**Lab work #4**

**"Study of memory allocation algorithms."**

**Objective**

To study memory allocation algorithms and peculiarities of their implementation.

Task **：**Implement a memory manager with segmental partitioning.

### 1. Constants Definition

**Description**: These define maximum number of memory segments (MAX\_SEGMENTS) and maximum size of each segment (MAX\_SEGMENT\_SIZE).

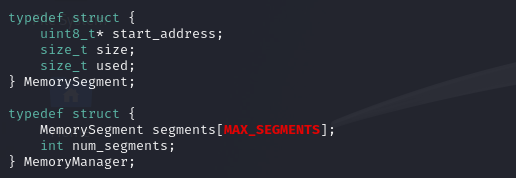
**Code Block**:



### 2. Structures Definition

**Description**: These define the structures used by the memory manager. Memory Segment represents a segment of memory with its start address, size, and how much of it is used. Memory Manager holds an array of Memory Segment and keeps track of the number of segments.

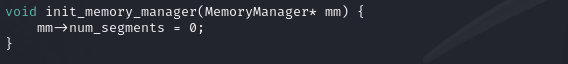
**Code Block**:



### 3. Memory Manager Initialization Function

**Description**: This function initializes the memory manager by setting the number of segments to zero.

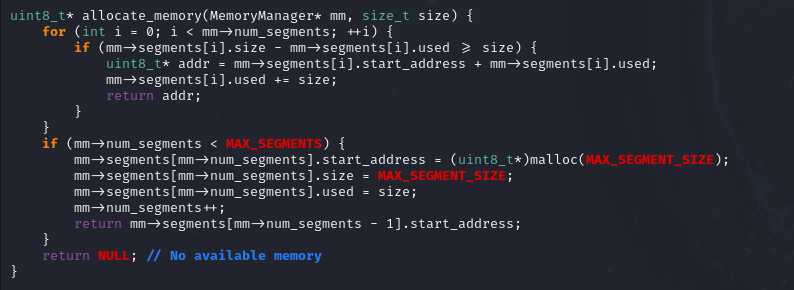
**Code Block**:



### 4. Memory Allocation Function

**Description**: This function allocates memory of the given size from the available segments. If there's not enough space in existing segments, it allocates a new segment.

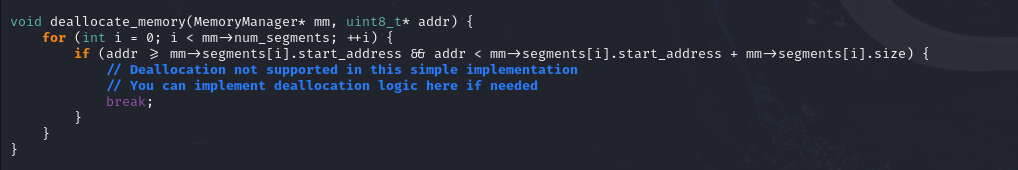
**Code Block**:



### 5. Memory Deallocation Function

**Description**: This function deallocates memory at the given address. (Not fully implemented in the provided code).

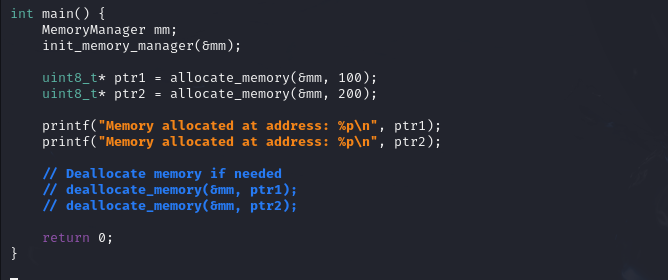
**Code Block**:



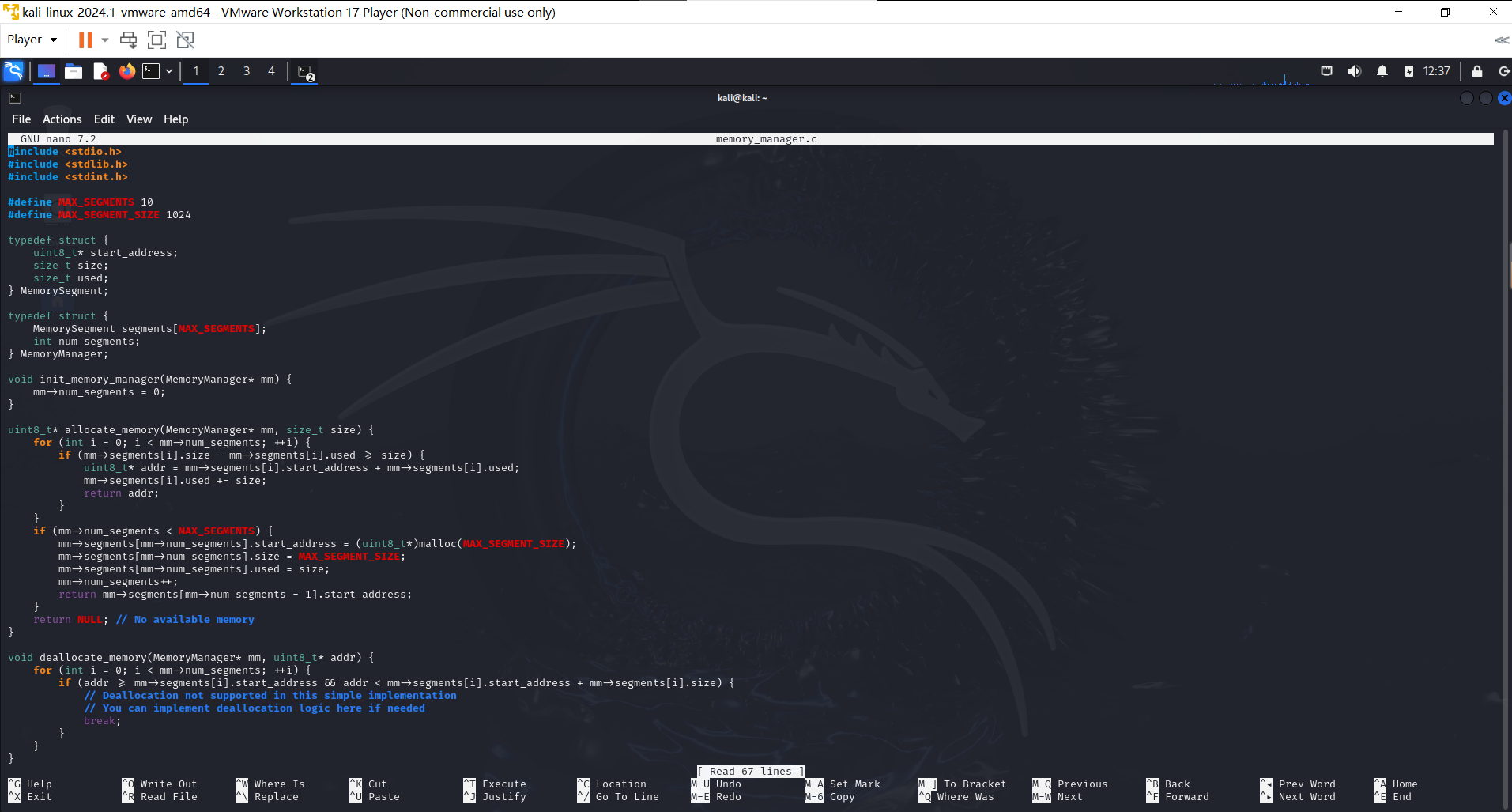
### 6. Main Function

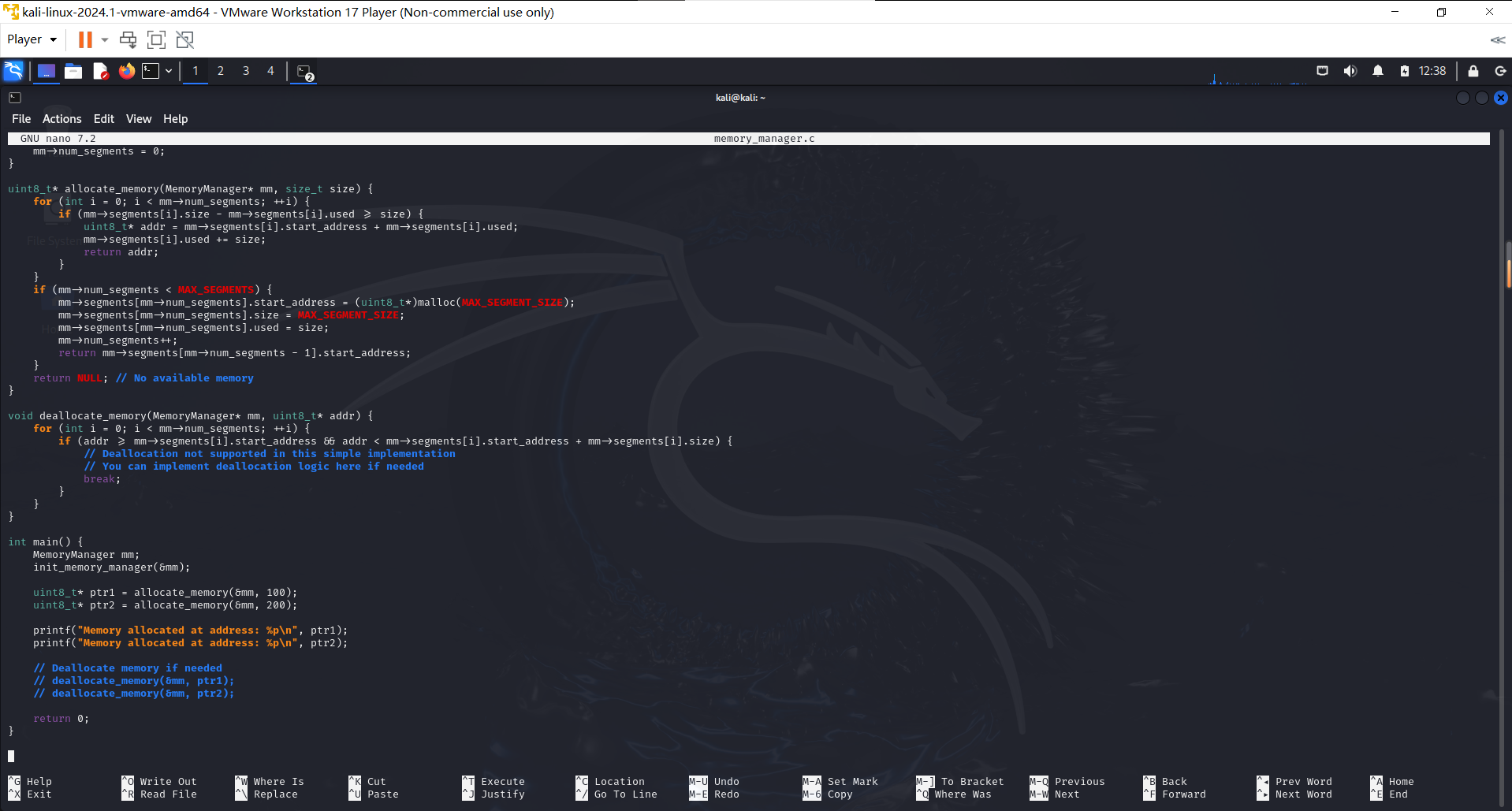
**Description**: This is the entry point of the program. It creates a MemoryManager instance, initializes it, allocates memory twice, prints the allocated memory addresses, and exits.

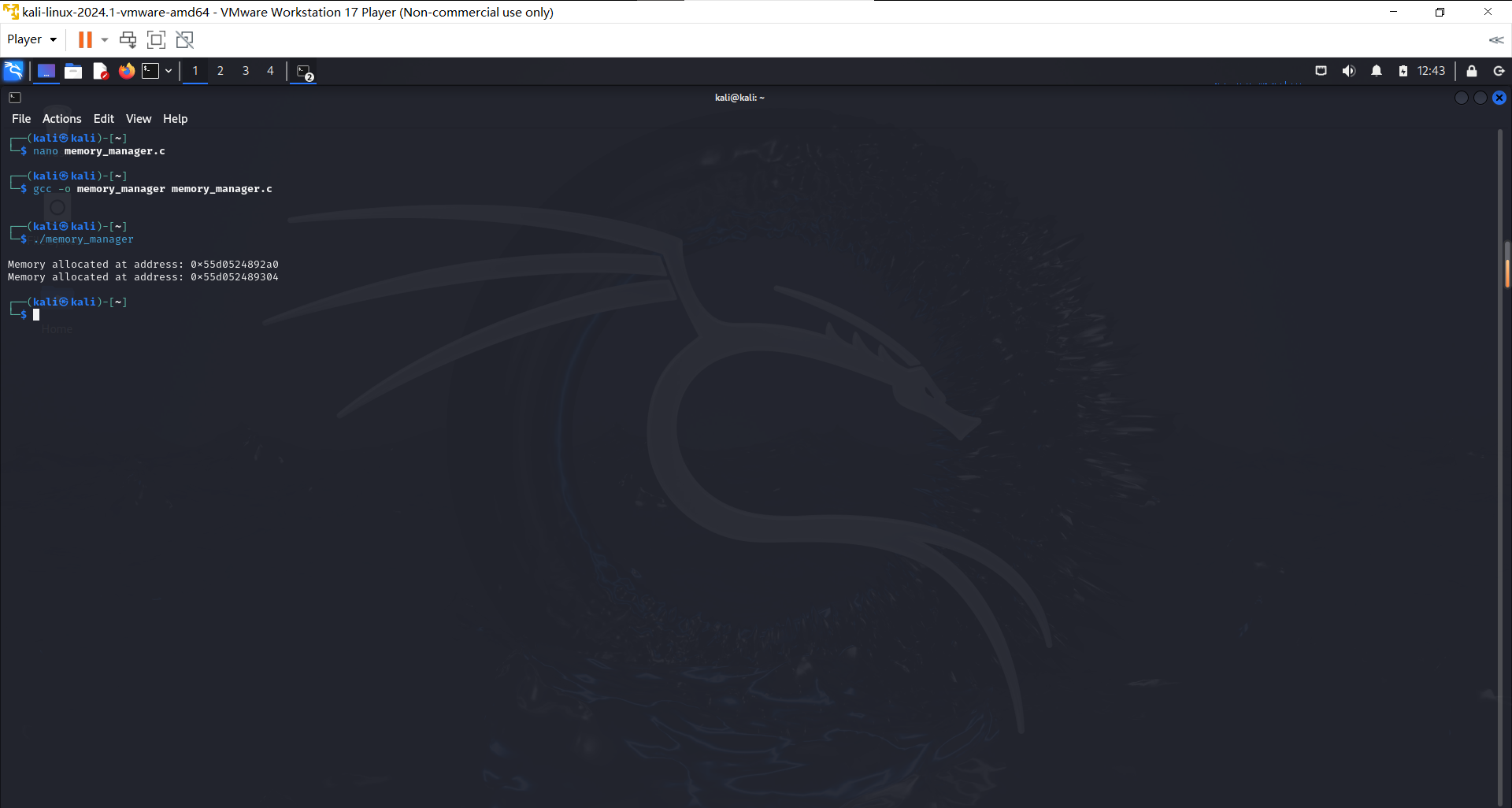
**Code Block**:



Complete code:







## Control questions

1. **What limits the maximum size of physical memory that can be installed in a particular model computer?**

The maximum size of physical memory that can be installed in a computer is limited by several factors:

- Motherboard design : The number of memory slots and the maximum capacity each slot can handle.

- Memory controller : The capabilities of the memory controller, which is often integrated into the CPU or chipset, determine the maximum amount of RAM.

- Operating system : The operating system's architecture (32-bit or 64-bit) and version can impose limits on the maximum addressable memory. For example, a 32-bit OS typically supports up to 4GB of RAM, while 64-bit versions can support much more.

**2. What limits the maximum size of virtual address space available to an application?**

The maximum size of the virtual address space available to an application is limited by:

- Processor architecture : 32-bit processors have a virtual address space of 4GB, while 64-bit processors can address a much larger space.

- Operating system : The OS may impose additional limits on the virtual address space for applications to manage system resources effectively.

- Application design : Some applications may be designed to use only a portion of the available virtual address space.

**3. In what cases does a translator create a program's object code in physical addresses rather than virtual addresses?**

A translator (e.g., compiler or assembler) creates a program's object code using physical addresses rather than virtual addresses in cases such as:

- Embedded systems : Where programs run directly on hardware without an operating system or memory management unit (MMU) to handle virtual addresses.

- Bootloaders : Early-stage programs that run before the OS initializes virtual memory.

- Real-time systems : Where deterministic performance is required, and virtual memory overhead is avoided.

**4. What is "swapping"?**

Swapping is a memory management technique where the operating system moves data from RAM to a swap file or swap partition on disk (and vice versa) to free up physical memory for other processes. This allows the system to handle more processes or larger data sets than can fit in physical memory alone.

**5. How does the size of a swap file affect system performance?**

The size of the swap file affects system performance in several ways:

- Insufficient swap space : If the swap file is too small, the system may run out of virtual memory, leading to application crashes or inability to open new programs.

- Excessive swapping : If the swap file is large, excessive swapping can occur, leading to high disk I/O and degraded performance, often referred to as "thrashing."

- Optimal size : An appropriately sized swap file can provide enough virtual memory to accommodate the system's workload without excessive swapping.

**6. Why is the page size chosen to be equal to a power of two? Can such a limitation be adopted for a segment?**

The page size is chosen to be a power of two for several reasons:

- Efficiency in address translation : Using powers of two simplifies the hardware implementation of address translation, as it can be done using bitwise operations.

- Alignment and boundary conditions : Powers of two ensure that pages align nicely on memory boundaries, simplifying memory allocation and management.

For segments, the limitation of power-of-two sizes is not adopted because segments can vary widely in size depending on the needs of the application. Segment sizes are often defined by the program's logical structure rather than hardware constraints.

**7. What is affected by the page size? What are the advantages and disadvantages of large page size?**

The page size affects:

- Memory management efficiency : Larger pages can reduce the overhead of page table entries and the frequency of page table lookups.

- Internal fragmentation : Larger pages can lead to more wasted space within each page if the data does not completely fill the page.

- TLB (Translation Lookaside Buffer) performance : Larger pages can reduce the number of entries needed in the TLB, potentially improving performance.

Advantages of large page size :

- Reduced page table size and overhead.

- Fewer page faults for large contiguous memory allocations.

Disadvantages of large page size :

- Increased internal fragmentation.

- Potentially higher latency in accessing smaller data sets that do not fill a large page.

**8. Why is the loading and unloading of data from cache memory done in blocks?**

Loading and unloading data from cache memory in blocks (cache lines) is done to exploit spatial locality. Spatial locality refers to the tendency of a program to access data locations that are close to each other within a short period. By transferring data in blocks, the cache can prefetch adjacent data that is likely to be used soon, thereby reducing the number of cache misses and improving overall performance.

**9. How is data in the cache reconciled using write-back and write-through methods?**

Write-back method :

- Modifications to data in the cache are not immediately written to the main memory. Instead, the data is marked as dirty, and only when it is evicted from the cache is it written back to the main memory.

- Advantages : Reduces the number of write operations to the main memory, potentially improving performance.

- Disadvantages : Data in the main memory may become stale, and additional complexity is required to manage dirty cache lines.

Write-through method :

- Modifications to data in the cache are immediately written to the main memory as well.

- Advantages : Ensures that the main memory always has the most up-to-date data, simplifying consistency.

- Disadvantages : Increases the number of write operations to the main memory, which can degrade performance.