1547

HO

Bell Laboratories Cover Sheet for Technical Memorandum

The information contained herein is for the use of employees of Bell Laboratories and is not for publication (see GEI 13.9-3)

Title: **Overview of the REDAS Data Base Manager in UNIX/TS Augmented.**

Date: **March 14, 1980**

TM: **80-3121-1**

Other Keywords: **UNIX**
Relational databases

Author(s)	Location	Extension
I. Brovman	**HO 3C511**	**5697**
C. E. Masters	**HO 1E307**	**5606**

Charging Case: **49622**
Filing Case: **49622**

ABSTRACT

REDAS is a relational database that is used in the NOC and EADAS OSS systems. It was used to support a database for the "ACS Surrogate Node" experiment in June, 1979. This involved installing REDAS under the UNIX/TS Augmented operating system. This required changes to that operating system including the addition of the Logical File System. This paper describes that effort.

Pages Text: 4 Other: 0 Total: 4

No. Figures: 0 No. Tables: 0 No. Refs.: 2

Bell Laboratories

subject: **Overview of the REDAS Data Base Manager in UNIX/TS Augmented.**
Case: **49622**
File: **49622**

date: **March 14, 1980**

from: **I. Brovman**
HO 3121
3C511 x5697

C. E. Masters
HO 3121
1E307 x5606

TM: **80-3121-1**

MEMORANDUM FOR FILE

1. Introduction

The purpose of this paper is to describe an experiment to select and install a database management system as part of the "ACS Surrogate Node" experiment in June, 1979 in department 3121. Because of limited development time (6 weeks) we searched for an existing database management program that we could modify to suit our needs. We also wanted a system that could be run on our UNIX*/TS Augmented system. Finally, the system had to be general enough to do the job and fast enough to provide reasonable response.

Because of previous experience we considered INGRES, a general purpose relational database system developed at the University of California (Berkeley). However, INGRES has a reputation for being slow, even with relatively small databases, and we verified this by some sample usage. We wanted a system that could handle the kind of a load that our ACS Surrogate database would have, and INGRES was obviously not suitable.

As an alternate, we chose REDAS [1] (RElational Data Access System for real-time applications), a system that is successfully used in the NOC and EADAS OSS systems. The REDAS version we obtained runs under CB-UNIX. CB-UNIX is a version of UNIX that has been modified to include features such as line disciplines, shared memory (MAUS - multiple access user space), semaphores, and a fast access flat file system with contiguously allocated files called the Logical File System (LFS) [2]. We were running a pre-release of UNIX/TS Augmented at the time, which did have MAUS and semaphores but not the LFS. The task we undertook was to modify our version of UNIX/TS Augmented to contain the LFS, then install the REDAS syntactic level routines, and finally construct a sample data base and write semantic level routines to populate and access our database.

Our intent here is to document the problems and solutions we discovered in setting up our sample database using REDAS. We are not attempting to provide an extensive or complete explanation of the logical file system or REDAS. The two referenced memos [2,1] do that. However, a brief description of some of the key features is in order. REDAS routines are compiled into a user program. The locking and protection of files is done through memory shared by all user programs using the database. The shared memory also contains basic relations (file) information and has some hooks for interprocess communication. When the shared memory

* UNIX is a trademark of Bell Laboratories.

(MAUS) has to be updated, semaphores are used to ensure exclusive use. The actual disk manipulations are done by the LFS.

2. Logical File System

The LFS is implemented as a pseudo character device and is installed like any other driver by modifying the configuration table and I/O driver library. However, the LFS uses several routines in the UNIX buffer management system which were unavailable or different in our UNIX.

One difference was the fact that CB-UNIX drivers, for instance the LFS, utilize a pool of buffer headers per disk device, instead of just one buffer header as in UNIX/TS. This leads to an improvement in performance, especially for raw disk I/O, so we decided to modify our UNIX. We added a routine to initialize the buffer pool, added routines to get and free buffer headers, and changed the existing RP06 disk driver to use the new routines.

A more serious problem was due to a change in the buffer manipulation policies of UNIX/TS. All buffers in older UNIX's (upon which CB-UNIX was based) were directly addressable but in UNIX/TS there are two pools of buffers, directly addressable and non-directly addressable. The LFS needed directly addressable buffers. In UNIX/TS, there was no way to request a directly addressable buffer and the buffer returned could be of either type. We had to design and implement a routine for returning a directly addressable buffer on request. We accomplished this by adding another level of routine with interfaces for asking for directly addressable and for the "don't care" case.

We also had to redesign the interface between REDAS and the LFS because it depended on the stty system call which was not available in our UNIX. Our new interface used the ioctl system call instead. One problem with this was that stty had returned a value to the user and ioctl did not. Our solution was to send the return value back to the user in the user structure that was originally passed to ioctl.

The changes described here affected 25 routines in the operating system (including the LFS) containing approximately 700 lines of code. About 50 of these lines of code were changed or added, and of the 25 routines, 5 were new. Also involved were 5 interface routines containing 30 lines of code with changes to 5 lines. It took about three weeks to get this done.

3. REDAS

REDAS is a real time data retrieval package which deals with two types of data representations: records, consisting of n-tuples of information about a particular entity, and binary relations, used to represent the data from records that must be retrieved rapidly. Data input and retrieval algorithms are written in a high level interpretive language which is implemented as postfix strings. Access to the database is done only through the REDAS interpreter which is one reason why the system is fairly portable to other applications. Shared memory (MAUS) and semaphores are used to provide protection from multiple writes and file deadlocks. Finally, REDAS depends on the LFS as its interface to the disk.

Once the LFS was installed, the next task was to install REDAS. Our first problem here was that, as delivered, the MAUS bit map was not being set up correctly. When REDAS requested more memory, the system gave away the MAUS virtual memory area, causing a memory fault the next time REDAS tried to access MAUS. The solution was to establish a bit map for the segmentation registers so that they would not be re-used incorrectly.

We still had a problem getting memory, however, because we used our one spare data segmentation register for MAUS. If we needed more memory, the allocate request would fail because there were no more data segmentation registers. We resolved this by compiling with separated i and d space, thereby increasing the number of available data segmentation registers. This problem will occur again if the programs using REDAS get large enough.

Several problems in installing REDAS were due to the fact that REDAS had originally been intended for use in specialized OSS systems and not as a general purpose database management system. The system was designed such that the interpreter is data independent and portable to other applications (syntactic level), while the algorithms for accessing the database are implemented as part of the applications program (semantic level). This was a fairly clean split except for the problem of deciding data access permissions. This part of the syntactic level was more tightly coupled to the application. Only certain application programs were given access permissions to relations in the REDAS database. Since our entire application structure was different (and not even defined at this point), we merely disabled this code.

Another problem along the same lines was the difficulty in understanding and learning to use the data retrieval language of REDAS. This language consists of postfix strings compiled into arrays where each string specifies a sequence of operations to be performed on the database. We found this awkward and confusing, and would prefer a more common type query language. We alleviated this problem somewhat by creating and using standard subroutines for linear file read, linear file write, indexed read, indexed write, etc. In this manner, all application programs could use calls to such functional subroutines and eliminate the need to use elaborate postfix strings.

A final problem was the installation of utilities to create and maintain the database. The LFS has to be checked for consistency in a manner similar to the way UNIX file systems are checked. In addition, a MAUS area in memory has to be created, saved whenever it changes, and restored at each boot. There were five utility programs that were essential and that had to be installed. These were for MAUS initialization, writing MAUS to disk, reading MAUS in from disk, checking the logical file system, and for MAUS creation. Several others were installed because they were useful (e.g., a utility to read MAUS). Unfortunately, there was no documentation on these, which made the task non-trivial.

The changes to install REDAS affected two operating system routines containing approximately 120 lines of code with 20 lines changed or added. The REDAS syntactic routines did not have to change but at the semantic level, 18 routines were affected involving 1650 lines. This took two weeks.

4. The Application

The database we finally constructed, once the system was installed, consisted of several linear and indexed files describing ACS type records; LNA, UBN, FS, FT, etc. The indexed files were basically n-tuples of fields related in some way. For example, the FS records had fields for an FS name, a flag, a capabilities list, and a time to and from which the FS could be used. The LNA records consisted of an LNA name, flag, port name, and a default UBN to be associated with this LNA.

In addition, the standard UNIX password file was implemented using a binary relation to aid in searches. We had an indexed file of password n-tuples and an inverted directory containing the hashed value of the login id and an index into that file. This inverted directory was a REDAS binary relation of the hashed login id and the file index. Since we hashed the login id's, we did have a collision problem to resolve. We did this by finding all the collided records and doing a linear search of those for the correct one.

Routines were written so that you could add, delete, retrieve, or change a record in any file based on the primary key for the file. There were many more sophisticated relationships that we would have liked to try but could not in our time frame. Also we were unable to run any load or timing tests. Work is continuing in this area, however, and some data will soon be available. Basically we had accomplished our purpose, to get the system on-line and try it.

There were thirty-six routines written to create our sample database consisting of 1800 lines of code. This part of the job took approximately one week. At this point, more files could be added to the database with relative ease.

5. Conclusions

In a relatively short time (a total of 6 weeks), we were able to learn, install and use the REDAS database system. This is particulary noteworthy in light of the fact that the version of UNIX we ran it under was new. Indeed, that is what caused most of our difficulties. This had a beneficial side effect in that we learned more about the internals of UNIX, REDAS and the Logical File System than we might have otherwise. More work needs to be done, however, in sample applications using REDAS before we can justify any criticism of the system. Hopefully this work will be undertaken in the future. For our needs of the "ACS Surrogate Node", REDAS accomplished what we intended it to do.

Some general observation on REDAS can be made. We found postfix strings difficult to use. One possible area of improvement would be to design a query language. This would make it easier to construct the application programs. However optimization would then be more difficult since by using the postfix strings you can control the way the search is done by how you construct the string. Some experiments in optimizing queries this way would prove interesting.

We also found the restriction that binary relations be of only integer fields rather limiting. It forced us to hash the character string we were interested in setting up a relation on. This was awkward, inefficient and caused extra problems (how to resolve collisions). It would be more general if the binary relation could be between a character string and an integer.

The most important consideration, however, is performance. Unfortunately we did not get a quantitative assessment of how REDAS would score in this area. It would be very interesting to run some bench marks, especially against other database systems that exist on UNIX. This area is certainly worth future consideration.

REFERENCES

[1] McSkimin, James R., REDAS - A Relational Data Access System for Real-Time Applications. TM 78-5221-1, October 9, 1978.

[2] McSkimin, James R., The Logical File System - A Fast Access File System Using UNIX Raw I/O. TM 79-9471-1, September 14, 1979.

I. Brovman

I. Brovman

C. E. Masters

HO-3121-IB/CEM-cem

C. E. Masters