**1. Explain the difference between internal and external fragmentation.**

External fragmentation means that enough free space is available to satisfy a memory request, but that memory is not contiguous. The result of this is that memory is fragmented into a bunch of small holes.

Internal fragmentation occurs when a process is allocated a bit more memory than it needs. This extra memory is not used by the process, but cannot be used by the system or other processes.

**2. Given memory partitions of 100K, 500K, 200K, 300K, and 600K (in order), how would each of the First-fit, Best-fit, and Worst-fit algorithms place processes of 212K, 417K, 112K, and 426K (in order)? Which algorithm makes the most efficient use of memory?**

First fit:

212K process in 500K slot

417K process in 600K slot

112K process in 200K slot

426K process must wait

Best fit:

212K process in 300K slot

417K process in 500K slot

112K process in 200K slot

426K process in 600K slot

Worst fit:

212K process in 600K slot

417K process in 500K slot

112K process in 300K slot

426K process must wait

The best fit algorithm makes the most efficient use of memory, and is the only one that does not require a process to wait.

**3. Why are page sizes always powers of 2?**

To convert a linear address into a page and offset, the OS must divide the address by the page size. If the page size is a power of 2, this division can be achieved by a simple bit shift. However, if the page size is not a power of 2, the OS must perform an inefficient integer division operation.

In addition, if the address is stored as the page number concatenated to the offset, then having the page number size be a power of 2 ensures that the offset portion of the address is maximally efficient. For instance, say the page size is 6; this means that the offset can take the values 0 through 5. This requires that the offset take 3 bits. However, the numbers 6 and 7 can also be represented in 3 bits, so the offset should actually be able to take the values 0 through 7. Thus, the most efficient page size would be 8, which is a power of 2.

**4. On a system with paging, a process cannot access memory that it does not own; why? How could the operating system allow access to other memory? Why should it or should it not?**

Virtual to physical address translation requires the use of a page table to translate the virtual page number to the physical page number. The operating system controls the page table, so it can ensure that a process does not access physical pages that it does not own.

To allow a process to access other memory, the operating system can put that memory’s page number into the process’s page table. This enables interprocess communication through shared memory. However, the OS must be careful not to give a process access to another process’s memory unless that process allows it. Otherwise, the system would have security issues.

**5. Consider a paging system with the page table stored in memory.**

***a. If a memory reference takes 200 nanoseconds, how long does a paged memory reference take?***

400 nanoseconds because there are two total accesses to memory. The first access is to the page table to translate the virtual page number to the physical page number. The second access to the actual physical location in memory that contains the data of interest.

***b. If we add associative registers, and 75 percent of all page-table references are found in the associative registers, what is the effective memory reference time? (Assume that finding a page-table entry in the associative registers takes zero time, if the entry is there.)***

250 nanoseconds is the average time. 75% of the time, the total memory reference time is 200 nanoseconds, because only the actual physical location of the data in memory must be accessed. However, 25% of the time, the total memory reference time is 400 nanoseconds, as both the page table and the physical location of the data must be accessed.

**6. What is the effect of allowing two entries in a page table to point to the same page frame in memory? Explain how this effect could be used to decrease the amount of time needed to copy a large amount of memory from one place to another. What effect would updating some byte on the one page have on the other page?**

The effect of this is that two virtual addresses point to the same physical address in memory. This allows for processes to circumvent the need to copy data. Presume that one process has some data that another process wants access to. The operating system can put an entry in that process’s page table that points to this physical page frame. Thus, that process immediately gets access to the data, and the data doesn’t need to be copied at all.

However, if a byte is updated by one page, then that same byte is updated on the other page as well.

**7. Consider the following page reference string: 1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6. How many page faults would occur for the following replacement algorithms, assuming one, two, three, four, five, six, or seven frames? Remember all frames are initially empty, so your first unique pages will all cost one fault each.**

Faults with page size = 1

LRU: 20

FIFO: 20

Optimal: 20

Faults with page size = 2

LRU: 18

FIFO: 18

Optimal: 15

Faults with page size = 3

LRU: 15

FIFO: 16

Optimal: 11

Faults with page size = 4

LRU: 10

FIFO: 14

Optimal: 8

Faults with page size = 5

LRU: 8

FIFO: 10

Optimal: 7

Faults with page size = 6

LRU: 7

FIFO: 10

Optimal: 7

Faults with page size = 7

LRU: 7

FIFO: 7

Optimal: 7

**8. A system has a 32-bit architecture so each process has 32-bit virtual address space. Each page is 4KB. If a program's total data/code can be fit into two pages, then what are the minimal/maximum page table overhead (in terms of the number of page table entries) for**

***a. a one-level page table***

For a one level page table, we need 2^32 (bit of architecture) / 2^12 (size of page) = 2^20 = 1048576 page table entries regardless of the amount or location of data that the process uses.

If all of the data and code is located in two different pages, then only 2 page table entries contain valid mappings. Thus, 2^20 - 2 = 1048574 page table entries are wasted.

If each byte of data and code is located in a different page, then an entry is needed for each byte. Since the program’s total size = 2 \* 4KB = 2^13 = 8192 bytes, 8192 page table entries are needed. Thus, 2^20 - 8192 = 1040384 entries are wasted.

***b. a two-level page table, assuming level-1 and level-2 tables are of the same size.***

For the two level page table in this example, memory addresses contain three sections. The first section is 2^10 bits, and is the level 1 page number. The second section is also 2^10 bits, and is the level 2 page number. The third section is 2^12 bits and is the offset.

Regardless of the location of the process’s data, the level 1 page table will require 2^10 entries.

If all of the date and code is located in two different but adjacent pages, then only one page of the level 2 page table needs to be created. This results in 2^10 entries being created, for a total number of entries = 2^10 + 2^10 = 2048 entries. Only three of these entries are actually needed (one in the level 1 table and two in the level 2 table), so 2048 - 3 = 2045 entries are wasted.

If the 8192 = 2^13 bytes of data and code are spread out sparsely but evenly across memory, then every possible second level page table must be created. This results in 2^10 \* 2^10 = 2^20 second level page table entries, with the total number of entries = 2^20 + 2^10 = 1049600. The total number of wasted entries is 1049600 - 2^10 - 8192 = 1040384.