lecture.



- Page size: 8KB=2^{13}B, therefore the offset has 13 bits
- PTE size: 4B=2^2B
- Every page table is required to fit into a single page, which contains $\frac{2^{13}}{2^2} = 2^{11}$ entries PTEs
- The virtual address apace is 46 bits, and the page size is 2^{13}B, therefore, the number of pages required to map the entire virtual address space are $\frac{2^{46}}{2^{13}} = 2^{33}$ pages
- In order to map all the virtual spaces, we need to have have at least 3 levels page table, each page table has 11 bits representation

The virtual address translation scheme is as follows:

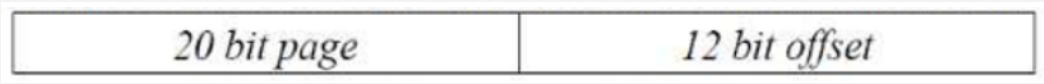
L1	L2	L3	Offset
45-35(11 bits)	34-24(11 bits)	23-13(11 bits)	12-0(13 bits)

Q4

[25 pts] Consider a system with following specifications: Both virtual address space and physical address are 32bits.

Page table entry size of 4Bytes

(a) Suppose it uses 1-level page table, the format of the translation scheme is



What is the page size? What is the maximum page table size?

(b) Suppose it uses 2-level page table, the format of the translation scheme is

10 bit page	10 bit page	12 bit offset

- Please write down the 1-st level page number and its offset in decimal(base 10) of virtual address 0xC302C302(base 16).
- Please write down the 2-nd level page number and its offset in decimal(base 10) of virtual address 0xEC6666AB (base16).

(a)

- The offset has 12 bits. Thus, the page size is 2^{12} bits = 4KB
- The 1-level page table has 20 bits, thus it can store 2^{20} entries of PTE. Since each PTE is 4Bytes, the maximum page table size is $2^{20} \times 2^2 = 2^{22}$ bits = 4MB

(b)

0xC302C302(base 16) = 11000011000000101100001100000010(base 2)

L1	L2	Offset
1100001100	0000101100	001100000010

- The 1-st level page number is 1100001100(base 2) = 780(base10)
- offset: 001100000010(base 2) = 770(base 10)

0xEC6666AB(base16) = 11101100011001100110011010101011(base 2)

L1	L2	Offset
1110110001	1001100110	011010101011

- The 2-nd level page number is 1001100110(base 2) = 614(base 10)
- offset: 011010101011(base 2) = 1707(base 10)