CS2106 Introduction to Operating Systems

Semester 2 2023/2024

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. Assume that the memory is currently full.

   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] [Adapted from AY1920S2 Final – Page Table Structure] The Linux machines in the SoC cluster used in the labs have 64-bit processors, but the virtual addresses are 48-bit long (the highest 16 bits are not used). The physical frame size is 4KiB. The system uses a hierarchical direct page table structure, with each table/directory entry occupying 64 bits (8 bytes) regardless of the level in the hierarchy.

Assume Process *P* runs the following simple program on one of the lab machines:

```
1.  #include <stdio.h>
2.  #include <stdlib.h>
3.  #define ONE_GIG 1<<30
4.  #define NUM_PAGES 1<< 18
5.  void main(){
6.    char* array = malloc(ONE_GIG);
7.    int i;
8.    for (i=0; i<NUM_PAGES; i++)
         array[i<<12]='a';
9.    printf("0x%lX\n", array);
10.   // point of interest
11.   free(array);
12. }
```

At line 9, the program prints out the value of variable **`array`** in hexadecimal:

```
0x7F9B55226010
```

Consider the point in time when the process execution reaches the point of interest (line 10 in the listing). Answer the following questions:

a. If we use a single-level page table, how much memory space is needed just to store the page table for process *P*?
b. How many levels are there in the page-table structure for process *P*?
c. How many **entries in the root page directory** are holding information related to process *P*'s **dynamically allocated data**?
d. How many **physical frames** are holding information related to process *P*'s dynamically allocated data in the **second level** of the hierarchical page-table structure (next after the root)?
e. How many **physical frames** are holding information related to process *P*'s dynamically allocated data in the **penultimate level of the page-table structure** (i.e., the level before the last one).
f. How many **physical frames** are holding information related to process *P*'s dynamically allocated data in the **last level of the page-table structure?**

3. [Adapted from final exam, AY 2018/19 S1] 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.

	Frame #	**Contents**		**Ref.**	
	0	B17	Victim ➔	1	
RAM	1	A31		1	**OS**
	2	A04		0	
	3	A17		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. [Virtual memory](Adapted from final exam, AY 2019/20 S 1) The virtual address space supported is 2^{64} bits (not bytes). The page size is 1KiB (2^{10} bytes), the size of the physical memory (RAM) is 32KiB, the size of a page table entry is two bytes. Assuming the addressing is at the byte level, calculate the size of the page table required for both standard and inverted page tables (the size of the entry in the inverted page table is the same with the entry in the direct page table). [ONLY IF TIME PERMITS]

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?