KEV – A Kernel for Bubba

W. Kevin Wilkinont
Bell Communications Research
435 South St.
Morristown, NJ 07960

Haran Boral
MCC
3500 West Balcones Center Dr.
Austin, TX 78759

Abstract

Bubba is a parallel database machine under development at MCC. This paper describes KEV, Bubba’s Operating System kernel. After a brief overview of Bubba, a set of requirements for KEV is presented. Our requirements are contrasted with the requirements for conventional, uniprocessor database systems as outlined by Gray, Stonebraker and others. The KEV design is then presented and its features are compared to other distributed operating systems.

INTRODUCTION

Operating systems provide users with basic resource management services. These services range from memory management, task management, and file systems to synchronization and consistency. In his ground-breaking “Notes on Database Operating Systems”, Gray [1] relates the needs of data managers to some of these services by providing a detailed review of concurrency control and crash recovery protocols. He also makes a number of comments concerning time and space costs of certain services provided by general-purpose operating systems. Gray’s message is that whether one chooses to use an existing operating system or design and build a new one, care must be taken to ensure that certain low-level services are provided efficiently for a data management application.

Based on his Ingres [2] implementation experience, Stonebraker [3] catalogs problems in the services provided by Unix [4] in each of five major areas: buffer management, file system, task management and scheduling, inter-process communication, and consistency management. Stonebraker’s message is that, due to their generality, current operating systems do not provide the appropriate services. Real time OS’s which provide minimal facilities efficiently are much better suited for use by data management systems.

A substantiation of sorts for this argument is the Britton-Lee IDM-500 back-end database machine [5,6]. The IDM-500, without the database accelerator, runs on a Z8000; yet in benchmarks, it performed quite well when compared with Ingres running on a Vax 11/750 [7]. The IDM’s performance was limited not only by the relatively slow processor but also by the slow communications link between it and the host computer. Thus, its relatively “good” performance is, at least partially, due to the poor performance of Ingres which had to use a general-purpose OS (VMS in this case).††

Additional evidence to support this argument is provided by a study carried out by Hagmann and Ferrari [8] in which they compared alternative decompositions of Ingres across a front-end computer and a back-end computer. They conclude that a back-end should not run under a standard operating system. Instead a streamlined implementation of the necessary functionality should be provided.

The IDM-500 is evidence that special-purpose OS’s can substantially improve performance for limited application domains. Based on this and other work [9], we feel that special

†Work was done while author was visiting with the Database Program at MCC.

††The implicit and highly subjective assumption underlying our argument is that the code of Ingres and the IDM-500 is roughly of the same quality.
OS's can improve system performance 2–4 times over a general-purpose OS's. Our work is an attempt to extend this result to a highly parallel environment. We are building a high performance database machine, called Bubba. Bubba differs from other database systems in many aspects and thus places a different set of requirements on the OS. Although there exist many distributed operating systems, each of which has something to offer, we feel none is suitable because Bubba's environment is substantially different. Further it will evolve as a result of experimentation. For maximum flexibility, we chose to design our own operating system rather than use an existing one.

In this paper we describe the requirements for an OS kernel in our environment and the resulting design. Most of the ideas described are not new when examined individually. The integrated ideas represent something new. Preliminary data from the two implementations of our kernel, KEV, and its usage indicate that we did a good job but more data is required for a real evaluation. We expect to change KEV as a result of further evaluation so the reader should keep in mind that KEV's design is evolving, and this paper presents a recent snapshot of the design. The paper is organized as follows. We first review previous work on the relationship between database managers and operating systems. Next we provide an overview of those Bubba features relevant to this discussion. An overview of KEV, as defined and implemented at this time is then given. We conclude with status and plans.

OPERATING SYSTEMS FOR DATABASE MANAGEMENT SYSTEMS

In this section, we state a number of design principles for database operating systems. These are based on a review of previous work in the area. Our discussion is organized as a list of topics. This list was freely plagiarized from [3] which addressed five major areas: buffer management, file system, task management, IPC, and consistency.

Buffer Management

The main question here is how much of the buffer manager should be pushed into the operating system. The consensus seems to be very little because the OS does not have knowledge of the complex algorithms used by the DBMS. For example, Stonebraker [3] states that 4 access patterns are found in most DBMS's: random/sequential vs. repeated access/accessed once. Most OS's have a single page replacement strategy so they cannot provide good performance for each type of access (see Chou and DeWitt [10] for a study of this relationship). Traiger [11] points out that when the DBMS uses write-ahead logging for recovery, either the OS must do the logging or the OS must check with the DBMS before freeing buffers. This is because a modified data buffer cannot be released until its log records have been written and the OS has no knowledge of logging activity.

Typically, the DBMS is forced to implement its own buffer manager on top of the OS memory manager (usually a page manager). For performance, the DBMS buffer manager imposes some requirements on the OS service. In particular, it needs the ability to fix buffers in memory. Otherwise, unexpected page faults will result in convoys when the blocked task is holding a critical resource [12]. An explicit unfix operation is also required. The DBMS also requires the ability to share pages among processes, for example, the lock table and data dictionary information.

File System

The major issue here is the nature of the interface between the DBMS and the disk service provided by the OS. Most DBMS's prefer a block interface which corresponds to