Operating System Support for Database Management

Michael Stonebraker
University of California, Berkely

Instructor: P.Krishna Reddy

Introduction

- What do Operating Systems do that force DBMS designers to start from scratch? Can the OS take over some of this stuff?
- **Note**: Some of these gripes have been fixed in commercial OS'es. Some in research OS'es. Many remain as artifacts of the distinction between OS'es and DBMSs.

Introduction…

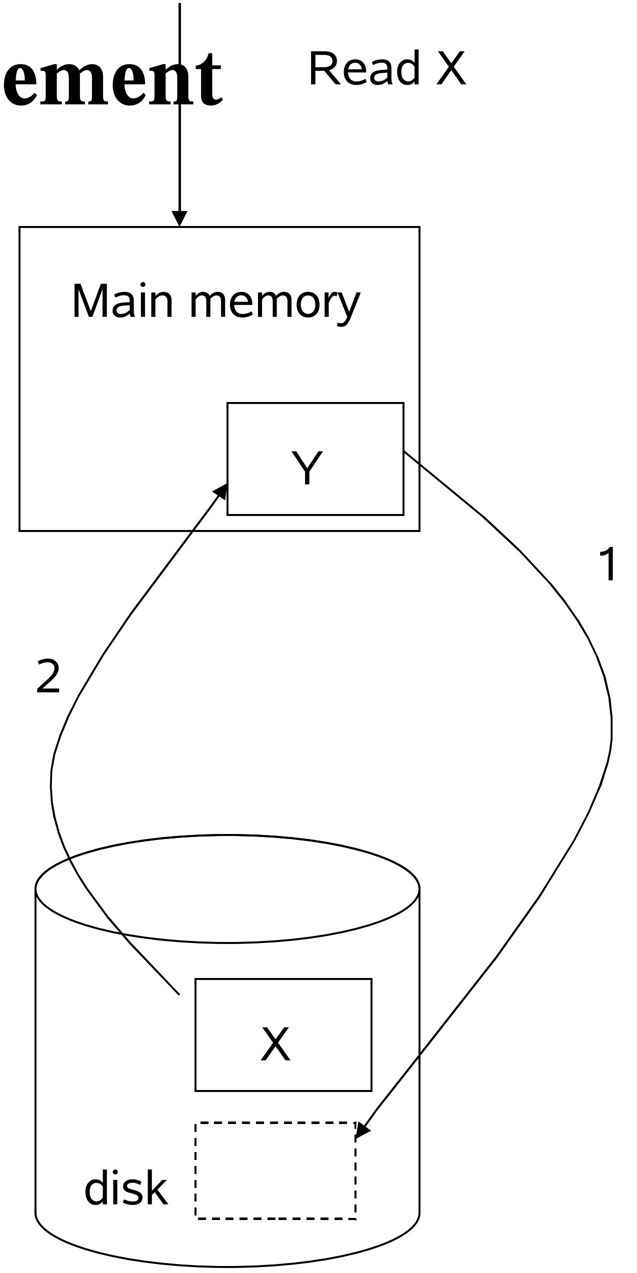
- DBMS provide higher level user support than conventional OSs.
- Different OSs are designed for different use.
- We discuss several OS services and indicate whether they are appropriate for DBMS.
  - Conclusion: Wrong service is provided or severe performance problems exists.
  - Some suggestions are provided.

Outline

- Examples are drawn from UNIX, but the points raised have a general applicability.
- Several popular OS services are examined.
  - **Buffer pool management**
  - The file system
  - Scheduling and IPC
  - Consistency control
- Conclusions

Buffer Pool Management

- Many OSs provide main memory cache for the file system.
  - Example: UNIX provides a buffer pool whose size is set when the OS is compiled.
  - A file-read returns data directly from a block in the cache, otherwise, the block is pushed to the disk and replaced by desired block.
  - Conceptually it is desirable because, the blocks of locality-of-reference remain in the disk.
  - However, several problems are encountered in case of DBMS.
    - Performance, replacement policy control, pre-fetch and crash recovery



Structure of a cache

Buffer Pool…

- **Performance problems**
  - getting a page from the OS to user space is a system call (process switch) and a copy
  - Cost to fetch 512 bytes exceeds 5,000 instructions.
  - Many DBMSs (System R and Ingres) constructed their own buffer pool to reduce overhead.
  - OS should cut this overhead to few hundred instructions.

Buffer Pool…

- **Replacement policy control**
  - LRU is a good tactic for buffer management in OSs.
  - Database access in INGRES is a combination of
    1. Sequential access to blocks which will not be referenced.
    2. Sequential access to blocks which will cyclically referenced.
    3. Random access to blocks which will not be referenced again;
    4. Random access to blocks for which there is a nonzero probability of reference.
  - Although LRU works well for the fourth one, it is a bad strategy for other situations.
  - Since a DBMS knows which blocks are in each category, a composite strategy may be good.
  - For case 4, it can use LRU, and for others it can use toss-immediate strategy.
  - Some means should be provided so that OS can accept an advice from application program concerning the replacement policy.
  - DBMS knows access pattern in advance - should dictate policy
  - This is a major OS/DBMS distinction!

Buffer Pool…

- **Pre-fetch:**
  - Although UNIX correctly pre-fetches pages when a sequential access is detected, on several instances it fails.
  - In INGRES, the examination of a block knows the next block to be accessed. Unfortunately it may not be the next one in logical file order.
- **Crash Recovery:**
  - DBMS should provide recovery from hard and soft crashes..
  - The unit of work may be big.
  - DBMS provide this service with intention lists.
  - The page for which commit flag is set must be forced to disk after all pages in the intentions list.
  - So the service required from OS buffer manger is a selected force out.
  - Requires page-level control of flushing.

Outline

- Examples are drawn from UNIX, but the points raised have a general applicability.
- Several popular OS services are examined.
  - Buffer pool management
  - **The file system**
  - Scheduling and IPC
  - Consistency control
- Conclusions

File System

- UNIX supports files of character arrays of dynamically varying size.
- On top of this abstraction, DBMS can provide different abstractions.
- Second approach is to provide record management by OS.
  - Structured files are provided.
- The second service provided by DBMS may not be efficient on top of character arrays.

File System…

- **Physical contiguity**
  - Character array block is expanded one block at a time. Next logical block is not physically close to the previous block.
  - Since DBMS does considerable sequential access, the result is considerable disk arm movement.
  - lack of clustering (translate: block granularity problem.)
- **Multiple trees: (dir, file, database). Unify?**
  - UNIX implements two services by means of data structures.
    - The blocks in a given file are kept track of in a tree, pointed by i-node block.
    - The files in a mounted file system have a user visible hierarchical structure.
    - DBMS adds a third tree (B-tree)
- One tree may be efficient than three trees!

Outline

- Examples are drawn from UNIX, but the points raised have a general applicability.
- Several popular OS services are examined.
  - Buffer pool management
  - The file system
  - **Scheduling and IPC**
  - Consistency control
- Conclusions

Scheduling, IPC

- Simplest way is one process per user.
  - System R and UNIX follow this method.
- Alternative: allocate a one run-time database process which acts as a server.
- However, the design of OS favors the first one.
- There are two problems with the process per user approach.
  - Performance and critical section

Scheduling, IPC…

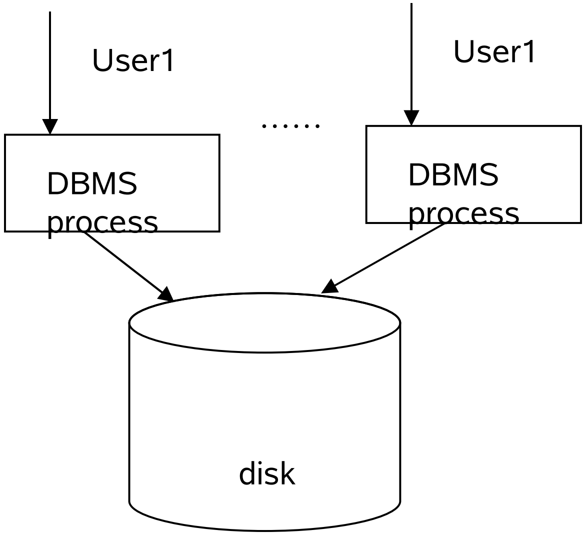
- Performance problem:
  - When the page is not found in the main memory, DBMS suspends the process and another process is run.
  - Task switch is expensive in many OSs as there is a lot of state information.
  - There is a high price to pay for a buffer pool miss.

Scheduling, IPC ..

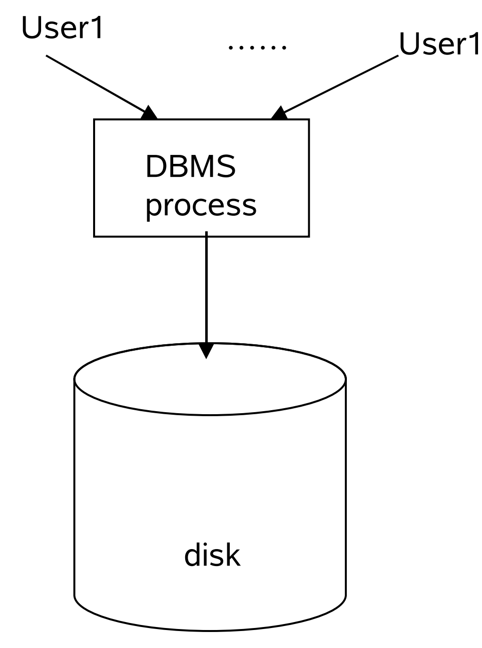
- **Critical sections (Convoys)**
  - DBMS processes have critical sections.
    - Buffer pool is a critical section.
  - System R sets short term locks.
  - Problem occurs when OS deschedules process which holding short-term lock.
  - This convoy effect has a devastating effect on the performance.

Scheduling, IPC…

- **Over process-per-user model other model may be good**
- **Server model**
  - It is viable if OS provides a facility to allow multiple processes to send a message to single process.
  - Such a server should do own scheduling and multi tasking.
  - Also, server can avoid multi-tasking by resorting to FCFS.
  - I/O process model
  - Expensive messages?
    - The cost of round-trip message is 5000 instructions.
    - Care must be exercised
  - Some systems do this now & get good performance.



Process per user structure



Server DBMS structure

Scheduling: Summary

- Both server model and process per user model seems unattractive.
- Problem is the overhead for task switches and messages.
- OS should provide fast path functions for DBMS consumers.
- Otherwise, DBMS users will implement their own multitasking, scheduling, and message systems.
  - Mini OS running in user space.
- Solution: OS should create a special scheduling class for DBMS.
  - Processes in this class would never forcibly de-scheduled but might voluntarily relinquish the CPU.
  - It will solve the convoy problem.
- Fast path through task switch/scheduler loop to pass control to one of their sibling processes.
  - Passing control with less overhead.

Outline

- Examples are drawn from UNIX, but the points raised have a general applicability.
- Several popular OS services are examined.
  - Buffer pool management
  - The file system
  - Scheduling
  - **Consistency control**
- Conclusions

Consistency/Locking

- OS locking granularity fixed (pages or pages)
- Few OSs support finer granularity of locks.
- Byte-level recover doesn't help with transactions
- Sometimes it is suggested that both CC and recovery should be provided by the OSs.
  - However, problem comes with the buffer management as locking, recovery & buffer management interact
  - e.g.: must write log pages in order, so must be able to flush individual pages. can't release locks until commit is logged.
  - if any of these is missing from OS, DBMS must implement all of them to avoid duplication.

Paged virtual memory

- Bind files in user's paged virtual memory address space.
- Such files can be referenced by the program.
- No need of doing explicit reads and writes;
  - He can depend on paging facilities of OS.
  - There are some problems with this approach.
- Large files:
  - Paging hardware creates 4 bytes per 4K page.
  - 100M-byte fiile will have an overhead of 100 K bytes.
  - So there will be a paging for a page table.
    - So each I/O operation causes two page faults: one for the page containing the page table and one on the data itself. address space big enough?

Conclusion

- **Bottom line**
  - OS services in many existing OSs are either too slow or inappropriate.
  - Current DBMS usually provide their own and make little or no use of those offered by OS.
  - A DBMS would prefer small efficient OS with only desired services.
  - Real-time operating systems may be ideal
  - General purpose OSs offer all things to all people at much higher overhead.