Lecture 49
Recovery – Shadow Paging

Shadow Paging

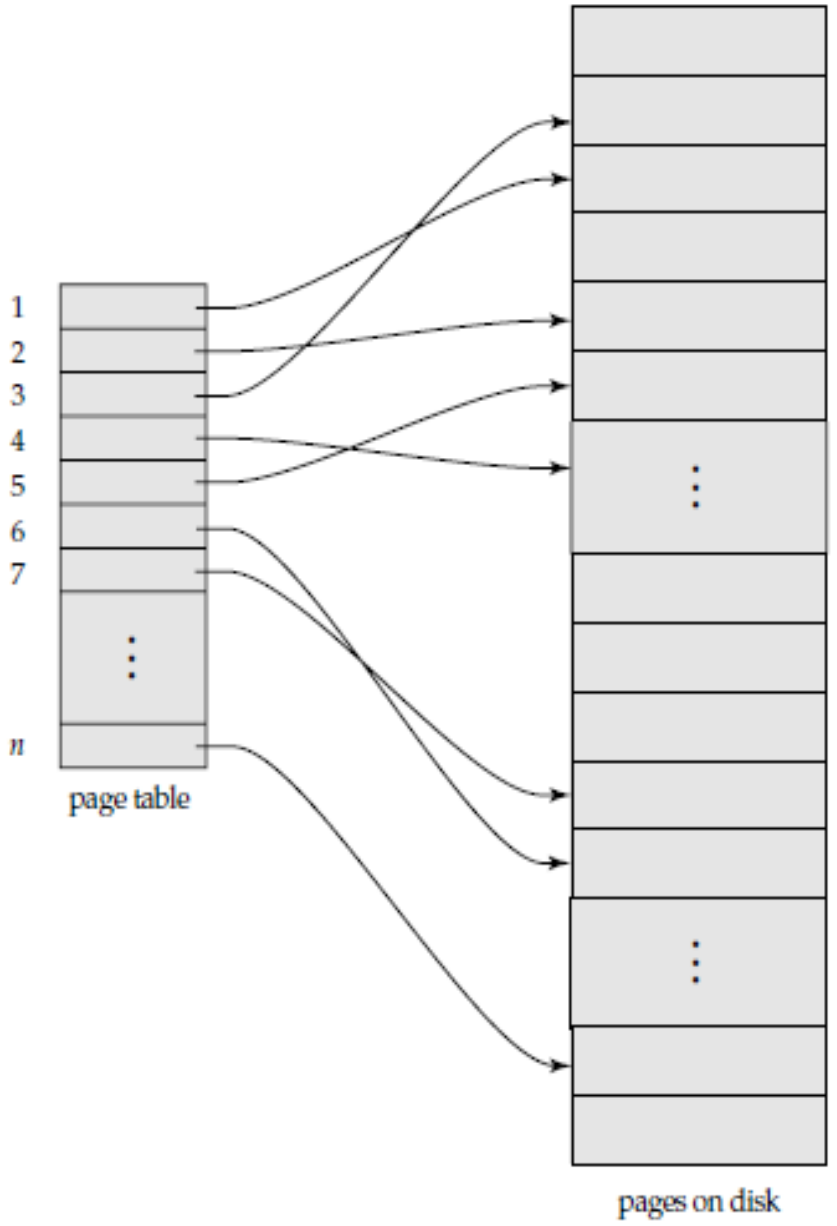
- An alternative to log-based crash-recovery techniques is **shadow paging**.
- The shadow-paging technique is essentially an improvement on the shadow-copy technique.
- Under certain circumstances, shadow paging may require fewer disk accesses than do the log-based methods.
- There are, however, disadvantages to the shadow-paging approach, as we shall see, that limit its use.
- For example, it is hard to extend shadow paging to allow multiple transactions to execute concurrently.

Shadow Paging

- The database is partitioned into some number of fixed-length blocks, which are referred to as **pages**.
- The term *page* is borrowed from operating systems, since we are using a paging scheme for memory management.
- Assume that there are *n* pages, numbered 1 through *n*.
- These pages do not need to be stored in any particular order on disk.
- However, there must be a way to find the i^{th} page of the database for any given *i*.

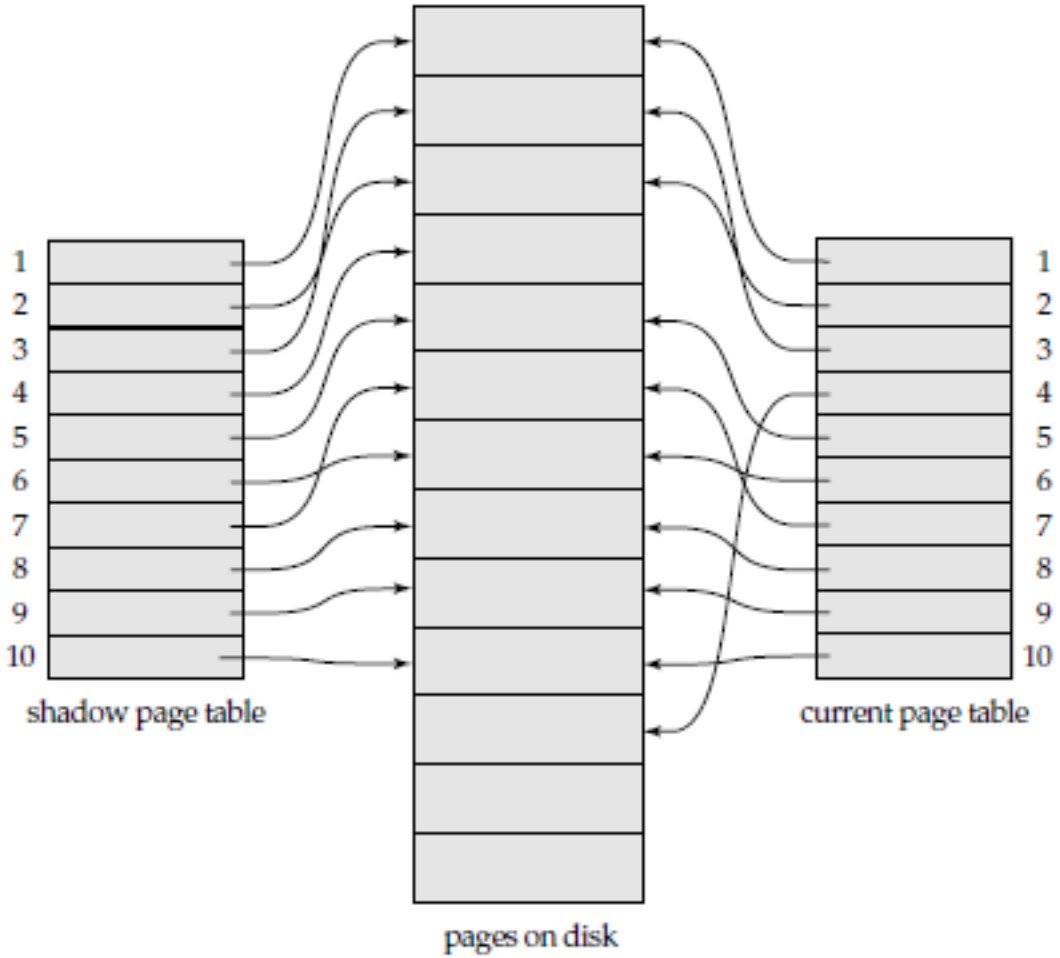
Shadow Paging

- We use a **page table**, for this purpose. The page table has *n* entries—one for each database page.
- Each entry contains a pointer to a page on disk. The first entry contains a pointer to the first page of the database, the second entry points to the second page, and so on.
- The example in Figure shows that the logical order of database pages does not need to correspond to the physical order in which the pages are placed on disk.



Shadow Paging

- The key idea behind the shadow-paging technique is to maintain *two* page tables during the life of a transaction: the **current page table** and the **shadow page table**.
- When the transaction starts, both page tables are identical.
- The shadow page table is never changed over the duration of the transaction.
- The current page table may be changed when a transaction performs a write operation.
- All input and output operations use the current page table to locate database pages on disk.



Shadow Paging

- Suppose that the transaction T_j performs a write(X) operation, and that X resides on the i^{th} page. The system executes the write operation as follows:
  1. If the i^{th} page (that is, the page on which X resides) is not already in main memory, then the system issues input(X).
  2. If this is the write first performed on the i^{th} page by this transaction, then the system modifies the current page table as follows:
     a) It finds an unused page on disk. Usually, the database system has access to a list of unused (free) pages,
     b) It deletes the page found in step 2a from the list of free page frames; it copies the contents of the i^{th} page to the page found in step 2a.
     c) It modifies the current page table so that the i^{th} entry points to the page found in step 2a.
  3. It assigns the value of x_j to X in the buffer page.

Shadow Paging

- Intuitively, the shadow-page approach to recovery is to store the shadow page table in nonvolatile storage, so that the state of the database prior to the execution of the transaction can be recovered in the event of a crash, or transaction abort.
- When the transaction commits, the system writes the current page table to nonvolatile storage. The current page table then becomes the new shadow page table, and the next transaction is allowed to begin execution.
- It is important that the shadow page table be stored in nonvolatile storage, since it provides the only means of locating database pages.
- The current page table may be kept in main memory (volatile storage).
- We do not care whether the current page table is lost in a crash, since the system recovers by using the shadow page table.

Shadow Paging

- Successful recovery requires that we find the shadow page table on disk after a crash.
- A simple way of finding it is to choose one fixed location in stable storage that contains the disk address of the shadow page table.
- When the system comes back up after a crash, it copies the shadow page table into main memory and uses it for subsequent transaction processing.
- Because of our definition of the write operation, we are guaranteed that the shadow page table will point to the database pages corresponding to the state of the database prior to any transaction that was active at the time of the crash. Thus, aborts are automatic.
- Unlike our log-based schemes, shadow paging needs to invoke no undo operations.

Shadow Paging

- To commit a transaction, we must do the following :
  1. Ensure that all buffer pages in main memory that have been changed by the transaction are output to disk. (Note that these output operations will not change database pages pointed to by some entry in the shadow page table.)
  2. Output the current page table to disk. Note that we must not overwrite the shadow page table, since we may need it for recovery from a crash.
  3. Output the disk address of the current page table to the fixed location in stable storage containing the address of the shadow page table. This action overwrites the address of the old shadow page table. Therefore, the current page table has become the shadow page table, and the transaction is committed.
- If a crash occurs prior to the completion of step 3, we revert to the state just prior to the execution of the transaction.
- If the crash occurs after the completion of step 3, the effects of the transaction will be preserved; no redo operations need to be invoked.

Shadow Paging

- Shadow paging offers several **advantages** over log-based techniques.
- The overhead of log-record output is eliminated, and recovery from crashes is significantly faster (since no undo or redo operations are needed).

Shadow Paging

- However, there are **drawbacks** to the shadow-page technique:
  1. **Commit overhead**. The commit of a single transaction using shadow paging requires multiple blocks to be output—the actual data blocks, the current page table, and the disk address of the current page table. Log-based schemes need to output only the log records, which, for typical small transactions, fit within one block.
  2. **Data fragmentation**. Shadow paging causes database pages to change location when they are updated. As a result, either we lose the locality property of the pages or we must resort to more complex, higher-overhead schemes for physical storage management.
  3. **Garbage collection**. Each time that a transaction commits, the database pages containing the old version of data changed by the transaction become inaccessible.

For Video lecture on this topic please subscribe to my youtube channel.

The link for my youtube channel is

https://www.youtube.com/channel/UCRWGtE76JlTp1iim6aOTRuw?sub_confirmation=1