Library Express - Interlibrary Loan Request

9/7/00

ARPACI-DUSSEAU, REMZ

Requestor ID 37476385950

Email remzi@cs.wisc.edu

Status: Staff/Faculty

Citation

Stonebraker, Michael
Operating System Support for Database Management

Notice warning concerning copyright restrictions: The Copyright Law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material. Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be "used for any purpose other than private study, scholarship, or research." If a user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of "fair use," that user may be liable for copyright infringement. This institution reserves the right to refuse to accept a copying order if, in its judgement, fulfillment of the order would involve violation of copyright law.

Upon receipt of this electronic reproduction of the publication you have requested, we ask that you comply with copyright law by not systematically reproducing it, or in any way distributing or making available multiple copies of it.

ISSN/ISBN 0001-0782

OCLC #

Updated: 

Notified

2112
In this issue:

**ACM '81 Advance Program**

The Proposed New Computing Reviews Classification Scheme

1. A POL.A - POL.B, RETURNS QUOTIENT AND REMAINDER

\[ \text{DIO} + 1 + 0 x + /, \text{RNK} + (p p A + \text{STRIP} A) \Gamma p p B + \text{STRIP} \rightarrow (p p A) = p p B) / S0 \]

5. \[ h (((\text{RNK} - p p B) p 1), p B) p B \]

6. \[ S O: + (1 + / p B) / S I \]

8. \[ x p p R E S + (2, p Q) \uparrow (1, p Q) p Q + (A + /, B) \]

9. \[ S 1: R C H + 1 + (p B) + I + (x / p B) - (\phi, (|B| / / / |B|) O C T \downarrow) \uparrow \]

10. \[ R B + \uparrow \uparrow \uparrow, R \]

11. \[ Q \leftarrow (R N K p 1) p 0 \]

12. *CAN A QUOTIENT BE FOUND?*

13. \[ H A S H, S, 1 \rightarrow 0 \]

14. \[ + (O + / / I + O C T |(A S + R C H - 1 + A) - S C A L E)| / N O \]
communications of the acm

Editor-in-Chief
Robert L. Ashenhurst
The University of Chicago
1101 East 58th Street
Chicago, IL 60637
312 753-3673

Associate Editor-in-Chief
Lloyd D. Fox
Department of Computer Science
University of Colorado
Campus Box 430
 Boulder, CO 80309
303 492-7514

Department Editors
Artificial Intelligence and
Language Programming
Christine A. Montgomery
Operating Syg Systems, Inc.
Computer Architecture and Systems
John P. Hayes
University of Southern California
Applications: Engineering and the Sciences
F. H. Frank
Lawrence Livermore National Laboratory
Graphics and Image Processing
James D. Foley
George Washington University
Human Aspects of Computing
Henry F. Ledgard
International Business Machines Corporation
Management of Computing
M. H. Schwartz
DesignTech Corporation
Applications: Management Science/Operations Research
Harvey J. Greenberg
Energy Information Administration
Operating Systems
Arista K. Jones
Carnegie-Mellon University

Executive Editor
Janet G. Benton
ACM
1133 Avenue of the Americas
New York, NY 10036
212 262-6300

Computing Practices Editor
Patricia Breckinridge
ACM
1133 Avenue of the Americas
New York, NY 10036
212 262-6300

Applications: Operations and Management
Howard L. Mogul
The Wharton School
Programming Techniques and Data Structures
Elia Horowitz
University of Southern California
Simulation Modeling and Statistical Computing
Robert G. Sargent
Syracuse University
Social Impacts of Computing
Rob Kliger
University of California, Irvine
Systems Modeling and Performance Evaluation
Herbert D. Schwerin
Furdie University

Technical material intended for publication and communications on editorial matters should be sent to quadrant to the Associate Editor-in-Chief. Reports and Articles should be sent to the Editor-in-Chief. Computing Practices Editor. Addresses are given on this page. Papers on any subject relevant to the ACM's purposes will be considered: information for authors on format and style appears in the July 1981 issue.

News items and meeting and calendar information should be addressed to News Editor, Communications of the ACM, 1133 Avenue of the Americas, New York, NY 10036. Closing date is the 3rd of the month preceding the month of publication.

Letters of interest to Communications readers should be sent to the Editor-in-Chief (at the address on this page). Correspondence intended to be considered for ACM Forum should state that it was typed double-spaced, as Should Technical Correspondence.

Advertising (display and classified): The Williams & Watkins Co., 429 East Preston St., Baltimore, MD 21202; 301 269-4783.


Communications of the ACM (ISSN 0001-0782) is published monthly at Mt. Royal & Guildford Ave., Baltimore, MD 21202, by the Association for Computing Machinery, Inc. Second class postage paid at New York, NY 10001, and other mailing offices.

Subscriptions to Communications: ACM member subscriptions (in the annual dues of $40 (students $13): the nonmember annual subscription rate is $55. Single copies are (except July 1972, $7 to $10) members $3 and nonmembers $8. Microfilm editions through 1978 can be purchased from Microform Dept., Humanities Services Div., Waverly Press, 428 East Preston Street, Balti- more, MD 21202. Microfiche editions through 1978 can be purchased from Kraus Microform, Route 110, Millwood, NY 10546. Microfilm and microfiche are also available from University Microfilm International, 300 North Zeeb Road, Ann Arbor, MI 48106. See top of mailing label for subscription expiration date coded in four digits, the first two are year, last two, month of expiration.

In addition to Communications, ACM publishes the quarterly Journal of the Association for Computing Machinery, Computing Surveys, ACM Transactions on Mathematical Software, ACM Transactions on Database Systems, and ACM Transactions on Programming Languages and Systems. Collected Algorithms from ACM, the monthly Computing Reviews, and the annual ACM Guide to Computing Literature. For rates, see ACM Publications Catalog available upon request.

Ordering information, Publications Catalog, and back issues of ACM journals, proceedings, and special reports (these three prefaced by a check mark and published by the ACM Order Department, P.O. Box 64145, Baltimore, MD 21264).

Copyright © 1981 by the Association for Computing Machinery, Inc. Certain material without fee is permitted provided that the copies are not made or distributed for direct commercial advantage and credit to the source is given. Abstracting with credit is permitted. For other copying of articles that carry a code at the bottom of the first page, copies permitted provided that the per-copy fee indicated in the code is paid through the Copyright Clearance Center, P.O. Box 76, Schenectady, NY 12301. For permission to republish write to: Director of Publications, Association for Computing Machinery. To copy otherwise, or republish, requires a fee and/or specific permission.

Notices of address changes and requests for membership applications and subscription information should be addressed to Association Services, 1133 Avenue of the Americas, New York, NY 10036. Allow 6 to 8 weeks for change of name and address membership to become effective. Send old label with new address notification. To avoid interruption of service, notify your local post office to forward mail. In case of absence, send a fee, the Post Office will forward 2d and 3d class periodicals.

When payment is included with membership applications, renewal notices, or invoices, please include the address: Association for Computing Machinery, P.O. Box 12115, Church Street Station, New York, NY 10024.

Postmaster: Please send Form 3579 to Communications of the ACM, 1133 Avenue of the Americas, New York, NY 10036.

ACM European Secretariat: 40, Paulus-Polarstrasse, 1071 DB Amsterdam, The Netherlands; Phone: 020 76 3581; Telex: 844 18352.

References Guides to the organization of ACM publications and communications manuals are available from Headquarters: ACM Boards, Committees, Chairs, and Representatives to Other Organizations (October 1980)
ACM Chapters and Chairman (November 1980)
ACM Student Chapters and Chairman (January 1981)
ACM Special Interests Groups and Chairman (December 1980)

association for computing machinery
1133 Ave. of the Americas, New York, NY 10036

Council
Peter J. Denning, President
Michael A. Harrison, Vice-President
Kathleen A. Wagner, Secretary
Aaron Finerman, Treasurer
M. Stuart Lynn, Chairman, Publications Board
Paul Abrahams, Chairman, SIG Board
Daniel D. McCracken, Past President
Members-at-Large
Robert L. Ashenhurst (1978–82)
David Brandin (1980–1982)
Herbert R. Gorsch (1980–84)
David K. Hulse (1980–84)
Malcolm G. Lane (1978–82)
Raymond E. Miller (1978–82)
Regional Representatives
Thomas A. D. Austrian, Greater New York (1979–82)
Dennis J. Frailey, South Central (1981–84)
Peir W. Foster, Northeast (1981–84)
Myrna Littman, Allegheny (1979–82)
Larre Mischke, Mountain (1980–83)
Robert D. Parasol, European (1980–83)
Robert E. Sargent, North Central (1981–84)
Evelyn A. Swan, Southern California (1979–82)
Charles S. Williams, Southeast (1980–83)
David C. Wood, Capital (1980–83)
Marshall C. Yovets, East Central (1979–82)
Stephen N. Zilles, Pacific Region (1981–84)

Executive Director
Sydney Weinstein

Executive Secretary
Irene Holister
Director of Publications
Mark Mandelbaum

Executive Editors
Jane L. Benskin
Communications, Computing Surveys, Journal, Transactions
Arthur R. Blum
Computing Reviews, ACM Guide to Computing Literature

Editorial Staff at Headquarters
Jane C. Catterton, Administrative Assistant
John Stier, News Editor
Tina Papathes, Assistant Editor
Marlyn Salesiano, Copy Chief
Don Wertheiser, Production Editor
Catherine Yunque, Production Assistant
Carol Scharnberger, Editorial Secretary

Publications Board
M. Stuart Lynn, Chairman
Robert L. Ashenhurst
Michael Cooper
Adele Goldberg
John Grosse
Raymond E. Miller
Christina A. Montgomery
John R. Rice
Evelyn A. Swan

Editorial Committee
Robert L. Alpert, Chairman; Editor-in-Chief, Communications of the ACM
R. Daniel Bergeron, Editor-in-Chief, ACM Transactions on Graphics
Patrick C. Fischer, Editor-in-Chief, ACM Special Publications, Liaison for Proceedings Volumes
Michael Garey, Editor-in-Chief, Journal of the Association for Computing Machinery
Adele Goldberg, Editor-in-Chief, Computing Surveys
Susan L. Graham, Editor-in-Chief, ACM Transactions on Programming Languages and Systems
David K. Hulse, Editor-in-Chief, ACM Transactions on Database Systems
Richard Hanson, Editor ACM Transactions on Mathematical Software
Webb Miller, Editor-in-Chief, ACM Transactions on Mathematical Software
John R. Rice, Editor-in-Chief, ACM Transactions on Programming Languages
Jean E. Sambrot, Editor-in-Chief, Computing Reviews and ACM Guide to Computing Literature
Earl Schweppes, Coordinator
George E. Forsythe Student Paper Competition
Thomas A. Standish, Editor, ACM Monograph Series

212 262-6300
Operating System Support for Database Management

Michael Stonebraker
University of California, Berkeley

1. Introduction

Database management systems (DBMS) provide higher level user support than conventional operating systems. The DBMS designer must work in the context of the OS he/she is faced with. Different operating systems are designed for different use. In this paper we examine several popular operating system services and indicate whether they are appropriate for support of database management functions. Often we will see that the wrong service is provided or that severe performance problems exist. When possible, we offer some suggestions concerning improvements. In the next several sections we look at the services provided by buffer pool management; the file system; scheduling, process management, and interprocess communication; and consistency control. We then conclude with a discussion of the merits of including all files in a paged virtual memory.

The examples in this paper are drawn primarily from the UNIX operating system [17] and the INGRES relational database system [19, 20] which was designed for use with UNIX. Most of the points made for this environment have general applicability to other operating systems and data managers.

2. Buffer Pool Management

Many modern operating systems provide a main memory cache for the file system. Figure 1 illustrates this service. In brief, UNIX provides a buffer pool whose size is set when the operating system is compiled. Then, all file I/O is handled through this cache. A file read (e.g., read X in Figure 1) returns data directly from a block in the cache, if possible; otherwise, it causes a block to be "pushed" to disk and replaced by the desired block. In Figure 1 we show block Y being pushed to make room for block X. A file write simply moves data into the cache; at some later time the buffer manager writes the block to the disk. The UNIX buffer manager used the popular LRU [15] replacement strategy. Finally, when UNIX detects sequential access to a file, it prefetches blocks before they are requested.

Conceptually, this service is desirable because blocks for which there is so-called locality of reference [15, 18] will remain in the cache over repeated reads and writes. However, the problems enumerated in the following subsections arise in using this service for database management.
virtual memory (e.g., Pilot [16]) may provide a solution to this problem. This topic is examined in detail in Section 6.

2.2 LRU Replacement

Although the folklore indicates that LRU is a generally good tactic for buffer management, it appears to perform only marginally in a database environment. Database access in INGRES is a combination of:

1. sequential access to blocks which will not be rereferenced;
2. sequential access to blocks which will be cyclically rereferenced;
3. random access to blocks which will not be referenced again;
4. random access to blocks for which there is a nonzero probability of rereference.

Although LRU works well for case 4, it is a bad strategy for other situations. Since a DBMS knows which blocks are in each category, it can use a composite strategy. For case 4 it should use LRU while for 1 and 3 it should use toss immediately. For blocks in class 3 the reference pattern is 1, 2, 3, ..., n, 1, 2, 3, .... Clearly, LRU is the worst possible replacement algorithm for this situation. Unless all n pages can be kept in the cache, the strategy should be to toss immediately. Initial studies [9] suggest that the miss ratio can be cut 10-15% by a DBMS specific algorithm.

In order for an OS to provide buffer management, some means must be found to allow it to accept “advice” from an application program (e.g., a DBMS) concerning the replacement strategy. Designing a clean buffer management interface with this feature would be an interesting problem.

2.3 Prefetch

Although UNIX correctly prefetches pages when sequential access is detected, there are important instances in which it fails.

Except in rare cases INGRES at (or very shortly after) the beginning of its examination of a block knows exactly which block it will access next. Unfortunately, this block is not necessarily the next one in logical file order. Hence, there is no way for an OS to implement the correct prefetch strategy.

2.4 Crash Recovery

An important DBMS service is to provide recovery from hard and soft crashes. The desired effect is for a unit of work (a transaction) which may be quite large and span multiple files to be either completely done or look like it had never started.

The way many DBMSs provide this service is to maintain an intentions list. When the intentions list is complete, a commit flag is set. The last step of a transaction is to process the intentions list making the actual updates. The DBMS makes the last operation idempotent (i.e., it generates the same final outcome no matter how many times the intentions list is processed) by careful programming. The general procedure is described in [6, 13]. An alternate process is to do updates as they are found and maintain a log of before images so that backout is possible.

During recovery from a crash the commit flag is examined. If it is set, the DBMS recovery utility processes the intentions list to correctly install the changes made by updates in progress at the time of the crash. If the flag is not set, the utility removes the intentions list, thereby backing out the transaction. The impact of crash recovery on the buffer pool manager is the following.

The page on which the commit flag exists must be forced to disk after all pages in the intentions list. Moreover, the transaction is not reliably committed until the commit flag is forced out to the disk, and no response can be given to the person submitting the transaction until this time.

The service required from an OS buffer manager is a selected force out which would push the intentions list and the commit flag to disk in the proper order. Such a service is not present in any buffer manager known to us.
2.5 Summary

Although it is possible to provide an OS buffer manager with the required features, none currently exists, at least to our knowledge. Designing such a facility with prefetch advice, block management advice, and selected force out would be an interesting exercise. It would be of interest in the context of both a paged virtual memory and an ordinary file system.

The strategy used by most DBMSs (for example, System R [4] and IMS [8]) is to maintain a separate cache in user space. This buffer pool is managed by a DBMS specific algorithm to circumvent the problems mentioned in this section. The result is a "not quite right" service provided by the OS going unused and a comparable application specific service being provided by the DBMS. Throughout this paper we will see variations on this theme in several service delivery areas.

3. The File System

The file system provided by UNIX supports objects (files) which are character arrays of dynamically varying size. On top of this abstraction, a DBMS can provide whatever higher level objects it wishes.

This is one of two popular approaches to file systems; the second is to provide a record management system inside the OS (e.g., RMS-11 for DEC machines or Enscribe for Tandem machines). In this approach structured files are provided (with or without variable length records). Moreover, efficient access is often supported for fetching records corresponding to a user supplied value (or key) for a designated field or fields. Multilevel directories, hashing, and secondary indexes are often used to provide this service.

The point to be made in this section is that the second service, which is what a DBMS wants, is not always efficient when constructed on top of a character array object. The following subsections explain why.

3.1 Physical Contiguity

The character array object can usually be expanded one block at a time. Often the result is blocks of a given file scattered over a disk volume. Hence, the next logical block in a file is not necessarily physically close to the previous one. Since a DBMS does considerable sequential access, the result is considerable disk arm movement.

The desired service is for blocks to be stored physically contiguous and a whole collection to be read when sequential access is desired. This naturally leads a DBMS to prefer a so-called extent based file system (e.g., VSAM [11]) to one which scatters blocks. Of course, such files must grow an extent at a time rather than a block at a time.

3.2 Tree Structured File Systems

UNIX implements two services by means of data structures which are trees. The blocks in a given file are kept track of in a tree (of indirect blocks) pointed to by a file control block (i-node). Second, the files in a given mounted file system have a user visible hierarchical structure composed of directories, subdirectories, etc. This is implemented by a second tree. A DBMS such as INGRES then adds a third tree structure to support keyed access via a multilevel directory structure (e.g., ISAM [7], B-trees [1, 12], VSAM [11], etc.).

Clearly, one tree with all three kinds of information is more efficient than three separately managed trees. The extra overhead for three separate trees is probably substantial.

3.3 Summary

It is clear that a character array is not a useful object to a DBMS. Rather, it is the abstraction presumably desired by language processors, editors, etc. Instead of providing records management on top of character arrays, it is possible to do the converse; the only issue is one of efficiency. Moreover, editors can possibly use records management struc-
ble. The DBMS suspends while waiting for required data and another process is run. It is possible to make task switches very efficiently, and some operating systems can perform a task switch in a few hundred instructions. However, many operating systems have "large" processes, i.e., ones with a great deal of state information (e.g., accounting) and a sophisticated scheduler. This tends to cause task switches costing a thousand instructions or more. This is a high price to pay for a buffer pool miss.

4.2 Critical Sections
Blasgen [3] has pointed out that some DBMS processes have critical sections. If the buffer pool is a shared data segment, then portions of the buffer pool manager are necessarily critical sections. System R handles critical sections by setting and releasing short-term locks which basically simulate semaphores. A problem arises if the operating system scheduler deschedules a database process while it is holding such a lock. All other database processes cannot execute very long without accessing the buffer pool. Hence, they quickly queue up behind the locked resource. Although the probability of this occurring is low, the resulting convoy [3] has a devastating effect on performance.

As a result of these two problems with the process-per-user model, one might expect the server model to be especially attractive. The following subsection explores this point of view.

4.3 The Server Model
A server model becomes viable if the operating system provides a message facility which allows n processes to originate messages to a single destination process. However, such a server must do its own scheduling and multitasking. This involves a painful duplication of operating system facilities. In order to avoid such duplication, one must resort to the following tactics.

One can avoid multitasking and a scheduler by a first-come-first-served server with no internal parallelism. A work request would be read from the message system and executed to completion before the next one was started. This approach makes little sense if there is more than one physical disk. Each work request will tend to have one disk read outstanding at any instant. Hence, at most one disk will be active with a non-multitasking server. Even with a single disk, a long work request will be processed to completion while shorter requests must wait. The penalty on average response time may be considerable [18].

To achieve internal parallelism yet avoid multitasking, one could have user processes send work requests to one of perhaps several common servers as noted in Figure 3. However, such servers would have to share a lock table and are only slightly different from the shared code process-per-user model. Alternatively, one could have a collection of servers, each of which would send low-level requests to a group of disk processes which actually perform the I/O and handle locking as suggested in Figure 4. A disk process would process requests in first-in-first-out order. Although this organization appears potentially desirable, it still may have the response time penalty mentioned above. Moreover, it results in one message per I/O request. In reality one has traded a task switch per I/O for a message per I/O; the latter may turn out to be more expensive than the former. In the next subsection, we discuss message costs in more detail.

4.4 Performance of Message Systems
Although we have never been offered a good explanation of why messages are so expensive, the fact remains that in most operating systems the cost for a round-trip message is several thousand instructions. For example, in PDP-11/70 UNIX the number is about 5,000. As a result, care must be exercised in a DBMS to avoid overuse of a facility that is not cheap. Consequently, viable DBMS organizations will sometimes be rejected because of excessive message overhead.

4.5 Summary
There appears to be no way out of the scheduling dilemma; both the server model and the individual process model seem unattractive. The basic problem is at least, in part, the overhead in some operating systems of task switches and messages. Either operating system designers must make these facilities cheaper or provide special fast path functions for DBMS consumers. If this does not happen, DBMS designers will presumably continue the present prac-
tice: implementing their own multitasking, scheduling, and message systems entirely in user space. The result is a “mini” operating system running in user space in addition to a DBMS.

One ultimate solution to task-switch overhead might be for an operating system to create a special scheduling class for the DBMS and other “favored” users. Processes in this class would never be forcibly descheduled but might voluntarily relinquish the CPU at appropriate intervals. This would solve the convoy problem mentioned in Section 4.2. Moreover, such special processes might also be provided with a fast path through the task switch/scheduler loop to pass control to one of their sibling processes. Hence, a DBMS process could pass control to another DBMS process at low overhead.

5. Consistency Control

The services provided by an operating system in this area include the ability to lock objects for shared or exclusive access and support for crash recovery. Although most operating systems provide locking for files, there are fewer which support finer granularity locks, such as those on pages or records. Such smaller locks are deemed essential in some database environments.

Moreover, many operating systems provide some cleanup after crashes. If they do not offer support for database transactions as discussed in Section 2.4, then a DBMS must provide transaction crash recovery on top of whatever is supplied.

It has sometimes been suggested that both concurrency control and crash recovery for transactions be provided entirely inside the operating system (e.g., [13]). Conceptually, they should be at least as efficient as if provided in user space. The only problem with this approach is buffer management. If a DBMS provides buffer management in addition to whatever is supplied by the operating system, then transaction management by the operating system is impacted as discussed in the following subsections.

5.1 Commit Point

When a database transaction commits, a user space buffer manager must ensure that all appropriate blocks are flushed and a commit delivered to the operating system. Hence, the buffer manager cannot be immune from knowledge of transactions, and operating system functions are duplicated.

5.2 Ordering Dependencies

Consider the following employee data:

<table>
<thead>
<tr>
<th>Employee</th>
<th>Salary</th>
<th>Manager</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td>
<td>10,000</td>
<td>Brown</td>
</tr>
<tr>
<td>Jones</td>
<td>9,000</td>
<td>None</td>
</tr>
<tr>
<td>Brown</td>
<td>11,000</td>
<td>Jones</td>
</tr>
</tbody>
</table>

and the update which gives a 20% pay cut to all employees who earn more than their managers. Presumably, Brown will be the only employee to receive a decrease, although there are alternative semantic definitions.

Suppose the DBMS updates the data set as it finds “overpaid” employees, depending on the operating system to provide backout or recover-forward on crashes. If so,
Brown might be updated before Smith was examined, and as a result, Smith would also receive the pay cut. It is clearly undesirable to have the outcome of an update depend on the order of execution.

If the operating system maintains the buffer pool and an intentions list for crash recovery, it can avoid this problem [19]. However, if there is a buffer pool manager in user space, it must maintain its own intentions list in order to properly process this update. Again, operating system facilities are being duplicated.

5.3 Summary

It is certainly possible to have buffering, concurrency control, and crash recovery all provided by the operating system. In order for the system to be successful, however, the performance problems mentioned in Section 2 must be overcome. It is also reasonable to consider having all 3 services provided by the DBMS in user space. However, if buffering remains in user space and consistency control does not, then much code duplication appears inevitable. Presumably, this will cause performance problems in addition to increased human effort.

6. Paged Virtual Memory

It is often claimed that the appropriate operating system tactic for database management support is to bind files into a user's virtual address space. In Figure 5 we show the address space of a process containing code to be executed, data that the code uses, and the files F1 and F2. Such files can be referenced by a program as if they are program variables. Consequently, a user never needs to do explicit reads or writes; he can depend on the paging facilities of the OS to move his file blocks into and out of main memory. Here, we briefly discuss the problems inherent in this approach.

6.1 Large Files

Any virtual memory scheme must handle files which are large objects. Popular paging hardware creates an overhead of 4 bytes per 4,096-byte page. Consequently, a 100M-byte file will have an overhead of 100K bytes for the page table. Although main memory is decreasing in cost, it may not be reasonable to assume that a page table of this size is entirely resident in primary memory. Therefore, there is the possibility that an I/O operation will induce two page faults: one for the page containing the page table for the data in question and one on the data itself. To avoid the second fault, one must wire down a large page table in main memory.

Conventional file systems include the information contained in the page table in a file control block. Especially in extent-based file systems, a very compact representation of this information is possible. A run of 1,000 consecutive blocks can be represented as a starting block and a length field. However, a page table for this information would store each of the 1,000 addresses even though each differs by just one from its predecessor. Consequently, a file control block is usually made main memory resident at the time the file is opened. As a result, the second I/O need never be paid.

The alternative is to bind chunks of a file into one's address space. Not only does this provide a multiuser DBMS with a substantial bookkeeping problem concerning whether needed data is currently addressable, but it also may require a number of bind-unbind pairs in a transaction. Since the overhead of a bind is likely to be comparable to that of a file open, this may substantially slow down performance.

It is an open question whether or not novel paging organizations can assist in solving the problems mentioned in this section.

6.2 Buffering

All of the problems discussed in Section 2 concerning buffering (e.g., prefetch, non-LRU management, and selected force out) exist in a virtual memory context. How they can be cleanly handled in this context is another unanswered question.

7. Conclusions

The bottom line is that operating system services in many existing systems are either too slow or inappropriate. Current DBMSs usually provide their own and make little or no use of those offered by the operating system. It is important that future operating system designers become more sensitive to DBMS needs.

A DBMS would prefer a small efficient operating system with only desired services. Of those currently available, the so-called real-time operating systems which efficiently provide minimal facilities come closest to this ideal. On the other hand, most general-purpose operating systems offer all things to all people at much higher overhead. It is our hope that future operating systems will be able to provide both sets of services in one environment.

References

1. Bayer, R. Organization and maintenance of large ordered indices. Proc. ACM-SIGFIDET Workshop on Data Description and Access, Houston, Texas, Nov. 1970. This paper defines a particular form of a balanced n-ary tree, called a B-tree. Algorithms to maintain this structure on inserts and deletes are presented. The original paper on this popular file organization tactic.
3. Blasgen, M., et al. The convoy phenomenon. Operating Syst. Rev. 13, 2 (April 1979), 20-25. This article points out the problem with descheduling a process which has a short-term lock on an object which other processes require regularly. The impact on performance is noted and possible solutions proposed.


9. Kaplan, J. Buffer management policies in a database system. M.S. Th., Univ. of Calif., Berkeley, Calif., 1980. This thesis simulates various non-LRU buffer management policies on traced data obtained from the INGRES database system. It concludes that the miss rate can be cut 10-15% by a DBMS specific algorithm compared to LRU management.


13. Lampson, B., and Sturgis, H. Crash recovery in a distributed system. Xerox Res. Ctr., Palo Alto, Calif., 1976 (working paper). The first paper to present the now popular two-phase commit protocol. Also, an interesting model of computer system crashes is discussed and the notion of "safe" storage suggested.

14. Lauer, H., and Needham, R. On the duality of operating system structures. Operating Syst. Rev. 13, 2 (April 1979), 3-19. This article explores in detail the "process-per-user" approach to operating systems versus the "server model." It argues that they are inherently dual of each other and that either should be implementable as efficiently as the other. Very interesting reading.


17. Ritchie, D., and Thompson, K. The UNIX time-sharing system. Comm. ACM 17, 7 (July 1974), 365-375. The original paper describing UNIX, an operating system for PDP-11 computers. Novel points include accessing files, physical devices, and pipes in a uniform way and running the command-line interpreter as a user program. Strongly recommended reading.


20. Stonebraker, M. Retrospection on a database system. ACM Trans. Database Syst. 5, 2 (June 1980), 225-240. A self-critique of the INGRES system by one of its designers. The article discusses design flaws in the system and indicates the historical progression of the project.