Operating Systems 2016/17
Solutions for Assignment 10

Prof. Dr. Frank Bellosa
Dipl.-Inform. Marc Rittinghaus

T-Question 10.1: Caches

a. Why do virtually-indexed, physically-tagged caches not suffer from the ambiguity problem? **1 T-pt**

Solution:
The ambiguity problem arises when two identical virtual addresses point to different physical address at two different points in time and the cache potentially holds old data after the mapping has been modified.

Physically-tagged caches use the physical address as tag and thus can determine if a change in the mapping has occurred.

b. Consider the following code fragment. The algorithm has been optimized so that the loop only has to modify every 16th array element ($k = 16$) instead of every element ($k = 1$). However, although the loop now does only 6% of the original work, the execution of the loop still needs 98% of the original time. What might be the reason? Explain your answer! **1 T-pt**

```
#define ARRAY_LENGTH 64 * 1024 * 1024
int a[ARRAY_LENGTH];

// --- some algorithm here ---

for (int i = 0; i < ARRAY_LENGTH; i += k) { a[i]++; }
```

Solution:
The array stores `int` elements which are each 32 bits (i.e., 4 bytes) in size. The elements are stored contiguously in memory. When the CPU accesses the first element, it has to read the element's value from memory. This operation will trigger a fetch of a whole cache line. Access to any subsequent elements that are on the same cache line do not require a costly memory transfer. Since the performance nearly stays constant even with $k = 16$, the cache line size must be at least $16 * 4bytes = 64bytes$.

c. What kinds of cache-misses do exist? What can you do to reduce the number of cache misses of each type? **3 T-pt**

Solution:

Compulsory Misses: *Cannot be avoided. Prefetching can help to avoid their latency, though: The missing cache line still needs to be fetched from memory but the processor does not have to wait for that fetch. Reducing the number and frequency of context switches can help reduce compulsory misses for virtually-indexed virtually-tagged caches, as those caches need to be flushed upon a context switch.*

Capacity Misses: *Get larger caches or reduce the working set. (One answer is sufficient.)*

Conflict Misses: *Consider page colors and access pattern in memory allocation. (Increasing cache associativity is also ok as an answer. Unfortunately, we as OS architects cannot change the hardware, though).*

T-Question 10.2: Paging

Consider a system that translates virtual addresses to physical addresses using hierarchical page tables. Every page table comprises 512 entries, with each entry having a size of 8 bytes. The size of both the virtual and the physical address spaces is 512 GiB. The page size is 4096 bytes.

a. How many page tables build a full page table hierarchy in the given system? How many levels does the hierarchy have? **2 T-pt**

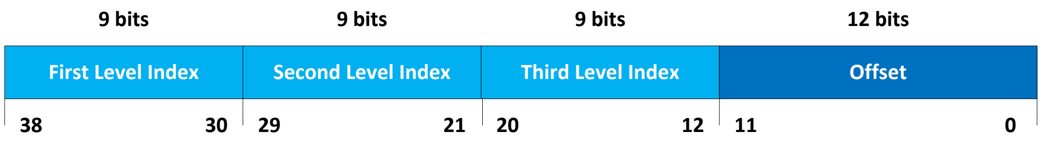
Solution:
The number of pages in a 512 GiB address space is $\frac{512\,GiB}{4\,KiB} = 128\,Mi = 2^{27}$*. A page table has* $512 = 2^9$ *entries* $\Rightarrow \frac{2^{27}}{2^9} = 2^{18}$ *page tables.*

To form a hierarchy with 512 entries per table we need two extra levels to address the page tables: $2^{18} = (2^9)^2 = 512^2$*. That gives us* $512 + 1$ *extra page tables for a total of* $2^{18} + 2^9 + 1$ *page tables in three levels.*

b. Into what parts would a MMU for the system split the virtual address during address translation? For each part give its length in bits. **2 T-pt**

Solution:
The page size is 4 KiB $= 2^{12}$*. The lower 12 bits are used as offset. From a) we know that we have three page table levels, with each page table having 512* $= 2^9$ *entries. That means we need 9 bits per level. Test: We get a total of* $27 + 12 = 39$ *bits,* $2^{39} =$ *512 GiB.*

You may also solve the question the other way around, without knowing the number of levels in advance: The address space is 512 GiB in size, the addresses must therefore be 39 bits wide. We know that the offset is 12 bits, leaving us 27 bits. Each level should hold 512 $= 2^9$ *entries and must therefore consume 9 bits, giving us* $27/9 = 3$ *levels.*



c. How many page tables exist when using inverted page tables and single level forward page tables in the operating system, respectively? Briefly explain how you derived your answer. **1 T-pt**

Solution:

Inverted Page Table *1 inverted page table per system. The inverted table is shared across all processes and maps page frames to pages.*

Single-level Page Table *1 page table per process, which map each process's virtual pages to physical frames.*

Total: 10 T-pt