MEM2-SEG: 151-180

mem2-segmentation:

151 [T / F] With segmentation we don't have to worry about the process bound ... yay! A process now believes it has the entire logical address space (*e.g.*, a 4GB logical space in a 32-bit architecture).

152 [T / F] In Segmentation, processes of the same program can share the same code segment.

153 [T / F] "*(int*)main = 7" will cause a segfault (or sometimes called "bus error") because the code segment is not readable.

154 [T / F] If you have N processor cores in your computer, all the N cores share the same MMU (*i.e.*, there is only only one MMU in your computer).

155 [T / F] In segmentation, the width of the logical address is the width of the segment bits plus the width of the segment-offset bits.

156 [T / F] With segmentation, the top bits represent the index to the segment table. That is, the top bits tell us which segment (code/heap/stack) that the logical address is trying to go to.

157 [T / F] In all the memory management we discuss in class, the values in the segment/page table entries are populated (decided) by the MMU. DLS

158 [T / F] In this class, code is in the *lowest* segment of the process address space and stack is in the *highest*. This means the *most significant bit* of logical addresses within the *code* segment is "1". This also means the *most significant bit* of logical addresses of your local variables within the *stack* area is "0".

LMAO **159** [T / F] I promise to do all the address-translation examples (logical to physical, plus the error-bound and r/w-bit checking).

160 [T / F] In Segmentation, the OS can support more segments (*e.g.*, 16) than what your process typically uses (*e.g.*, 4), hence the OS must introduce read-write bits in every segment table entry.

161 [T / F] In Segmentation, the MMU throws a segfault exception when the segment offset is out of bound *AND* the memory access violates the read/write bits.

162 [T / F] In this class, even for stack, I will do stackBase+offSet, not stackBase-offSet (as in the OSTEP book). The professor would like to make my life simpler.

163 [T / F] Chop, chop, chop ... I promise I will be careful when chopping the virtual (logical) address. I will read the recipe (i.e. the specification) carefully because a wrong chop will make the meal taste bad (i.e. wrong answer). There will be no partial credits! The graders are mean like Gordon Ramsay.

164 [T / F] Virtual addresses provide privacy and fault isolation. By default there is no way i can have a pointer pointing to another process' address space.

165 [T / F] An instruction that dereferences a bad/dangling pointer does not always directly lead to a segfault/crash.

166 [T / F] Using top 3 bits as the segment indexing is sufficient for OSes that want to support up to 15 segments per process.

167 [T / F] With 32-bit addressing and 64-segment support, we can create a segment as large as 128 MB.

168 [T / F] To limit a segment size to only 4 MB, we must use 10 bits for the segment indexing. (Assume a 32-bit addressing).

169 [T / F] Segmentation solves the BB's external fragmentation problem.

mem2: mem. mgmt. today

170 [T / F] Today's OSes basically have solved both the internal and external fragmentation problems by forcing everyone to use "small, fixed-sized luggages" (i.e. paging), but I'm glad I don't have to know about 4-level page tables (for now).

171 [T / F] Today, many OSes use paging because paging solves both external and internal fragmentation. If a page size is 4KB, the internal fragmentation still exists (*e.g.*, a user expands the heap by 1Byte but the OS must give a 4KB-page), but the amount of wasted space (e.g. 3.999 KB) is so small compared to typical memory sizes today (4GB or bigger).

172 [T / F] Virtual memory gives the illusion of a memory larger than the physical memory.

173 [T / F] If you have a 32-bit logical address space, that means you can always fill up your address space with 2^{32} bytes (4GB) of data (of various types, code, heap, etc.), because the size of the logical address space represents the true limit of how many physical pages can be allocated for your process.

174 [T / F] If you have a 48-bit logical address space, that means you can always allocate 2^{48} bytes (256TB) of data (of various types in the code, heap, etc.).

175 [T / F] When my Microsoft Powerpoint is open but I never use it for many days (because I never review the professor's slides), the moment I click on it, things look slow because the powerpoint's heap, code, and stack have been "swapped out" to the disk's swap space area by the OS, and they must be "swapped in" back to the memory. This phenomenon is called "page faults" – i.e. the data (page) you want to access are not in the memory but in the disk, and must be brought up to the memory first before your application can continue to run. The analogy is your carry-on luggage has been swapped out from the overhead bin (the memory) to the baggage compartment (the disk) because you have been sleeping in the airplane too long and never touch your luggage and the flight attendant (the OS) needs to the space for someone else' luggage. When you wake up and need your luggage, it will take some time for the flight attendant (the OS) to bring your luggage from the baggage compartment to the overhead bin (swap in).

176 [T / F] ...

177 [T / F] ...

178 [T / F] ...

179 [T / F] ...

180 [T / F] 180