MPE XL Mapped Files

Bryan Carroll
Computer Systems Division
Hewlett Packard
Building 44-MV
19111 Pruneridge Avenue
Cupertino, Ca. 95014

Introduction

The MPE XL operating system introduces many new, exciting and powerful features to make programs more efficient and perform more effectively. Perhaps the single most powerful and exciting new feature of MPE XL is User Mapped File Access. User Mapped File Access can increase the efficiency of an application causing it to run many times faster than it would without User Mapped File Access. This paper will explore how mapped files are implemented, the performance gains possible, and some uses of User Mapped Files to replace MPE V features and increase application performance.

User Mapped Files are possible because of the HP Precision Architecture's expanded 64 bit address space. A Mapped File is a disc file that is mapped directly into the virtual address space. User Mapped Files are managed with pointers that are returned from a new system intrinsic, HPFOPEN, which has similar functionality to the existing MPE V FOPEN. The use of pointers allows a programmer to view a Mapped File as if it were a large array. Once the Mapped File is open, the file system is not needed to access the file until you are ready to close the file. As a program moves through a file it will eventually access a part of the file which is not in memory. The memory manager will be invoked to bring the missing part of the file into memory so it can be accessed by the program.

Impressive performance gains have been observed with Mapped Files. With the use of pointers, all file system overhead can be bypassed which increases performance. User Mapped File Access essentially replaces MPE Disc Caching functionality in many cases without the overhead of the file system or Disc Caching software which further increases performance.

Mapped Files

User Mapped File Access is available directly to MPE XL users without special capabilities via the HPFOPEN intrinsic (new with MPE XL). User Mapped File Access is also available to the various modules in the operating system and, in a way, all files on the system are accessed as Mapped Files. The difference between User Mapped Files and Mapped Files accessed through the file system is who controls the pointer to the file. If a file is accessed with the file system intrinsics (FREAD, FWRITE, etc), the file system will maintain the pointer to the file. If a file is opened with HPFOPEN specifying the appropriate options, a pointer is returned to the caller who can access the file by referencing the pointer. A file can be opened with User Mapped Access and accessed with a pointer as well as accessed with the file system intrinsics. In this case, there are two separate pointers for the file, one maintained by the file system and the other maintained by the user program.

MPE XL File System

The major advantage in using User Mapped File Access is the performance benefit that can be realized in bypassing the file system. Let's take a look at some of the tasks performed by the MPE XL File System so we can better understand the size of the performance gains possible with User Mapped File Access.

File System Example - FREAD

Let's examine a commonly used intrinsic like FREAD. The path through FREAD can be broken down into three major sections; the FREAD Intrinsic itself, the Type Manager, and the Storage Manager.

The **FREAD** intrinsic is responsible for the usual checking of parameters that takes place for all intrinsics. FREAD must also obtain the Process Local File Descriptor (PLFD) entry for the given file. The PLFD is like the MPE V Active File Table (AFT) which keeps track of all open files. The FREAD intrinsic must also verify your access rights to the file each time the intrinsic is called.

The **Type Manager** is a new concept within MPE XL. There is a Type Manager for each specific type of file such as Fixed Record Length disc files, or Variable Record Length disc

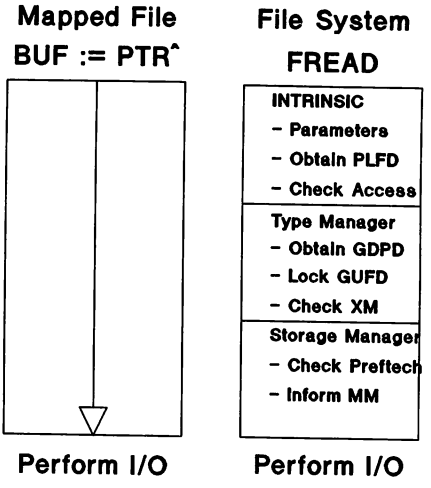
files. The Type Manager is responsible for obtaining the Global Data Pointer Descriptor (GDPD), which is very similar to the virtual address used for User Mapped Access which we will discuss later. The Type Manager is also responsible for locking the Global Unique File Descriptor (GUFD), which is like a combination of File Control Blocks (FCB) used on MPE V. The final task of the Type Manager is to check the connection between this file and another new MPE XL concept, the transaction manager. The transaction manager is an operating system subsystem which provides data integrity to many disc resident structures including selected user files.

The third section of the FREAD path involves another new concept, the **Storage Manager**. There is a unique Storage Manager for each type of physical storage device such as discs, tapes and printers. In the FREAD path, the storage manager is responsible for determining if a prefetch must be performed. The Storage Manager is also responsible for informing the memory manager when an I/O must be performed. The memory manager is responsible for all I/O on MPE XL systems.

The prefetch defined by the Storage Manager comes in two varieties: hard prefetches in which a process must block, and soft prefetches which do not require a process to block. The Storage Manager will prefetch from two pages (8K bytes) to a maximum of 64 pages (256K bytes) depending on the method of access (random or sequential). The Storage Manager optimizes the prefetch very well. If four consecutive reads take place to consecutive addresses in the file, the Storage Manager will recognize this as sequential I/O and perform a prefetch of 32 pages EVEN IF YOU ARE USING FREADDIR!

User Mapped Files

User Mapped File Access bypasses all of the above file system code and allows the user to directly access file pages. A file page is just a main memory copy of a 4K byte portion of a Mapped File. User Mapped File Access works by referencing the file as an array. A reference to a part of the file that is not already in memory will result in a page fault and the memory manager will bring the needed file page into memory, just as when a fault on a code page causes the memory manager to bring in the needed code page.



Mapped Files vs. File System

The performance gains possible with User Mapped Files when compared with using the MPE XL File System vary depending on the operation. In general, the CPU time required to perform a set of file operations will always be less when using User Mapped Files. The elapsed times however, will vary depending on the access (random or sequential). In general, the elapsed times for random access using User Mapped Access will be less than when using the file system. The elapsed times for sequential access of large files is often longer when using User Mapped File Access because of the prefetching the MPE XL File System can perform. Consider the following two examples:

Example 1 - Random Reads

Environment: Native Mode Pascal/XL program, 2 HP7933 Discs, 25000 record (100 MB) file, blocking factor = 1.

Test: 500 Random Reads

Results:

	CPU	Elapsed
User Mapped File:	913 ms	9958 ms
MPE XL File System:	1629 ms	28459 ms
Performance Gain:	78%	186%

Example 2 - Sequential Reads

Environment: Native Mode Pascal/XL program, 2 HP7933 Discs, 25000 record (100 MB) file, blocking factor = 1.

Test: 25000 Sequential Reads

Results:

	CPU	Elapsed
User Mapped File:	41017 ms	917393 ms
MPE XL File System:	73277 ms	307447 ms
Performance Gain:	79%	-66%

MPE Disc Caching

User Mapped File Access is a new and improved implementation of MPE V Disc Caching for some situations. The overhead involved with searching cache domains and mapped entries is eliminated when compared with User Mapped File Access.

The concepts of User Mapped File Access are very similar to MPE V Disc Caching except for the reduced CPU overhead of Mapped Files, and the prefetch or domain size is fixed at 4K bytes. When performing random access, the reduced CPU overhead allows User Mapped Files to perform better than all other access methods as illustrated in Example 1 above. Despite the larger prefetches available through the file system, User Mapped File Access can sometimes outperform the file system even when performing sequential operations because of the greatly reduced CPU requirement of User Mapped File Access.

Languages

User Mapped File Access requires the use of a "pointer type" variable and therefore can only be used with languages that will support a pointer type variable. User Mapped File Access is also not available in compatibility mode. The only languages that currently meet these requirements are HP Pascal/XL and HP C/XL. A possible third alternative is the SPL language when used with the native mode SPLash compiler since SPL supports a pointer variable type. Please contact the SPLash vendor, Software Research Northwest, for more information about the use of SPLash with User Mapped File Access.

It is possible to use User Mapped File Access from languages that do not support a pointer type variable by using procedures, functions or subroutines. A routine could be written in a language that does support the pointer type variable like Pascal/XL which could then be called from the main application which could be written in a language without pointer types like COBOL/XL or FORTRAN/XL. This option should be carefully considered since it adds complexity to the design of the application and therefore reduces supportability. This added level of complexity may not be worth the potential increased performance of User Mapped File Access.

HPFOPEN

The new MPE XL File System Intrinsic, HPFOPEN, provides access to the file pointers needed by programs wanting to use User Mapped File Access. Although the FOPEN intrinsic is still available, the new HPFOPEN intrinsic is a native mode superset of FOPEN and should be used in all native mode applications.

The new HPFOPEN intrinsic uses the following format.

```
HPFOPEN(FILENUM, STATUS, itemnum, item,
                         itemnum, item,
                            .       .
                            .       .
                            .       .
                         itemnum, item);
```

The itemnum/item pairs replace the positional FOPEN parameters and are used for specifying all file open criteria to HPFOPEN. A complete list of the item numbers, which currently range from 0 to 59, is available in the MPE XL Intrinsics Reference Manual (Part Number 32650-90028). There are item numbers for

all FOPEN parameters like record size, file code and blocking factor, as well as many new parameters like the 'will access' parameter to specify your anticipated access (Random or Sequential), and the 'long mapped' option for use with User Mapped Files.

User Mapped Access HPFOPEN Parameters

There are only two HPFOPEN parameters that are specific to User Mapped File Access, Long Mapped and Short Mapped. Since all files are really mapped files, the only thing we must do to receive mapped access to the file is to obtain the pointer to the file. This ability is provided in two different forms; a 64 bit long pointer (Long Mapped) and a 32 bit short pointer (Short Mapped). Specifying either of these two parameters to the HPFOPEN intrinsic will return the appropriate length pointer to your program. Opening a file with either of these mapped parameters does not prevent you from using any of the file system intrinsics like FREAD and FWRITE. File system intrinsics and User Mapped File Access can be used together by the same application.

Once the file is opened by HPFOPEN using either the Long or Short Mapped options, data can be read from or written to the file by indexing off the pointer. The pointer returned by HPFOPEN points to the first byte in the first record of the file. Unpacking of records in a variable length or undefined length record file must be performed by the user program. If the file has user labels, these can be accessed by negatively indexing from the pointer returned by HPFOPEN.

The following example contains portions of a Pascal/XL program which randomly reads and writes to a file using User Mapped File Access:

```
const page_len           = 4096;
type  page_type          = packed array [1..page_len] of char;
      file_pointer_type = ^ $extnaddr$ page_type;
var   base_file_ptr  : file_pointer_type;
      file_ptr       : file_pointer_type;
      rec_num        : integer;
      num_recs       : integer;
      filename       : packed array [1..38] of char;
      perm_file      : integer;
      update_access  : integer;
      buf            : page_type;
```

```
begin
        .
        .
        .

HPFOPEN(file_num, file_status,
        2, filename,
        3, perm_file,
       11, update_access,
       21, base_file_ptr);  { Long Mapped Pointer }

        .
        .
        .



file_ptr := addtopointer(base_file_ptr, rec_num * page_len);
if read_only then
   buf := file_ptr^;  { Read record 'rec_num' into buf }
   else
   file_ptr^ := buf;  { Write buf into record 'rec_num' }

        .
        .
        .
```

In the above example, a file is opened with the HPFOPEN intrinsic and depending on the 'read_only' flag, data is written to or read from the file using the Long Mapped pointer. Part of the power of User Mapped File Access can be seen in this example. The reading and writing of the file is performed by the memory manager when the pointer is simply referenced in an assignment statement. Notice in the example that the pointer returned by HPFOPEN which points to the first byte in the file is retained and a second pointer is used to move through the file.

Programming Considerations and Limitations

Most new features and enhancements have tradeoffs and it is no different with User Mapped File Access. The benefits of bypassing the file system with User Mapped Files also has the cost that some file system operations must still be performed and are left to the user. Designers and Programmers must be aware of several considerations and limitations involving the

use of User Mapped Files. We have already discussed the Language considerations and will now introduce other considerations and limitations one at a time.

End of File Pointer

One of the benefits of using User Mapped Files is that the file system overhead can be eliminated. Some functions that the file system would normally perform, like maintaining the end of file (EOF) pointer, must still be performed. Since the file system is not normally called when using User Mapped Files, the user must maintain the end of file pointer. Anytime data is added to a file beyond the current end of file pointer, this data will be lost unless the program also moves the end of file pointer. The end of file pointer can be maintained with the FPOINT and FCONTROL intrinsics as in the following example.

```
const page_len          = 4096;
type  page_type         = packed array [1..page_len] of char;
      file_pointer_type = ^ $extnaddr$ page_type;
var   base_file_ptr  : file_pointer_type;
      file_ptr       : file_pointer_type;
      new_eof        : integer;
      current_eof    : integer;
      num_recs       : integer;
      filename       : packed array [1..38] of char;
      perm_file      : integer;
      update_access  : integer;

begin
        .
        .
        .

hpfopen(file_num,file_status,
        2, filename,
        3, perm_file,
       11, update_access,
       21, base_file_ptr);
        .
        .
        .

{ Get the current EOF }
flabelinfo(filename, 0, error, itemnums, items, itemerrors);
        .
```

```
        .
        .

new_eof := current_eof + num_recs;  { Compute new EOF }
fpoint(file_num, new_eof);          { Move the File Pointer }
if ccode <> cce then error;

fcontrol(file_num, 6, dummy);  { Post the new EOF }
if ccode <> cce then error;
```

In the above example, a User Mapped File is opened and 'num_recs' records were added to the end of the file. In order to preserve these records, the end of file pointer had to be moved out 'num_recs' records which is performed by moving the current record pointer with FPOINT and posting the end of file pointer at that point with the FCONTROL intrinsic. If the end of file pointer had not been moved by the program which added records to the User Mapped File, the new records would have been lost since they were added to the file beyond the file limit.

User Mapped Files and File Types

Opening a file with User Mapped File Access allows the user to manipulate the data within the file, including any file structure information like end-of-record markers in variable length record files without restriction. In order to preserve the integrity of file types that incorporate file structure information in the file along with the data, User Mapped File Access will not be granted to certain file types. File types restricted from User Mapped File Access include Relative I/O files (RIO), Circular files (CIR), Message files (MSG) and any non-disc files. Files with variable length records and KSAM files can be opened as User Mapped Files only if they are also opened with read only access.

Concurrent User Mapped File Access

When a User Mapped File is opened with a Short Pointer, space for that file is allocated from a finite set of virtual addresses that I will refer to here as **Short Pointer Space.** This space for Short Pointers with finite capacity is a central pool of space for the entire system. Because it is a finite resource, its usage is limited to a maximum of four megabytes per file opened with a Short Pointer, and a maximum of six megabytes of Short Pointer Space used for User Mapped

Files at any one time. A four megabyte file is equivalent to a file holding about 15000 sectors while a limit of six megabytes for all files opened with Short Pointers is equivalent to files consuming about 23000 sectors.

An additional consideration regarding pointers is also related to the finite capacity of Short Pointer Space. If a file is first opened as a User Mapped File with a Long Pointer, any subsequent attempt to open the file as a User Mapped File with a Short Pointer will be denied because the file cannot be re-mapped into Short Pointer Space. If, however, the file was first opened as a User Mapped File with a Short Pointer, it could be opened as a User Mapped File with a Long Pointer with the above file size constraints. The default for both HPFOPEN and FOPEN is to open a file with a Long Pointer. Once a file has been opened with HPFOPEN (using defaults), FOPEN or User Mapped File Access with a Long Pointer, it cannot be opened with User Mapped File Access using a Short Pointer until all accessors have closed the file.

Protection ID's

Another consideration involving User Mapped File Access involves the way the memory manager keeps track of the pages that are in main memory. The memory manager maintains a table called the Page Directory (PDIR). There is one entry in the Page Directory for each page in memory. Each entry contains information about the page such as its disc address and a Protection ID or PID. Every time a page is referenced (such as when a record in a User Mapped File is read or written), the page directory is searched for the referenced page. This search is performed in hardware and is very fast (usually completing in 1/2 of one machine cycle). Once the entry is located, the Protection ID is compared to a cache of Protection ID's for the process which has just referenced the page. If the Protection ID is not found, a software routine must be invoked (an expensive operation) to determine if access to this page can be granted to this process. Since the Protection ID's are associated with the process, multiple processes that share pages with other processes or access a large number of pages randomly will require the software routine to be invoked frequently to determine if access can be granted to the new process. This software routine will add the Protection ID to the processes Protection ID cache and remove the oldest entry in the cache. Consider the following example:

Assume a process was accessing a User Mapped File. Assume

that this process also accessed many other memory structures such as code objects, data objects and other files. When the process began, it would access the Mapped File and the file's Protection ID would be added to the Protection ID cache for the process. If the process continued to access other memory structures, it is possible to create a situation where system software would have to be invoked every time a page was referenced to add the new Protection ID to the user Protection ID cache. This Protection ID thrashing could degrade system performance when compared to using the file system since the file system does not use the users Protection ID cache. The savings of bypassing the file system however, will usually allow improved performance when compared with access through the file system. We have not seen any cases where shared User Mapped Files have degraded performance, but the potential is there at least in theory.

Applications for User Mapped Files

There are many applications for User Mapped Files. This efficient access method can be used to enhance the performance of many applications. The most obvious application for User Mapped Files is when a new application is being designed that requires random access. User Mapped Files could be implemented to store and retrieve data quickly and efficiently.

User Mapped Files Replace Extra Data Segments

When migrating an application from MPE V to MPE XL, there are several MPE V features which have been duplicated in MPE XL, but may not be as efficient as another MPE XL feature. MPE V Extra Data Segments (XDS) is one such feature that has been provided in MPE XL, but it is still in compatibility mode and the original MPE V design remains which prevents it from performing as well as it could. One use of User Mapped Files is to replace the Extra Data Segment procedure calls with calls to user written native mode procedures that access User Mapped Files.

One approach to this problem would be to write several procedures with the same names as the Extra Data Segments (GETDSEG, DMOVIN, DMOVOUT, FREEDSEG, ALTDSEG). These procedure could then be added to a library that is referenced by the main program which uses the Extra Data Segment Intrinsics. The translation could be made between what the Extra Data Segments expect and a User Mapped File in the user

written procedures. These procedures could be adapted to work with several different applications and even third party applications or applications where source code is not available.

Summary

User Mapped Files are a powerful tool to add to the programming and design tool kit. As with most new features, there are drawbacks, but in most cases, the drawbacks are far outweighed by the benefits of User Mapped Files. Impressive performance improvements have been observed from a variety of applications running on MPE XL. User Mapped Files is one benefit of the new Hewlett Packard Precision Architecture which will prove very beneficial.