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

Question 9.1: Memory Management Basics

a. Discuss the difference between a virtual and a physical address.

Solution:
The difference occurs only in systems with no fixed 1:1 mapping of the applications' addresses to the physical addresses. In such systems, each executable program belongs to a virtual address space. All program addresses in fact are virtual addresses, that is, none of the instructions deal directly with physical addresses.

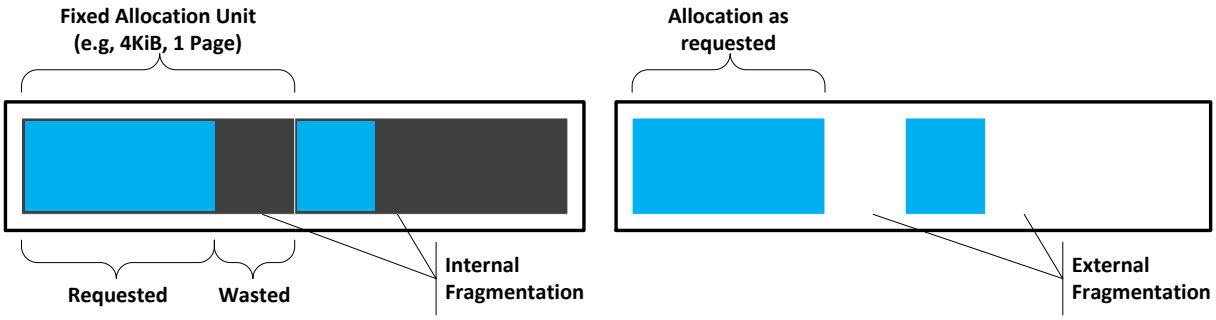
During execution, however, parts of the programs are mapped to physical memory. In order to access the contents, a memory management unit (MMU) translates virtual addresses into physical addresses via some mapping information, such as a base register or a page table in a paging-based memory system.

Addresses within a binary file that the compiler or the linker has computed and even the addresses resolved by the loader are all virtual addresses. The physical addresses are the addresses that are delivered to the main memory chips when an instruction fetch or a data access is executed.

b. Explain the difference between external and internal fragmentation.

Solution:
If your memory manager only offers memory blocks of fixed sizes, then internal fragmentation cannot be avoided: You often get more memory than you have asked for. The unused portion of the memory (i.e., the difference between requested size and block size) cannot be used and is called internal fragmentation.

On the other side, there are memory managers which can offer memory blocks of almost any size. Due to different lifetimes of these tailored memory blocks, the memory as a whole may be scattered: It may contain a lot of memory holes, each of them being too small for an upcoming memory request, although the total of free memory would be sufficient. The sum of the currently not usable memory is called external fragmentation.



Both types of fragmentation can happen at the same time, for example if a fixed allocation unit and segmentation for address translation is used. The free space in physical memory may be too scattered to allow the allocation of a segment of a certain length (which must be contiguous in physical memory), even if enough total free space is available (external fragmentation). The restriction to allocate only in fixed units (e.g., the segment size must be a multiple of 4 KiB) potentially wastes memory within a segment (internal fragmentation).

c. When is compaction a viable option to reduce external fragmentation?

Solution:
Compaction can only be done if all references to moved memory blocks can be updated. When using segmentation and external fragmentation in the physical address space should be reduced, that can easily be accomplished by updating the segments' base addresses. However, compaction may not be used for memory blocks allocated on the heap of a C program with a call to `malloc()`*. That is compaction cannot reduce the external fragmentation in a native heap, because the heap is not able to reliably track and update all pointers to the allocated memory.*

Question 9.2: Segmentation

a. How does segmentation work?

Solution:
With segmentation, a virtual address space is regarded as a collection of segments. Each segment represents a separate part of a program, such as the code, the stack, or some library. Segments occupy contiguous parts of the virtual address space.

Virtual addresses are considered tupels, consisting of a segment number and an offset. The segment number is used as an index into a segment table. A segment table contains a number of entries, each of which consists of a segment base and a segment limit. The base describes the starting address of the segment in physical memory, the limit specifies the length of the segment.

A virtual address can be translated to a physical address with the following steps:

(a) The segment number of the virtual address is used as an index into the segment table to find the correct segment table entry.

(b) The offset of the virtual address is compared against the limit of the segment table entry. If it is ***larger or equal****, the virtual address refers to a location outside the segment, and an exception is raised.*

(c) If the address is valid (i.e., inside the segment), the base value of the entry is added to the offset of the virtual address. The resulting value is the desired physical address.

b. Assume a system with 16-bit virtual addresses that supports four different segments, which uses the following segment table:

Segment Number	Base	Limit
0	0xdead	0x00ef
1	0xf154	0x013a
2	0x0000	0x0000
3	0x0000	0x3fff

Complete the following table and explain briefly how you derived your solution for each row in the table.

Virtual Address	Segment Number	Offset	Valid?	Physical Address
	3	0x3999		
0x2020				
		0x0204	yes	
			yes	0xf15f

Solution:
The segmentation address is split as follows:

Line 1:
To get the virtual address from the segment number and the offset, we can simply concatenate the bits, which gives us: `0xf999`

A lookup in the segment table reveals that the offset `0x3999` *is smaller than the segment's limit of* `0x3fff`*. We can therefore take the physical base of the segment and add the offset to get the physical address.*

Line 2:
We have to do the reverse operation and split the address `0x2020` *into the segment index and the offset part. This will give us segment number* `0` *and offset* `0x2020`*. Since the offset* `0x2020` *is greater than segment 0's limit, the virtual address is not valid and cannot be translated to a physical address.*

Line 3:
Line 3 constrains the virtual address to possess a valid translation with offset `0x0204`*. We therefore must take segment* `3`*. The valid virtual address is* `0xC204` *and the physical address is* `0x0204`*.*

Line 4:
The last line demands the virtual address that translates to the physical address `0xf15f`*. We can find the segment by subtracting each segments base address from the physical address and check if the resulting offset is within the respective segment's limits. This is the case for segment* `1`*, where* `0xf15f - 0xf154` = `0x000b` *(smaller than* `0x013a`*). We can then use the method from line 1 to get the virtual address* `0x400b`*.*

Note: Because segment 2's limit is 0, there cannot be a valid translation using this segment.

Virtual Address	***Segment Number***	***Offset***	***Valid?***	***Physical Address***
`0xf999`	`3`	`0x3999`	`yes`	`0 + 0x3999`
`0x2020`	`0`	`0x2020`	`no`	*Offset outside segment limit*
`0xC204`	`3`	`0x0204`	`yes`	`0x0204`
`0x400b`	`1`	`0x000b`	`yes`	`0xf15f`