DISC MEMORY CACHING ON A SERIES III WITH EXTRA DATA SEGMENTS

by KURT D. RAHM
SR. PROGRAMMER/ANAYLST
BOHEMIA, INC.

INTRODUCTION

A major bottleneck of any computer is disc I/O and the HP3000 is no exception. Hewlett-Packard has announced MPE V with disc memory caching which allows disc files to be resident in memory, therefore, providing increased throughput of disc I/O. Since disc memory caching will not be available on the Series III, users of these systems need some alternative to increase disc throughput. This paper explains how extra data segments, in conjunction with process handling and multiple rins, can be used to obtain the benefits of increased disc throughput without having disc memory caching.

DISC CACHING

Disc caching is a sub-system that utilizes excess main memory and processor capacity of certain HP3000's (Series 39, 42, 48 & 68) to eliminate a large portion of disc access delays. It locates, moves, and replaces disc domains in main memory for access by any user. A disc domain is all, or part, of a disc file.

When a process requests input from a disc file, a list of currently cached disc domains is searched. If the requested disc domain resides in main memory, a memory-to-memory transfer occurs between the cached disc domain and the requesting processes data area. The process will continue executing without an interrupt. If the data requested is not in main memory, a fetching strategy using numerous facts determines the optimal disc domain to bring into memory, then a memory-to- memory transfer occurs.

There are various options for disc caching when a write to disc is requested. The option selected is dependant on ; 1 - whether the required disc domain is currently cached, 2 - if there another write pending, and 3 - if disc caching enabled for writes. Assuming disc caching is enables for writes, a user should never have to wait for the physical write to disc.

With the addition of disc caching to a system, the delays to read or write disc data will be significantly reduced thus decreasing response time and increasing productivity. Disc caching, or an alternative technique, would therefore be a valuable asset to any system. You can get more information on disc caching from your SE, so I will not go into any more detail. This article will furhter discuss an alternative to disc caching with the use of extra data segments.

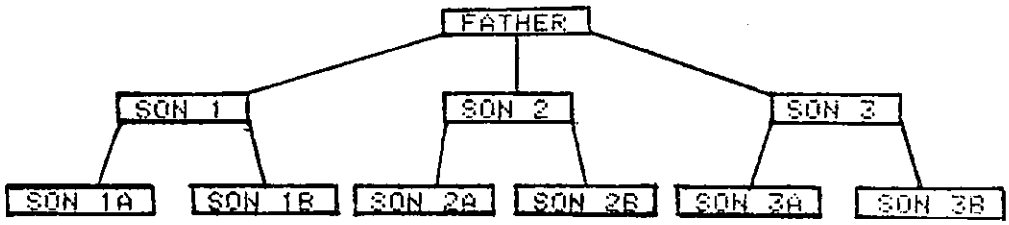
EXTRA DATA SEGMENTS.

An extra data segment, XDS, is a block of unstructured memory that is initialized to zero. The data in an XDS can be private, for the creator only, or shared by the creator and its sons (more later under process handling). Anything you use COBOL's workingstorage for, an XDS can also be used. Large arrays, table lookup, and parameter passing (linkage

section) are some examples. The question is then asked - If I can use working-storage or linkage section, why use extra data segments? Working-storage cannot be shared between processes, and what if your table is too large (i.e. MAXDATA can't be big enough). Extra data segments can be used to decrease an individual processes stack size, share data, decrease disc I/O, and increase throughput. Before we can look at how XDS's can be used to duplicate disc caching, we need to look at process handling (PH) and local rins.

PROCESS HANDLING

Process handling allows a user (process) to create, delete, activate, and suspend other processes. Process handling (PH) is required if you want to share an XDS. Refer to the family tree in Figure KDRTXT-1 for the following discussion. In order for a process to have access to an XDS, either it or its father or grandfather, etc, must have created the XDS. In order for 'SON 1A' to have access to an XDS, either 'SON 1A', 'SON 1' or 'FATHER' must have created the XDS. If 'SON 2' creates an XDS, 'SON 1A' doesn't have access to it, only 'SON 2', 'SON 2A' and 'SON 2B' would have access to this XDS. If you want more information about process handling, see chapter 7 in MPE Intrinsics manual.



LOCAL RINS

Local rins (resource identification numbers) are used to exclude simultaneous access of a resource by two or more processes within a family tree. Local rins can be locked, conditionally or unconditionally, and unlocked just like an IMAGE dataset, allowing exclusive resource access. If your application uses V/3000, you need some way to get the right screen to the right terminal at the right time. VOPENTERM defaults to $STDIN to paint the desired screen, but a parameter to VOPENTERM is 'termfile'. 'Termfile' can be file equated to some LDEV allowing the screen to be painted on that ldev and not $STDIN. By locking a local rin around the file equation and the VOPENTERM, some other process within the family tree can't change the file equation until the local rin is unlocked (assuming that all programs available to any process within the family tree locks the local rin prior to VOPENTERM). Local rins require MR capabilities and you now need to be careful to avoid deadlocking. Deadlocking occurs when two processes try to unconditionally lock a resource (local rin, data set, etc) that the other process has locked. Since MR allows multiple locks to be in affect, deadlock is a potential hazard. The only way to get out of a deadlock situation is to shutdown the system. If you want more information on local rins, see chapter 6 in the MPE Intrinsics manual. I want you to realize that only DS capabilities are required if you use extra data segments. PH is required if you intend to share extra data segments, and MR is required if local rins are required.

DUPLICATING DISC CACHING

Since data is either input from or output to disc I will handle these individually in presenting alternatives to disc caching. Since the majority of disc accesses are for input, let's discuss it first. Bohemia, Inc, has an Order/Entry system that uses 21 data sets to

edit orders and invoices for their sawmills, plywood and particleboard plants. These sets include information such as state, country, specie, grade, product, thickness, etc. Whenever an order/entry clerk enters an order, these sets are accessed to validate the entered data. Our procedure was to place these 21 sets into 3 XDS based upon record size and number of entries in each set. Each record consist of a dataset number, key, and miscellaneous data including a description. A batch job controls this whole process.

Referring to Figure KDRTXT-1, assume 'FATHER' is a program in the above batch job and therefore the key to the whole process. 'FATHER' creates the 3 XDS, sequentially reads the 21 data sets, loads the data into the approriate XDS and obtain a local rin so the right screen gets to the right terminal. Since the 'FATHER' has created the XDS and obtained the local rin, all processes it creates, or are created by his sons, have access to all 3 XDS and the local rin. The 'FATHER' then creates 5 sons (menus) on 5 different terminals, locking the local rin around the VOPENTERM. These menus later create a son based upon the option selected by the users, which will validate the input using the XDS's. If the value entered by the user is not in the XDS, the program goes to disc to attempt validation since the users have the capability to add items to the 21 data sets. In the Order/Entry system, the majority of disc input is a memory-to-memory transfer and not a physical disc access increasing throughput. For those who want more detail on how to create and access extra data segments, read about 'GETDSEG', 'DMOVIN' and 'DMOVOUT' intrinsics. For local rins, see 'GETLOCRIN', 'LOCKLOCRIN' and 'UNLOCKLOCRIN'.

This above mentioned process has been used at Bohemia since Jaunary, 1983, with great success. Our interactive users have commented on the noticable improvement in response time, and we have modified our printing of orders and invoices to access the XDS's for descriptions instead of going to disc.

What about output to disc? One goal of disc caching was to decrease the delays of physical disc activity, thus allowing the user to proceed sooner. We have showed how these delays can be decreased for disc input, but what about disc output. This can be duplicated by adding another XDS to the 'FATHER' and another process. Instead of your programs doing 'DBPUT', 'DBUPDATE', or, 'DBDELETE', they output to the new XDS and let an output process do the physical disc activity. In order to do this, some information needs to be available to all processes involved in the family tree. The first record of this output XDS can contain the following: 1 - disc output displacement, 2 - user output to XDS displacement, 3 - length of XDS, and 4 - status flag. Displacement is the length (in words) from the beginning of the XDS to the beginning of the data to be transferred. Disc output displacement is the number of words to the beginning of the next XDS record to be put (updated or deleted) to disc. User output displacement is the number of words to the beginning of the next available XDS record for interactive users (i.e. where the next user is going to put his/hers next disc output detail). Since we will By using the XDS as a circular file, the length of the XDS must be obtained to determine the "end-of-file". When the end is reached process starts back at the beginning. The process flag can be used to indicate various status' of the output XDS such as XDS full. Records two through the end of the XDS contain the following: 1 - database name or number, 2 - dataset name or number, 3 - activity flag (0-free space, 1-add, 2-update, 3- delete), and 4 - data to be put, updated, or deleted. After the users have put the disc activity to the XDS, the output process can read the same data, and perform the desired activity. Since the interactive user is writing to memory and not disc, the delays associated with a physical write are not encountered and the user can continue processing sooner. A local rin can be used to control whether a user or the update process has exclusive access to the output XDS.

POTENTIAL GOTCHA'S

Thinking through this process, realize there are a few potential problems in the above duplication of disc caching. The most critical involves the output phase. What happens if you get a system failure? The user thinks they have completed the transaction, but have they? If there is a system failure, there may be some data put by a user to the output XDS, but because of the volume or other delays, this data hasn't made it to the disc. If this is true, even if you have transaction logging enabled, no disc output has happened so your logging won't help and some data may be lost.

Security may also be a problem. Since the users at their terminals never get a colon prompt and never enter :HELLO, MPE's logon security won't work. Solution: create your own security that is used instead of MPE logon security.

One rule with process handling is - when the father stops, so does every other process in the tree. If something would happen to cause

'FATHER' to abort, all other process stop and the XDS's are lost. The only way this will happen with our 'FATHER' is with a "bounds violation" or "system failure".

The 'FATHER' is a batch job, and anything displayed to $STDLIST won't be seen until the job terminates. If a son aborts, we want to know the reason now, not hours later. SOLUTION: Redefine $STDLIST to a disc file and FCOPY the disc file to LP if you want to see it.

If you use local rins (MR capabilities), be aware of deadlock possibilities. SOLUTION: thoroughly test programs without local rins and MR capabilities (at more than 1 terminal at a time). Only after the program is tested should MR and local rins be added.

Kurt Rahm is the Senior Programmer/Analyst for Bohemia, Inc, in Eugene, Oregon. His work experience on the HP3000 family dates back to 1977, beginning on a Series II, to the Series III he now works on. Kurt has modified the Order/Entry system at Bohemia to access extra data segments, not disc files, to validate orders and invoices processed by their sales department. Kurt is an active member of the Oregon Regional Users Group (ORERUG) as a speaker and is a nominee for the presidents position of ORERUG.
