| | | |
|---|---|---|
| **Code:** | **Name:** | **Class:** |

**Question 1. Memory management is a critical aspect of operating systems. It ensures that processes only access their permitted address space. Which method is used to protect a process's address space?**

a. Using a pair of base and limit registers to define the logical address space of a process, ensuring that all memory accesses fall within this range.

b. Allowing processes to directly access physical memory without checks, which speeds up processing but risks security vulnerabilities.

c. Using a fixed mapping table in main memory to convert logical addresses to physical addresses without requiring registers.

d. Relying on the operating system to automatically allocate memory without hardware support, leading to less control.

**Question 2. Base and limit registers are used to protect processes. These registers can only be loaded by privileged instructions. Who has the authority to execute privileged instructions to load base and limit registers?**

a. User processes can execute privileged instructions to self-adjust base and limit registers, enhancing flexibility.

b. Hardware automatically loads base and limit registers without operating system intervention, reducing CPU load.

c. Only the operating system is permitted to execute privileged instructions to load base and limit registers, ensuring that user processes cannot modify them.

d. Dynamically linked system libraries can access and modify base and limit registers during execution.

**Question 3. Addresses in the source program are typically symbolic. The compiler converts these addresses into relocatable addresses. What is the role of the linker or loader in address binding?**

a. The linker retains symbolic addresses in the program so they can be used directly in physical memory.

b. The loader automatically generates absolute code during the compilation phase, eliminating the need for relocatable addresses.

c. The linker only checks for syntax errors in the source code without altering address mapping.

d. The linker or loader converts relocatable addresses into absolute addresses, such as address 74014, to determine the exact location in physical memory.

**Question 4. Address binding can occur at three different stages. One of these is at compile time. What happens if the memory location is known in advance at compile time?**

a. Absolute code is generated, but if the starting location changes, the code must be recompiled to ensure accurate address mapping.

b. Relocatable code is generated, allowing the process to move in memory without requiring recompilation.

c. Address binding is deferred until runtime, requiring hardware support such as base registers.

d. The operating system automatically adjusts memory locations without any changes to the code.

**Question 5. Runtime address binding requires hardware support. This allows processes to move in memory during execution. What type of hardware is needed to support runtime address binding?**

a. The Translation Look-aside Buffer (TLB) is used to store the entire page table, eliminating the need for base registers.

b. A co-processor is used to manage address mapping without intervention from the main CPU.

c. Base and limit registers are used to map logical addresses to physical addresses, enabling process relocation in memory without disrupting execution.

d. Main memory is statically partitioned to hold processes without dynamic mapping.

**Question 6. The Memory Management Unit (MMU) is a critical hardware device. It maps logical addresses to physical addresses at runtime. What is the role of the MMU in this process?**

a. The MMU adds the value in the relocation register to every logical address generated by a user process, converting them to physical addresses before sending them to memory.

b. The MMU stores the entire page table in main memory and accesses it without needing a relocation register.

c. The MMU only checks for syntax errors in logical addresses without mapping them to physical addresses.

d. The MMU automatically allocates physical memory without mapping logical addresses, reducing memory efficiency.

**Question 7. Dynamic Loading allows a program to execute without loading the entire program into memory upfront. Routines are loaded only when called. What is the primary benefit of dynamic loading?**

a. Dynamic loading requires the entire program to be loaded into memory initially, increasing execution speed but consuming more memory.

b. Dynamic loading automatically links system libraries into the program code at compile time, reducing runtime overhead.

c. Dynamic loading requires the operating system to provide specialized hardware for managing routine loading, increasing complexity.

d. Dynamic loading improves memory space utilization by loading routines only when needed, avoiding the loading of unused routines, which is particularly useful for infrequent cases.

**Question 8. Dynamic Linking postpones library linking until execution time. A small piece of code called a stub is used in this process. What is the function of the stub in dynamic linking?**

a. The stub creates a fixed address mapping table at compile time, eliminating the need for address checks during runtime.

b. The stub stores the entire library in main memory as soon as the program starts, increasing execution speed.

c. The stub locates the library routine in memory, replaces itself with the routine's address, and executes the routine, ensuring the library is loaded only when needed.

d. The stub prevents the operating system from checking if the routine is in the process's address space, leading to security vulnerabilities.

**Question 9. Dynamic linking is particularly useful for shared libraries. The operating system plays a critical role in managing these libraries. What does the operating system do when a routine is not in a process's address space?**

a. The operating system requires the process to reload the entire program, interrupting execution and reducing performance.

b. The operating system checks and adds the routine to the process's address space if it is not already present, ensuring the process can access the required library during runtime.

c. The operating system ignores the routine and continues execution, leading to unpredictable program errors.

d. The operating system moves the routine to swap space to free up main memory, slowing down execution.

**Question 10. Contiguous memory allocation is a technique used in simple operating systems. It requires each process to be contained in a single contiguous section of memory. How is the relocation register used in contiguous memory allocation?**

a. The relocation register stores the size of the entire physical memory, allowing the operating system to allocate memory without limit checks.

b. The relocation register holds the value of the smallest physical address, used alongside the limit register to protect processes from accessing each other or operating system code.

c. The relocation register automatically partitions memory into fixed blocks, eliminating the need for a limit register.

d. The relocation register is only used to store logical addresses, not participating in process protection.

**Question 11. In contiguous memory allocation, main memory is typically divided into two partitions. One partition is for the operating system, and the other is for user processes. Where is the operating system typically located in memory?**

a. The operating system is usually held in low memory along with the interrupt vector, ensuring it can quickly handle interrupt requests and manage system resources.

b. The operating system is stored in high memory to avoid conflicts with user processes, reducing access efficiency.

c. The operating system is allocated randomly in memory, depending on the size of user processes.

d. The operating system is stored in secondary memory and loaded only when needed, slowing down execution.

**Question 12. Contiguous memory allocation uses base and limit registers to protect processes. The MMU dynamically maps logical addresses. How does the MMU perform address mapping in contiguous allocation?**

a. The MMU stores logical addresses directly in main memory without mapping to physical addresses, increasing the risk of errors.

b. The MMU uses a page table for address mapping, even in simple contiguous allocation systems.

c. The MMU only checks the size of the process without mapping addresses, leading to inefficient memory use.

d. The MMU adds the value in the relocation register to the logical address and checks if it falls within the range of the limit register, ensuring valid access.

**Question 13. Memory allocation uses holes to allocate memory for processes. When a process terminates, its memory is freed and can be merged. What information does the operating system maintain to manage memory?**

a. The operating system tracks allocated partitions and free holes, allowing efficient memory allocation and merging of adjacent free partitions when a process terminates.

b. The operating system only stores information about process sizes without tracking free holes, reducing allocation efficiency.

c. The operating system uses a fixed table to map all logical addresses, even when memory is free.

d. The operating system does not maintain any memory information, relying entirely on hardware for management.

**Question 14. In memory allocation, the First-fit strategy is used to allocate memory. It is simple but effective in many cases. How does the First-fit strategy work?**

a. First-fit always selects the smallest hole large enough for the process, minimizing memory waste but requiring a full list search.

b. First-fit selects the largest hole in the list, resulting in larger free holes but slowing down allocation.

c. First-fit allocates the first hole large enough to accommodate the process, reducing search time and simplifying the allocation process.

d. First-fit requires the operating system to relocate all processes to create a large enough hole, increasing processing overhead.

**Question 15. The Best-fit strategy is used to optimize memory usage. However, it can be more time-consuming than First-fit. How does Best-fit work?**

a. Best-fit allocates the smallest hole large enough for the process, minimizing memory waste but requiring a search of the entire list of holes.

b. Best-fit selects the first hole large enough without further searching, speeding up allocation but potentially creating larger holes.

c. Best-fit selects the largest hole to ensure sufficient space, but it reduces memory utilization efficiency.

d. Best-fit requires memory compaction before allocation, significantly increasing computational overhead.

**Question 16. The Worst-fit strategy aims to leave the largest possible holes after allocation. However, it is not always efficient. How does Worst-fit work?**

a. Worst-fit selects the smallest hole large enough for the process, minimizing memory waste but requiring time-consuming searches.

b. Worst-fit allocates the largest hole in the list, creating larger free holes for future use, but it requires searching the entire list and is often less efficient than First-fit or Best-fit.

c. Worst-fit allocates the first hole large enough without further searching, speeding up allocation but not optimizing space.

d. Worst-fit requires the operating system to compact memory before allocation, increasing processing overhead.

**Question 17. External Fragmentation occurs in contiguous memory allocation. It causes issues when the total free memory is sufficient but not contiguous. What is external fragmentation?**

a. External fragmentation occurs when allocated memory is larger than requested, leading to wasted space within a partition.

b. External fragmentation is when the operating system cannot find any free memory space, even when memory is abundant.

c. External fragmentation occurs when the total free memory space is sufficient to meet a process's request, but the memory blocks are not contiguous, preventing allocation.

d. External fragmentation occurs when processes are continuously relocated in memory, increasing compaction overhead.

**Question 18. Internal Fragmentation is another issue in memory management. It occurs when allocated memory is not fully utilized. What is internal fragmentation?**

a. Internal fragmentation occurs when the total free memory is sufficient but not contiguous, preventing allocation.

b. Internal fragmentation occurs when memory allocated to a process is larger than required, resulting in unused memory within the allocated partition.

c. Internal fragmentation is when the operating system relocates processes to merge free holes, increasing processing overhead.

d. Internal fragmentation occurs when dynamically linked libraries are not loaded correctly, leading to execution errors.

**Question 19. Compaction is a solution to external fragmentation. However, it is not always feasible. What condition prevents compaction from being performed?**

a. Compaction is always feasible, regardless of the relocation method, as the operating system can automatically adjust addresses.

b. Compaction is only feasible when all processes are stored in secondary memory, increasing access overhead.

c. Compaction is prevented when the operating system uses page tables instead of relocation registers, reducing efficiency.

d. If relocation is static and performed at assembly or load time, compaction cannot be done because addresses are fixed, making process relocation impossible.

**Question 20. Paging is an advanced memory management technique. It allows a process's physical address space to be non-contiguous. What is the primary benefit of paging?**

a. Paging requires all processes to be stored in contiguous partitions, increasing memory utilization efficiency.

b. Paging automatically compacts memory when external fragmentation occurs, increasing processing overhead but ensuring contiguous space.

c. Paging only works when the entire program is loaded into memory, leading to inefficient memory usage.

d. Paging avoids external fragmentation and eliminates the need for compaction, enabling flexible memory allocation by dividing memory into fixed-size blocks called frames.

**Question 21. In paging, physical memory is divided into frames. Logical memory is divided into pages of the same size. What is the purpose of the page table in paging?**

a. The page table stores the entire content of logical pages, increasing storage overhead and reducing performance.

b. The page table is only used to check for syntax errors in logical addresses, not participating in address mapping.

c. The page table completely replaces the relocation register, eliminating the need for hardware support in paging.

d. The page table maps logical page numbers to physical frame numbers, allowing the operating system to efficiently translate logical addresses to physical addresses during execution.

**Question 22. Logical addresses in paging are divided into two parts. The page number and offset are used to determine the physical address. How is the page number used in the address translation process?**

a. The page number is added directly to the relocation register to form the physical address without needing a page table.

b. The page number is used as an index to look up the corresponding frame number in the page table, which is then combined with the offset to form the complete physical address.

c. The page number is only used to check if the logical address is valid, not participating in mapping.

d. The page number is stored in secondary memory and accessed only when a TLB miss occurs, slowing down translation.

**Question 23. Paging can still cause internal fragmentation. This occurs when the page size does not perfectly match the process’s requirements. How does internal fragmentation occur in paging?**

a. Internal fragmentation occurs when memory allocated for a page is larger than the process’s required size, resulting in unused memory within the allocated frame.

b. Internal fragmentation occurs when memory frames are not contiguous, preventing allocation despite sufficient space.

c. Internal fragmentation is when the operating system relocates pages to merge free frames, increasing processing overhead.

d. Internal fragmentation occurs when the page table is not updated correctly, leading to address mapping errors.

**Question 24. In paging, the page table is stored in main memory. This can slow down memory access. What is the main issue when the page table is stored in main memory?**

a. Each data or instruction access requires two memory accesses: one to look up the page table and one to retrieve the data or instruction, increasing processing time.

b. The page table occupies the entire main memory, reducing available space for user processes.

c. The page table is only accessed when a TLB miss occurs, making it irrelevant to overall performance.

d. The page table requires the operating system to compact memory before each access, increasing computational overhead.

**Question 25. The Translation Look-aside Buffer (TLB) is a specialized hardware cache. It is used to reduce address translation time in paging. How does the TLB address the memory access issue?**

a. The TLB stores the entire page table, eliminating the need to keep the page table in main memory, but increasing hardware costs.

b. The TLB is only used to check for syntax errors in logical addresses, not participating in address translation.

c. The TLB stores recently used address mappings, allowing quick frame number lookups without accessing the page table in main memory, reducing memory accesses to one in case of a TLB hit.

d. The TLB requires the operating system to clear its entire contents after each access, reducing performance.

**Question 26. Some TLBs store Address-Space Identifiers (ASIDs). ASIDs help protect the address space of processes. How are ASIDs used in the TLB?**

a. ASIDs store the size of the page table, allowing the TLB to automatically adjust its storage size.

b. ASIDs are only used to check for syntax errors in logical addresses, not related to process protection.

c. ASIDs uniquely identify each process, ensuring that the TLB only uses entries corresponding to the current process, preventing access to another process’s data and enhancing security.

d. ASIDs require the TLB to clear its entire contents during each context switch, reducing performance.

**Question 27. If the TLB does not use ASIDs, an issue may arise during context switching. This affects process security. What happens if the TLB does not use ASIDs?**

a. The TLB automatically adjusts its entries to match the new process without flushing, increasing performance but causing security vulnerabilities.

b. The TLB requires no changes during context switches, allowing processes to access each other’s data.

c. The TLB only stores fixed entries, unaffected by context switches, reducing flexibility.

d. The TLB must be flushed during each context switch to prevent the new process from accessing the old process’s data, ensuring security but reducing performance due to reloading TLB entries.

**Question 28. TLBs are typically small, containing 64 to 1,024 entries. This ensures fast access. What happens when a TLB miss occurs?**

a. When a TLB miss occurs, the system bypasses the page table and accesses physical memory directly, leading to address mapping errors.

b. When a TLB miss occurs, the system looks up the mapping in the page table in main memory and loads it into the TLB for faster future access, though this process is slower than a TLB hit.

c. When a TLB miss occurs, the TLB automatically clears its entire contents, significantly reducing performance.

d. When a TLB miss occurs, the system requires memory compaction to make space for new entries, increasing processing overhead.

**Question 29. Some TLB entries can be “wired down.” This ensures they are not replaced. What is the purpose of wiring down TLB entries?**

a. Wired-down TLB entries ensure permanent fast access for critical addresses, such as those of the operating system, by preventing them from being replaced by new entries.

b. Wired-down TLB entries store the entire page table, eliminating the need for main memory lookups.

c. Wired-down TLB entries are only used to check for syntax errors, not related to performance.

d. Wired-down TLB entries require the operating system to periodically clear the TLB, reducing performance.

**Question 30. In paging, logical addresses are divided into page numbers and offsets. The page size affects this division. If the logical address space size is 2^m and the page size is 2^n, how is the page number determined?**

a. All m bits of the logical address are used as the page number, while the offset is ignored during mapping.

b. The high-order n bits of the logical address are used as the page number, leading to address mapping errors.

c. The high-order m-n bits of the logical address designate the page number, while the n low-order bits designate the offset, ensuring accurate mapping of logical addresses to physical addresses via the page table.

d. The page number is determined by directly adding m and n, increasing the address space size.

**Question 31. Paging requires cooperation between hardware and the operating system. This ensures accurate address mapping. What is the role of the operating system in paging?**

a. The operating system only checks for syntax errors in logical addresses, not participating in page table setup.

b. The operating system requires hardware to automatically create the page table without intervention, leading to mapping errors.

c. The operating system stores the page table in secondary memory, slowing down memory access.

d. The operating system sets up and maintains the page table, allocates free frames to process pages, and ensures logical addresses are accurately mapped to physical addresses via the MMU.

**Question 32. The Page-Table Base Register (PTBR) is used in paging. It helps locate the page table in memory. What is the function of the PTBR?**

a. The PTBR points to the location of the page table in main memory, allowing the MMU to quickly look up the frame number corresponding to a page number during address translation.

b. The PTBR stores the entire content of the page table, eliminating the need to keep the page table in main memory.

c. The PTBR is only used to check the size of the page table, not participating in address translation.

d. The PTBR requires the operating system to compact the page table before each lookup, increasing processing overhead.

**Question 33. The Page-Table Length Register (PTLR) is used alongside the PTBR. It ensures the page table does not exceed its designated size. What is the role of the PTLR in paging?**

a. The PTLR stores the frame numbers, eliminating the need for page table lookups.

b. The PTLR specifies the size of the page table, ensuring the MMU only accesses valid entries in the page table during logical address translation, preventing memory access errors.

c. The PTLR is only used to check for syntax errors in logical addresses, not related to page table size.

d. The PTLR requires the operating system to periodically clear the page table, reducing performance.

**Question 34. In paging, frame sizes are typically powers of 2. This ensures compatibility with hardware addressing. What is the typical range of frame sizes?**

a. Frame sizes are always fixed at 1 Mbyte, regardless of process requirements, reducing flexibility.

b. Frame sizes are determined randomly, leading to address mapping errors in paging.

c. Frame sizes typically range from 512 bytes to 16 Mbytes, as powers of 2, ensuring pages and frames can be efficiently mapped without alignment errors.

d. Frame sizes are only used in contiguous allocation, not relevant to paging.

**Question 35. Paging allows running a program of size N pages. The system needs to find free frames to load the program. How many frames does the system need to run a program of size N pages?**

a. The system only needs one frame, regardless of program size, to store the entire program.

b. The system needs N/2 frames, as pages are compressed before loading into memory.

c. The system does not need any frames, as pages are stored directly in secondary memory.

d. The system needs to find N free frames to load the program, as each logical page of the program must be mapped to a separate physical frame in main memory.

**Question 36. In paging, logical addresses are divided into page numbers and offsets. Address translation requires looking up the page table. What is the first step in the address translation process?**

a. The MMU adds the offset directly to the relocation register to form the physical address, bypassing the page table.

b. The MMU extracts the page number from the logical address and uses it as an index to look up the corresponding frame number in the page table, proceeding with the mapping process.

c. The MMU checks for syntax errors in the logical address before looking up the page table, slowing down translation.

d. The MMU requires the operating system to compact the page table before lookup, increasing processing overhead.

**Question 37. Paging helps avoid external fragmentation. However, it still has some limitations. What is the primary limitation of paging?**

a. Paging requires all frames to be contiguous, leading to external fragmentation similar to contiguous allocation.

b. Paging requires periodic memory compaction, increasing processing overhead and reducing performance.

c. Paging does not allow mapping logical addresses to physical addresses, leading to execution errors.

d. Paging still causes internal fragmentation, as memory allocated for a page may be larger than the process's required size, resulting in wasted space within the frame.

**Question 38. In paging, the physical address is formed by combining the frame number and offset. This process is performed by the MMU. How does the MMU create the physical address?**

a. The MMU adds the page number directly to the relocation register to form the physical address, bypassing the page table.

b. The MMU replaces the page number in the logical address with the frame number from the page table, then combines it with the offset to form the complete physical address sent to memory.

c. The MMU only checks for syntax errors in the frame number, not participating in physical address creation.

d. The MMU requires the operating system to compact memory before creating the physical address, increasing processing overhead.

**Question 39. Paging uses the page table to map addresses. The page table is stored in main memory and accessed by the MMU. Who is responsible for setting up the page table?**

a. The operating system sets up the page table, mapping the process's logical pages to physical frames, and updates the PTBR so the MMU can access the page table during address translation.

b. The MMU automatically sets up the page table without operating system intervention, increasing the risk of mapping errors.

c. The TLB sets up the page table and stores it for fast access, eliminating the need for main memory.

d. The relocation register sets up the page table, reducing the operating system's role in paging.

**Question 40. Paging requires cooperation between hardware and the operating system. Hardware ensures fast and accurate address mapping. What is the role of hardware in paging?**

a. Hardware, including the MMU and TLB, performs real-time mapping of logical addresses to physical addresses, using the page table set up by the operating system to ensure accurate memory access.

b. Hardware only checks for syntax errors in logical addresses, not participating in address mapping.

c. Hardware automatically sets up the page table without operating system intervention, leading to mapping errors.

d. Hardware requires the operating system to compact memory before each mapping, increasing processing overhead.

**Question 41. The TLB improves paging performance. It stores recently used address mappings. What happens when a required entry is not in the TLB?**

a. When a TLB miss occurs, the TLB automatically clears its entire contents, significantly reducing performance.

b. When a TLB miss occurs, the system looks up the mapping in the page table in main memory and loads it into the TLB for future use, though this process is slower than a TLB hit.

c. When a TLB miss occurs, the system bypasses the page table and accesses physical memory directly, leading to mapping errors.

d. When a TLB miss occurs, the system requires memory compaction to make space for new entries, increasing processing overhead.

**Question 42. In paging, page sizes are powers of 2. This ensures compatibility with hardware. Why are page sizes powers of 2?**

a. Page sizes are powers of 2 to require all frames to be contiguous, increasing external fragmentation.

b. Page sizes are powers of 2 to store the entire page table in the TLB, eliminating the need for main memory.

c. Page sizes are powers of 2 to require periodic memory compaction, increasing processing overhead.

d. Page sizes are powers of 2 to ensure logical addresses can be efficiently divided into page numbers and offsets, enabling accurate address mapping without complex alignment.

**Question 43. Paging uses fixed-size frames for memory allocation. This simplifies memory management. What is the benefit of using fixed-size frames?**

a. Fixed-size frames enable flexible memory allocation, avoiding external fragmentation by mapping logical pages to any free frame, regardless of their location in physical memory.

b. Fixed-size frames require all processes to be of equal size, reducing flexibility.

c. Fixed-size frames are only used in contiguous allocation, not relevant to paging.

d. Fixed-size frames require periodic memory compaction, increasing processing overhead.

**Question 44. Paging requires the operating system to track free frames. This ensures processes can be loaded into memory. How does the operating system manage free frames?**

a. The operating system requires hardware to automatically manage free frames without intervention, leading to allocation errors.

b. The operating system maintains a list of free frames, allocates them to process pages as needed, and updates the page table to map logical pages to these frames.

c. The operating system only checks process sizes, not tracking free frames, reducing allocation efficiency.

d. The operating system stores free frames in secondary memory, slowing down memory allocation.

**Question 45. In paging, logical addresses are mapped to physical addresses via the page table. This process is performed by the MMU. How does the MMU perform address mapping in paging?**

a. The MMU adds the page number directly to the relocation register to form the physical address, bypassing the page table.

b. The MMU only checks for syntax errors in logical addresses, not participating in address mapping.

c. The MMU extracts the page number from the logical address, looks up the frame number in the page table, and combines the frame number with the offset to form the physical address sent to memory.

d. The MMU requires the operating system to compact memory before mapping, increasing processing overhead.

**Question 46. Paging uses the page table for address mapping. However, accessing the page table can slow performance. How does the TLB improve performance in paging?**

a. The TLB stores recently used address mappings, allowing the MMU to quickly look up frame numbers without accessing the page table in main memory, reducing address translation time.

b. The TLB stores the entire page table, eliminating the need to keep the page table in main memory, but increasing hardware costs.

c. The TLB is only used to check for syntax errors in logical addresses, not related to performance.

d. The TLB requires the operating system to clear its entire contents after each access, reducing performance.

**Question 47. Paging requires cooperation between the operating system and hardware. The operating system sets up the page table, while hardware performs address mapping. What is the role of the MMU in paging?**

a. The MMU only checks for syntax errors in logical addresses, not participating in address mapping.

b. The MMU automatically sets up the page table without operating system intervention, leading to mapping errors.

c. The MMU requires the operating system to compact memory before each mapping, increasing processing overhead.

d. The MMU performs real-time mapping of logical addresses to physical addresses, using the page table set up by the operating system to ensure accurate and efficient memory access.

**Question 48. Paging allows non-contiguous physical address spaces. This addresses issues of contiguous allocation. What is the primary benefit of allowing non-contiguous address spaces?**

a. Non-contiguous address spaces enable flexible memory allocation, avoiding external fragmentation by mapping logical pages to any free frame, regardless of their location in physical memory.

b. Non-contiguous address spaces require all processes to be of equal size, reducing flexibility.

c. Non-contiguous address spaces are only used in contiguous allocation, not relevant to paging.

d. Non-contiguous address spaces require periodic memory compaction, increasing processing overhead.

**Question 49. In paging, the page table is stored in main memory. This can introduce performance overhead. How can the overhead of accessing the page table be reduced?**

a. Storing the page table in secondary memory frees up main memory but slows down access.

b. Using a TLB to store recently used address mappings allows quick frame number lookups without accessing the page table in main memory, reducing the number of memory accesses.

c. Requiring the operating system to compact the page table before each lookup increases processing overhead.

d. Eliminating the page table and relying solely on the relocation register leads to address mapping errors.

**Question 50. Paging uses fixed-size frames for memory allocation. This simplifies the memory management process. Why is using fixed-size frames efficient?**

a. Fixed-size frames require all processes to be of equal size, reducing flexibility.

b. Fixed-size frames are only used in contiguous allocation, not relevant to paging.

c. Fixed-size frames allow the operating system to allocate memory flexibly, mapping logical pages to any free frame, avoiding external fragmentation and eliminating the need for memory compaction.

d. Fixed-size frames require periodic memory compaction, increasing processing overhead.