# Section 7: Memory Management

## Question 1

Describe both the logical and physical views of a process image’s memory addresses.

What is the device that translates from logical to physical addressing?

### Answer

Logical View: The logical view of the process image is the illusion that the process image is installed in a contiguous range of memory addresses that starts at address zero. A process image logical address space is owned by the process and is separate from all other process images. That is, that each process image has its own private 0-N address space.

Physical View: The physical view of the process image is the reality that all process images share common physical memory. Each process is installed into a memory partition at unique addresses in physical memory. Partitions can be relocated to new addresses, and with paged memory management can be placed in non-contiguous (non-adjacent) memory regions.

The translation from logical to physical addresses is primarily accomplished by the system’s Memory Management Unit hardware.

## Question 2

Describe the three types of dynamic partition placement algorithms described in class.

### Answer

First Fit: The operating system scans main memory, starting from the top, to find the first unallocated region of memory into which the process will fit regardless of amount of external fragmentation it creates.   
  
Next Fit: The operating system scans main memory, starting from the partition last allocated, to find the next unallocated region of memory into which the process will fit.  
  
Best Fit: The operating system scans main memory to find the unallocated region of memory that is the closest / best fit i.e. is the smallest unallocated region into which the process will fit.

## Question 3

1. Describe the meaning of process image partition relocation?
2. How is relocation of a process image implemented in the Primitive Memory Management Unit (MMU) described in the slides?
3. What is the unit of relocation in paged memory management?

### Answer

1. A process image is placed into and executed from a memory partition. A processes’ image can be relocated (moved) from one partition (region of physical memory) to a different partition throughout its lifetime. For example, slide 4 illustrates P1 relocation from one partition to a second.
2. The Primitive MMU implements relocation with two registers: Base and Bounds. The base register maintains the start of the process’s partition and the bounds maintains the partition’s size. A logical relative address is translated into a physical address by adding the bounds register to the relative logical address.
3. In Paged Memory Management, the unit of relocation is a Page of the process image.

## Question 4

What is the purpose of *memory* *overlaying* as described in the text?

### Answer

Overlaying is a technique that allows the execution of programs that are too large to fit into the available physical memory. Overlaying allows the programmer to explicitly divide their program into multiple modules and then specify which of the modules are moved into memory for execution. Newly loaded modules will replace (overlay) program modules that are currently in memory.

For example, some video games have many regions that a player can visit and a player cannot be in two regions at the same time. The meshes, textures, and other resources needed for all levels will not fit into a PC or console’s memory. These resources are loaded as the player enters a region and replace / overlay the currently active region’s resources allowing many regions to be loaded and played in the same physical memory.

## Question 5

1. Describe the purpose and contents of the page table as described in this section.
2. How many page tables are needed to maintain N processes?

### Answer

1. The page table maintains a number of page table entries each of which maintains process’s mapping from logical-address page numbers to physical-address (memory) frame numbers (base addresses).
2. Every process maintains its own page table.

## Question 6

NOTE: When creating the elearning assessment, be sure that superscripts are used.

Consider a simple paging system with the following parameters:

* 232 bytes of physical memory address range.
* A page size of 210 bytes.

1. What is the size of a frame?
2. How many bits of a logical address are used to identify a page table entry?
3. How many entries are needed in the process’s page table?
4. How many bits are needed in each page table entry?

### Answer

1. A frame is the same size as a page, 210 bytes.
2. 10 bits are used as the offset into the page / frame, so 22 (32-10) bits remain to index into the page table.
3. There is one Page Table Entry for each page in the logical address space, so 222 entries are needed.
4. Ignoring the present and modified bits, 22 bits are needed in each PT Entry to specify the base address of the frame in physical memory.

## Question 7

NOTE: When creating the elearning assessment, be sure that superscripts are used.

Note: Intel hardware’s page size is 212 or 4096.

1. What is the largest internal fragment that can occur with paging on Intel hardware?
2. What is the largest external fragment that can occur with paging on Intel hardware?

### Answer

1. The worst case is a page with a single byte resulting in an internal fragment of size (PageSize -1) or 4095 bytes.
2. A trick question. There is no external fragmentation with paging.

## Question 8

What is the maximum number of pages that can be addressed in Figure 7.12a?

### Answer

The page number is six bits long or 26 for 64 page entries.

## Question 9 (20 pts)

What is buffer overflow and how is the following code segment susceptible to it?

int main(void) {  
 **char buff[50];** int age = 2;  
 …  
 gets(&buff);  
 printf(“You Entered %s\n”, buff);  
 …  
}

### Answer

Buffer overflow occurs when a flaw in the code causes an unchecked input operation to overwrite its assigned input buffer to corrupt adjacent data on the stack or in memory.

In the example, the gets() function will copy input from stdin into a buffer passed to the function until a newline is found. Notice that gets is passed the location of a character buffer ‘buff’ with a fixed length of 50. If the data read from stdin exceeds 50 bytes, gets will begin to overwrite the adjacent integer ‘age’ with the data it is copying.

NOTE: The Unix / Linux library function gets(3) was replaced with the function fgets(3) which requires an additional argument: the buffer size to check for and prevent overflow.