CS2106 Operating Systems

Semester 1 AY2021/22

Tutorial 9

Virtual Memory Management

1. (TLB, paging, and virtual memory)

We have learnt the idea of paging, translation lookaside buffers (TLBs) and virtual memory in the last two lectures. In this question, we are going to put them all in the same scenario and study the interaction.

Here is the basic setup. Note: To make the question easier to comprehend, the scale is vastly reduced compared to real-world setups.

- Page Size = Frame Size = 4KB
- TLB = 2 entries (0..1)
- Page Table = 8 entries (i.e. 8 pages, 0..7)
- Physical Frame = 8 frames (0..7)
- Swap Page (Virtual page in hard disk) = 16 pages (0..15)

Below is a snapshot of process P at time T:

TLB (should have the same fields as the PTE, not shown for simplicity):

Page No	Frame No
3	7
0	4

Page table:

Page No	Frame No/ Swap Page No	Present?	Valid?
0	4	1	1
1	1	1	1
2	1	0	1
3	7	1	1
4	2	1	1
5	15	0	1
6	—	0	0
7	—	0	0

Although the content and layout of the physical memory frames and hard disk swap pages can be deduced from the above, you are encouraged to sketch the RAM and hard disk content to aid understanding.

a. Below is the skeleton of the access algorithm for this setup. Give the missing algorithm for the OS **page fault** handler.

Algorithm for accessing virtual address VA:
1. [HW] VA is decomposed into <Page#, Offset>
2. [HW] Search TLB for <Page#>:
    a. If TLB miss: Check the page table
3. [HW] Is <Page#> memory resident?
    a. Non-memory resident: Trap to OS { **Page Fault** }
4. [HW] Use <Frame#><Offset> to access physical memory.

b. Using (a), walkthrough the following access in sequence. For simplicity, only the page number is given. If TLB or page replacement is needed, pick the entry with the smallest page number.

i. Access page 3.
ii. Access page 1.
iii. Access page 5.

(Optional) Think about how a dynamically-allocated new page fits into this scheme.

c. Suppose we have the following hardware specifications.

- TLB access time = 1ns
- Memory access time = 30ns
- Hard disk access time (per page) = 5ms

Give the best and worst memory access scenarios and the respective memory access latencies.

d. If 2-level paging is used, is there a worse scenario compared to (c)?

2. (Page table structure)
Given the following information:

- Virtual Memory Address Space = 32 bits
- Physical memory = 512MB
- Page Size = 4 KB
- PTE Size = 4 bytes

Suppose there are 4 processes, each using 512MB virtual memory. Answer the following:

a. Direct paging is used. What is the total space needed for all page tables used?
b. 2-level paging is used. What is the total space needed for this scheme?
c. Inverted table is used. What is the total space needed? You can assume each inverted table entry takes 8 bytes.

Why do you think the inverted table entry takes up more space than a page table entry?

3. (Adapted from final exam, AY 2018/19 S 1)
Below is a snapshot of the memory frames in RAM on a system with virtual memory. The memory pages in the frame is shown as <Process><Page#>, e.g. B17 means Page 17 of Process B.

RAM

Frame #	Contents
0	B17
1	A31
2	A04
3	A17

OS

	Ref.
Victim ➔	1
	1
	0
	1

The OS maintains additional information shown on the right to perform second chance page replacement algorithm on the memory frames.

Suppose **process A** accesses **page 8** (i.e. **A08**) now.

a. Which memory frame will be replaced?
b. Give the steps the OS takes to update the affected page table entries (without an inverted page table).

Continuing from (a), suppose **process B** accesses **page 13** (i.e. **B13**) now.

c. Which memory frame will be replaced?
d. If the OS keeps an inverted page table, give the contents of the inverted page table after the replacements (after (a) and (c)).
e. Give the steps the OS takes to update the affected page table entries with the help of the inverted page table.

4. (Applications of virtual memory)
An application of virtual memory in some OSes (including Linux) is overcommit. When overcommit is enabled, the OS will not allocate pages immediately upon request, but instead only when they are actually used.

In Linux, a new memory allocation has its contents initialised to zero. Overcommit allows Linux to defer allocating pages for a memory allocation until the program writes to the memory, (if it ever does).

a. How can this be implemented, while allowing reads to be successful?

Another application of virtual memory is demand paging, which ties in closely with the idea of memory-mapped files (i.e. `mmap`). Memory-mapped files are a natural extension to the idea of a swap file; in a way, we are simply allowing programs to specify the "backing store" for a particular region of memory.

b. How can demand-paged (i.e. the file is not read until the program actually reads from the mapped region) memory-mapped files be implemented?
c. What are the advantages and disadvantages of using memory-mapped files compared to regular read/write system calls?

For your own exploration:

5. (Page table structure)
A computer system has a 36-bit virtual address space with a page size of 8 KB, and 4 bytes per page table entry.

a. How many pages are there in the virtual address space?
b. What is the maximum size of addressable physical memory in this system?
c. If the average process size is 8 GB, would you use a one-level or two-level page table? Why?