FIXING PAGES IN A DATABASE BUFFER*

Wolfgang Effelsberg
Department of Management Information Systems
University of Arizona, Tucson

1. Introduction

Most of today's general purpose database management systems (DBMSs) are implemented on top of a standard multi-user operating system. In order to avoid a duplication of implementation efforts, they use many of the features of the operating system, such as file management, process scheduling, interprocess communication, etc. Since the operating system is not tailored to the needs of a DBMS, many practical problems and inefficiencies result from this approach. A comprehensive overview over these issues can be found in [St81].

This short note addresses some of the the particular problems of DBMS buffer management under a virtual memory operating system with paging. Previous research in this area concentrated on prefetching strategies ([Re76],[Sm78]) and the double-paging problem ([Tu76],[SB76],[BS77], [LWF77],[FLW78]). None of the published papers discusses the difference between a page access, as seen by the database buffer manager, and a page access, as seen by the paging mechanism of the operating system. This paper attempts to clarify this difference and its consequences.

2. The Fixing Mechanism

The buffer manager is a component within the DBMS; it has a well-defined interface to the other components (see Fig. 1).

The components that call the buffer manager refer to the database as a page-structured address space. Callers are aware of page boundaries and use the DBMS catalog, index structures (like B-trees), address translation tables etc. to find the page numbers of the pages they have to access. Therefore, a typical call to the buffer manager includes the page number of the page needed. In addition, the intention to update can be passed as a call parameter. The buffer manager locates the requested page within the buffer, loads it from the disk if it is not yet there, and fixes it in the

---

*This work was supported by a NATO research grant, obtained via the German Academic Exchange Service.

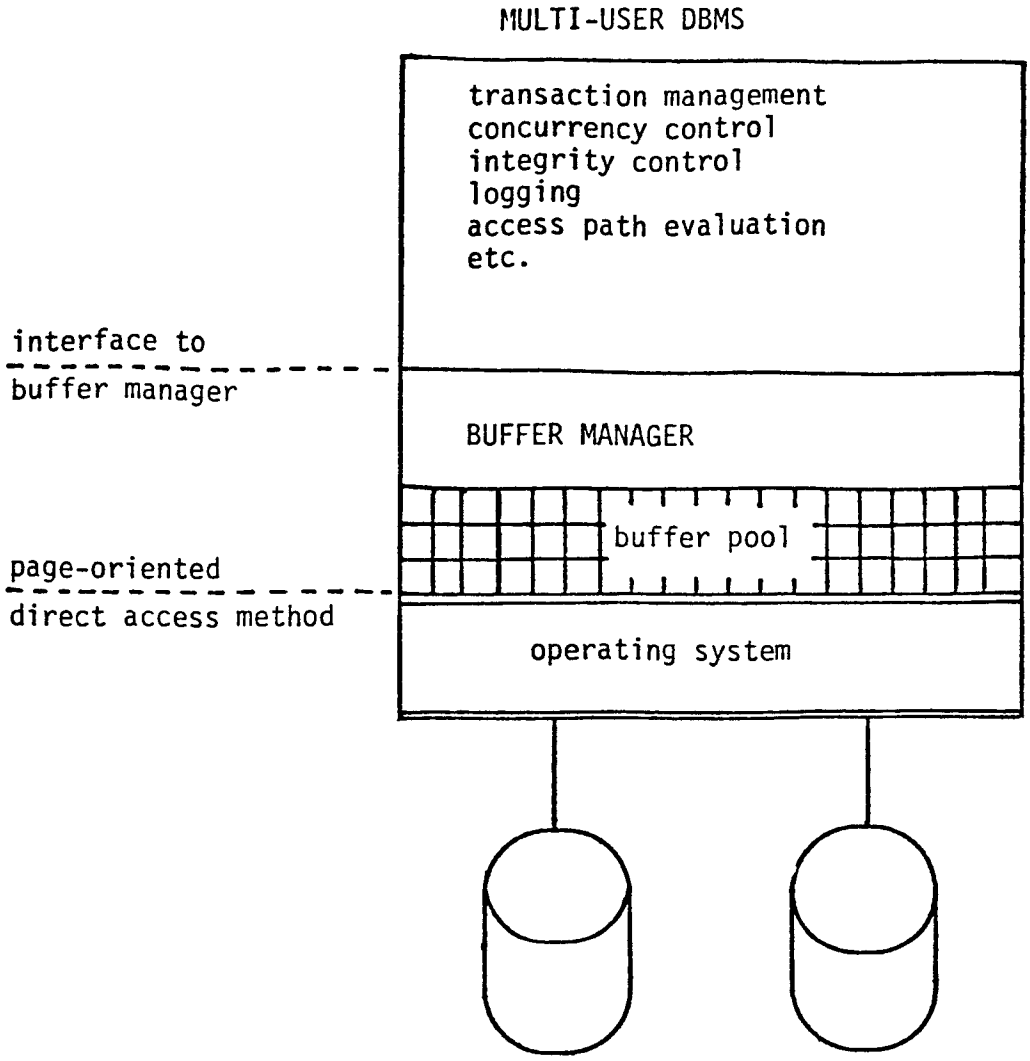


Fig. 1: Buffer management context

buffer in order to prevent replacement of the page during its use. It then passes the address of the page frame containing the requested page to the calling routine. Since the page has been fixed in the buffer. the calling routine can now execute machine instructions (like COMPARE. MOVE, etc.), addressing data objects within that page. The buffer manager guarantees addressability until the calling routine explicitly frees ("unfixes") the page. Since only the calling component knows when the phase of addressing within a certain page ends, it has to call the buffer manager again to perform an UNFIX-operation making the page available for replacement. This is usually done when no future accesses to the page are predictable. The fixing mechanism is illustrated in Fig. 2.

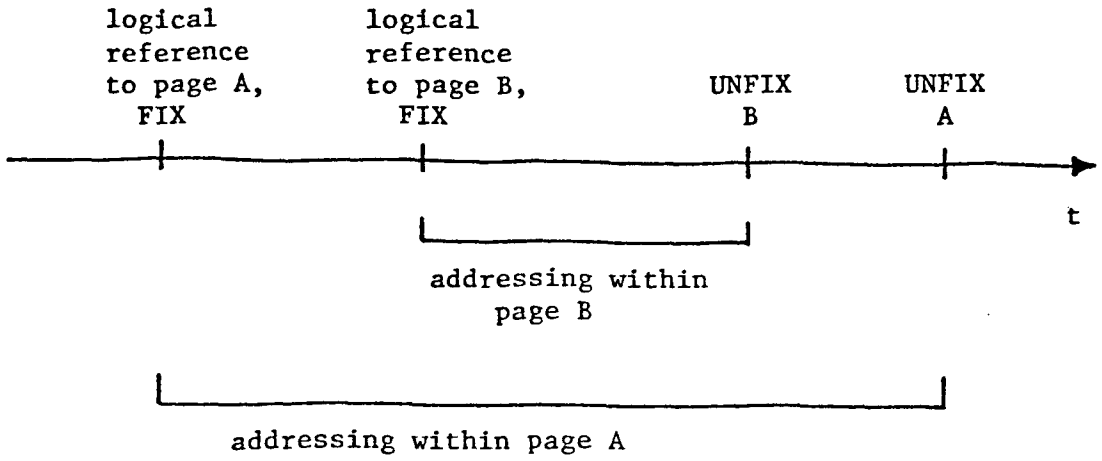


Fig. 2: Referencing, addressing and the fixing mechanism

Requests for pages at the interface of the buffer manager are called logical references; access to data objects within database pages by machine instructions is called addressing. Logical references are relevant for the replacement decisions of the buffer manager, whereas addressing is relevant for the paging routines of the virtual memory operating system.

In practice, many accesses to records of a database may consist of a logical reference to the page containing the record, followed by very few machine instructions addressing data within that page. For such operations (requiring only one page at a time in the buffer), the fixing mechanism can be implemented by implicitly fixing the youngest page of each transaction in the buffer. But more complex operations require the addressing of several buffer pages at the same time. An example is the split of a page of a B-tree (implementing an index or a set with set mode pointer-array), where tree entries have to be moved across page boundaries. For such operations, the fixing mechanism can be implemented as described above, and an explicit UNFIX-call is needed.

For each logical reference, the buffer manager has to perform the following actions:

- The buffer is searched, and the page is located.
- If the page is not in the buffer, a page replacement algorithm decides which of the buffer frames has to be replaced. Whenever the page selected for replacement has

been modified, it has to be written back to disk before the new page is read into the buffer frame. Each access to a database page on disk is called a physical reference. A physical reference is one of the most expensive operations within a DBMS; it not only costs 25-50 ms of disk access time, but also involves 2000-5000 CPU instructions in many operating system environments.

- The requested page is fixed in the buffer by marking its buffer control block.

- The fact that the page has been referenced is recorded since most replacement algorithms are based on the history of references to the buffer pages (e.g. LRU).

- Finally the address of the buffer frame containing the requested page is passed to the calling DBMS component as a return parameter.

The sequence of logical references to database pages (recorded as a sequence of page numbers) is called a (logical) page reference string. It describes the reference behavior of the DBMS, independent of the buffer size and replacement algorithm of the buffer manager. The logical reference behavior is determined by:

- the type of transactions constituting the DBMS load (retrieval/ update. direct/sequential access, etc.)

- the transaction mix (number and type of parallel transactions)

- the access path structures provided by the DBMS, in the form selected in the internal database schema. For example, direct access on the external schema level may result in quite different logical page reference strings depending on the existence of an index (e.g. B*-tree) for the attribute specified in a query. Without an index, all records of a certain type (or all tuples of a relation) would have to be scanned, leading to a much longer reference string.

Not every logical reference leads to a physical reference, but every physical reference is preceded by a logical reference. In contrast to logical references, physical references are strongly influenced by the size of the database buffer and the replacement algorithm of the buffer manager. Since physical references are expensive, the optimization of the replacement algorithm is very important for the overall performance of the DBMS. Optimization means the minimization of the number of physical disk accesses for a typical transaction load, described by a logical reference string. Some examples of studies using logical reference strings are [Ro76], [HS79], [LA80], [Lo81], [Ef81].

3. Consequences of Fixing

The most important consequence of the fixing mechanism is the fact that the page least recently addressed is not necessarily the same as the page least recently referenced (see example in Fig. 2). Usually, several database pages are fixed in the database buffer at the same time, and machine instructions addressing data objects within these pages can be executed without re-referencing any of the database pages.

If LRU stacks are used to illustrate this phenomenon, it is obvious that the first m pages in the LRU stack of the paging routines are not always the same as the first m pages in the LRU stack of the database buffer manager. It is only required that the first m pages (1<m<buffer size) in the stack of the operating system are a subset of the fixed pages in the database buffer. This is illustrated in Fig. 3. Models for double paging should not assume that the first pages in the two LRU stacks are identical (compare [LWF77], pages 343 and 344).

The observed phenomenon also has consequences for LRU replacement strategies in the buffer manager. Since the buffer manager has no knowledge about addressing within the buffer pages, replacement decisions can only be based on FIX and UNFIX events for a specific page. In general, the UNFIX event does not follow the FIX event immediately; rather, other pages are fixed and freed in between. In this context, two LRU strategies are possible:

- replace the page that was least recently referenced (fixed)

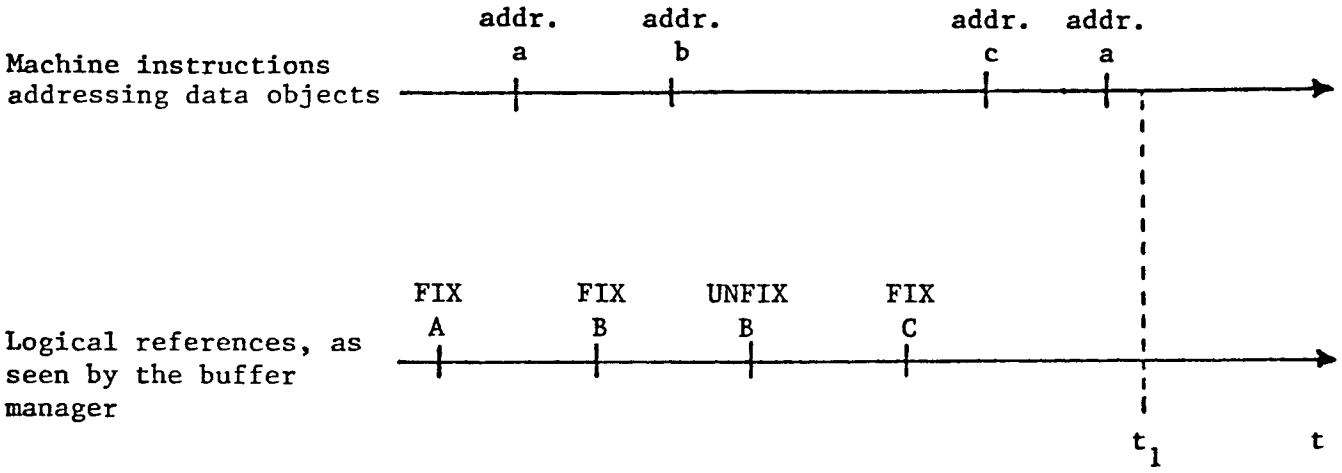
- replace the page that was least recently unfixed.

For example, in Fig. 2, page A would be the page least recently referenced, whereas page B would be the page least recently unfixed. In the traditional approach, a logical reference (the FIX event) would be interpreted as the "usage" of a database page. On the other hand, the UNFIX event is probably much closer to the last real usage (i.e., addressing) of the page. Therefore, it seems to be more appropriate to use the "Least Recently Unfixed" version of LRU for a database buffer manager.

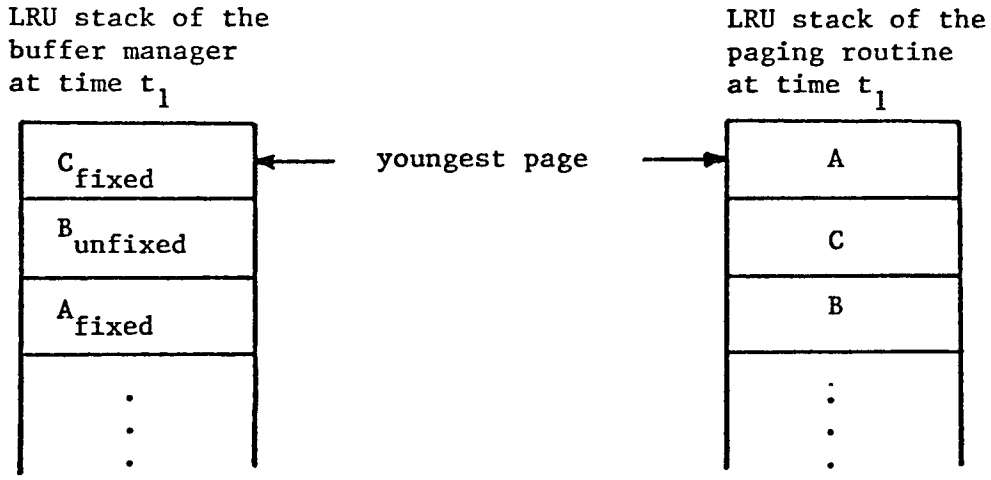
Another consequence of the fixing mechanism is its potential interference with replacement strategies. Only unfixed pages within the buffer are replacement candidates. Whereas a "Least Recently Unfixed" strategy ensures implicitly that only unfixed pages are considered for replacement, all other common replacement algorithms (e.g., "Least Recently Referenced", CLOCK, RANDOM) have to be modified so that they operate on unfixed pages only.

For simplicity, it is assumed that database pages and OS pages have the same size.

Data object a is in page A
" " b " " " B
" " c " " " C



(a) referencing and addressing behavior



(b) LRU stacks at time t_1

Fig. 3: LRU stacks of buffer manager and paging routine; an example

Since fixed pages can not be removed from the buffer, care should be taken to keep the fixing period for each page as short as possible. If too many buffer pages are fixed at the same time, thrashing occurs. This phenomenon was observed in a commercially available CODASYL DBMS ([Ef81]). In the worst case, if all buffer pages are fixed and a logical reference to a new page is made, one of the currently active transactions (or at least actions) has to be rolled back.

4. Conclusions

We have tried to explain the need for a fixing mechanism in the buffer manager of a DBMS executing under a virtual memory operating system. Fixing guarantees the addressability of several buffer pages at the same time.

The most important consequences of fixing are:

- Models for double paging become more complicated.
- For LRU and similar replacement strategies, there are two possible definitions for page "usage": the FIX event and the UNFIX event.
- Fixing can interfere with replacement algorithms; only unfixed pages can be replaced.
- The fixing period for each page should be as short as possible to prevent thrashing in the buffer.

To avoid misunderstandings, we would like to emphasize that the fixing mechanism is not related to the locking mechanism of the DBMS. In addition to fixing, some locking technique is always required to ensure database consistency in a multi-user environment.

Acknowledgment

The author wishes to thank Mary E. Loomis for her assistance during the preparation of this paper.

References

BS77 Brice,R.S. and Sherman,S.W.: An Extension of the Performance of a Database Manager in a Virtual Memory System using Partially Locked Virtual Buffers, ACM TODS, Vol.2, No.2 (1977), pp. 196-207.

Ef81 Effelsberg,W.: Systempufferverwaltung in Datenbanksystemen. Dissertation, Fachbereich Informatik, Technische Hochschule Darmstadt (1981).

FLW78 Fernandez,E.B., Lang,T. and Wood,C.: Effect of

Replacement Algorithms on a Paged Buffer Database System, IBM J. Res. Develop., Vol.22, No.2 (1978), pp. 185-196.

HS79 Hawthorn,P. and Stonebraker,M.: Performance Analysis of a Relational Database Management System, Proc. ACM SIGMOD Conf., Boston (1979), pp. 1-12.

LA80 Loomis,M.E.S. and Allen,F.W.: Paging Behavior and Performance Optimization in a CODASYL DBMS, Proc. International Conference on Databases, BCS/Heyden & Sons, London (1980), pp. 119-134.

Lo81 Loomis,M.E.S.: Logical, Internal and Physical Reference Behavior in CODASYL Database Systems, MIS Technical Report, U. of Arizona, 1981.

LWF77 Lang,T., Wood,C. and Fernandez,E.B.: Database Buffer Paging in Vitual Storage Systems, ACM TODS, Vol.2, No.4 (1977), pp. 339-351.

Re76 Reiter,A.: A Study of Buffer Management Policies for Data Management Systems, Technical Summary Report 1619, Mathematics Research Center, University of Wisconsin, Madison, March 1976.

Ro76 Rodriguez-Rosell,J.: Empirical Data Reference Behavior in Data Base Systems, Computer, Vol.9, No.11 (1976), pp. 9-13.

SB76 Sherman,S.W. and Brice,R.S.: Performance of a Database Manager in a Virtual Memory System, ACM TODS, Vol.1, No.4 (1976), pp. 317-343.

Sm78 Smith,A.J.: Sequentiality and Prefetching in Database Systems, ACM TODS, Vol.3, No.3 (1978), pp. 223-247.

St81 Stonebraker, M.: Operating System Support for Database Management Systems, Comm. ACM, Vol.24, No.7 (1981), pp. 412-418.

Tu76 Tuel,W.G.: An Analysis of Buffer Paging in Virtual Storage Systems, IBM J. Res. Develop., Vol.20, No.5 (1976), pp. 518-520.