Analysis and Correction of Fragmentation Problems

By Stan Sieler of Allegro Consultants, Inc. August 14, 1995

Mr. Sieler has worked with operating systems since 1973. He entered the HP 3000 community in 1979, working for Hewlett-Packard until he left to co-found Allegro Consultants. At HP, Mr. Sieler led the design of major enhancements to IMAGE/3000, the file backup system, and a major portion of the MPE/iX operating system. He participated in a number of inter-division task forces, chairing one on computer security, a special interest of his. He has a BA in Computer Science from U.C. San Diego and has done graduate work at UCSD and Stanford.

Allegro Consultants, Inc. is in partnership with Lund Performance Solutions.

To contact Allegro Consultants, Inc. or Mr. Sieler:

Allegro Consultants, Inc.
20700 Valley Green Drive
Cupertino, CA 95014-1704
U.S.A.

Voice: (408) 252-2330
Fax: (408) 252-2334
Email: sieler@allegro.com
www: http://www.allegro.com

Tutorial Goals

An understanding of disk and file fragmentation on MPE/iX, and an introduction to the internals of Volumes, Disks, Files, and Databases.

Tutorial Organization

This tutorial is broken into four sections: definition of terms, exploration, analysis, and correction.

The definition of terms section will define various terms used throughout the tutorial.

The exploration section looks at information about volume sets, volumes, disks, files, and databases using widely available free tools, and some third party products.

The analysis section defines fragmentation on a per disk basis, a per-file basis, and a per-database basis, and uses various tools to explore the status of files and disks.

The correction section demonstrates the use of various tools to control all three types of fragmentation.

Definition of terms

This section will define the terms:

- Volume Set
- Directory
- File Label
- HFS
- Label Table
- UFID
- Page & Sector
- Extent
- Extent Block
- SSM
- GUFD
- Extent B-Tree
- Database
- FOS

Volume Set, Volume Class, Volume Name

A Volume is a disk drive. Every volume has a name (e.g., "MEMBER2"). Every in-use volume belongs to exactly one volume Set. The first (original) volume in each volume set is called the Master Volume.

A Volume Set is a set (list) of consisting of one or more volumes (named disks). (The maximum number of volumes in a single volume set is 255.)

A Volume Class is a named subset of a volume set, and consists of one or more volumes. Note that a particular volume may belong to more than one volume class. Since a volume class is a subset of a volume set, it cannot contain volumes belonging to more than one volume set. (The maximum number of volume classes in a volume set is 255.)

Example

We have three disk drives, currently mounted on ldevs 11, 12, and 13. Ldev 11 is a big drive (4.2 GB), ldev 12 is a small (500 MB) but fast drive, and ldev 13 is a medium size (1GB) drive. Currently, the three disk drives have the following volumes mounted, all part of volume set USERS:

Ldev	Volume	Volume_Set	notes
11	USER1	USERS	big
12	USER2	USERS	fast
13	USER3	USERS	

- Volume Set USERS has three volumes: USER1, USER2, and USER3.
- Volume Class DISC has three volumes: USER1, USER2, and USER3.
- Volume Class FASTUSER has one volume: USER2.
- Volume Class BIG has one volume: USER1.
- Volume Class NOTFAST has two volumes: USER1 and USER3.

NOTE Strictly speaking, the volume classes shown above should have a prefix of "USERS:", to indicate that they belong to the volume set USERS.

A BUILD (or FOPEN) of a new file in a group that is "HOMEVS"'d to the USERS volume set will be built as follows:

DEV=	Volume used
omitted ("DISC")	any (USER1, USER2, USER3)
11	USER1
12	USER2
13	USER3
FASTUSER	USER2
BIG	USER1
NOTFAST	either USER1 or USER3
FOO	error (unknown volume class name)
9000	error (maximum disk ldev is 8192)
1	error (ldev 1 is in the volume set MPEXL_SYSTEM_VOLUME_SET)

Note that the above example didn't say "Ldev used", but instead "Volume used". Thus if you changed the ldev numbers of the three disks (say, to 200, 5, and 333), rebooted, and tried again most of the above examples would have the same results (i.e., "BIG" would result in USER1). The only ones that would "change" are the ones where a specific ldev is requested. The ldev is translated to a volume at that time (e.g., originally ldev 11 was translated to USER1, and after the change/reboot, ldev 11 would be an error, and ldev 200 would translate to USER1).

Most users don't change the ldev -to- volume mapping once a volume is created. However, on a system with several removable disks (e.g., 7935 disk drives), it is fairly likely that a given disk pack will be inserted into random drives from time to time. MPE/iX is extremely resilient, in that there is no hardcoded mapping between ldev and volume. At bootup (or volume set open) time, MPE builds a table of ldevs/volumes, corresponding to what it sees actually mounted.

The "system disk" of MPE/iX is actually just another volume set, MPEXL_SYSTEM_VOLUME_SET. Note that although long volume set names are allowed, they aren't required! At our site, we have two extra volume sets: USERS and TEST.

At bootup, MPE/iX tries to "open" every volume set that is mounted. With a partial exception for MPEXL_SYSTEM_VOLUME_SET, MPE will not open a volume set that doesn't have every volume mounted. Instead, it will wait until all volumes are mounted.

Directory

The "directory" is the set of data structures that keeps track of file names, account names, group names, and user names.

The directory is implemented as a set of hierarchical files, with the entries in each file being kept in alphabetic order.

The top-level file, the "root" of the system directory (known internally as $ROOT (or "/" from the POSIX viewpoint)), contains a list of all account and, as of MPE/iX 4.5, all root-level files and directories (e.g., "/foo" or "/tmp").

There are three kinds of entries in the $ROOT file:

- POSIX file names (e.g., /foo)
- POSIX directories (e.g., /tmp)
- MPE account names (e.g., ALLEGRO, or SYS)

Every account entry has two "pointers". The first of these "pointers" is to a file containing a list of all groups in the account. This file is called a $GROUP_NODE, or (in POSIX), /accountname (e.g., /SYS). The second "pointer" is to a file containing a list of all users in the account. This file is called a $USER_NODE, and currently has no POSIX name.

All other levels of the directory are simply files that contain a list of files and directories at that level. For example, the file /SYS/PUB contains a list of all files in @.PUB.SYS.

The "lists of files and directories" is a sorted, alphabetic order list. Each entry is a pair consisting of a name (e.g., "EDITOR") and a "pointer" to the file label for the file. The "pointer" is actually a UFID (Unique File IDentifier), which can be thought of as an ldev and a sector offset of the file label, although this is overly simplified.

File

For this tutorial, a "file" means "a disk file", not a printer, or a terminal, or any other kind of non-disk file.

File Label

A file label is a 512 byte data area that is used to keep track of information about a file. This information includes: creation date, record size, creator, file name (usually), and more. In a few cases, the file label will point to "extension labels" that have additional information (e.g., ACD security).

If a file has disk space allocated, the file label will point to the start of a list of records that keep track of disk allocation for the file (see Extents and Extent Blocks below).

HFS

With the release of MPE/iX 4.5 and 5.0, MPE has been enhanced to support a full featured hierarchical file system, like MULTICS or MCP. (Some would say "like UNIX", but UNIX is a latecomer here.)

HFS (Hierarchical File System) refers to the POSIX file names that aren't representable as MPE file names. (E.g., /tmp)

Label Table

A Label Table is a file that stores file labels and extent blocks (see below).

On MPE/iX, a file label is not contiguous with the first record of data of the file. In fact, it is possible for a file label to exist, and no other disk space be associated with the file. (Such a file would show as having 0 sectors in a LISTF,2 command.)

On MPE V, the file label for a file immediately preceded the first record of data for the file on disk. (This also implied that MPE couldn't allocate a file label on disk without also allocating the first extent of the file's data.)

Every volume has a Label Table. Using our "USERS" volume set from above, the volumes USER1, USER2, and USER3 each have a Label Table. If you BUILD/FOPEN a new file on USERS, MPE (or you) picks a volume to put the file on (or, at least, the file label). If USER1 was picked, for whatever reason, the file label for the new file will be stored in the Label Table on USER1. Generally, but not always, the first extent of a file's data will be allocated on the same disk drive as the file's label is on.

When a volume is mounted, MPE "maps" the Label Table into a virtual address range, allowing it (and us) to view the SSM via LOAD/STORE instructions (or via Debug's DV command).

When you do a LISTF/LISTFILE with the -3 option, the virtual address shown for the file label is within a Label Table.

Although not guaranteed, the Label Table for ldev 1 has always started at virtual address $11.0.

UFID

A UFID (Unique File IDentifier), is a "pointer" to a file. In essence, the UFID can be thought of as specifying a volume and an index into the Label Table on that volume where a file label is found. A UFID is 20 bytes of binary data.

The entries in the directory referred to earlier as "pointers" are UFIDs.

Page, Sector

On MPE/iX, disk is allocated in units of a page, where a page is 4,096 bytes.

On MPE V, disk was allocated in units of sectors, where a sector is 256 bytes.

You might encounter disk diagnostic utilities that report that a disk drive has a sector size of 512 bytes instead of 256. This is because some disk drives have a minimum hardware addressable unit of 512 bytes instead of 256. However, this tutorial will always mean "256 bytes" when referring to a "sector". In all cases, every disk page starts on a 4,096 byte boundary on the disk.

Extent

An extent is a contiguous block of disk data allocated to a file.

Extents are always allocated on page boundaries on disk, and always occupy an integral number of pages. Thus, the smallest file that actually has disk data will occupy at least 1 page, or 4,096 bytes, or 16 sectors.

On MPE V, no file could have more than 30 extents, and no file could have less than 1 extent.

On MPE/iX, a file can have 0 or more extents. There is no hard coded limit, the number of extents is stored in a 16 bit field, thus imposing a functional limit of 65,535 extents.

Extent Block

An extent block keeps track of up to 20 extents for a single file. Extent Blocks are stored in the same Label Table as the associated file's file label.

If a file has at least one page of disk allocated to it, it will have at least one extent block in use in the Label Table.

When all 20 entries in an extent block are used, the next request for allocating disk space for the file will cause a new extent block to be allocated. The entries are not sorted in any order whatsoever.

A sample extent entry in an extent block looks like:

```
EXTENT_SECTOR_ADDR : 3f7ee0
SECTORS_IN_EXTENT  : 800
VOL_SET_INDEX      : 1
FILE_SECTOR_OFFSET : b98c00
```

The VOL_SET_INDEX specified which volume of a volume set the extent is on ... which means that it isn't necessarily on the same disk that the extent block entry is on.

The EXTENT_SECTOR_ADDR specifies where on the disk drive the file is located. Note: this is always in units of 256 byte sectors, and is always a multiple of 16 sectors (4,096 bytes).

The SECTORS_IN_EXTENT records the size of the extent. It is always a multiple of 16 sectors (4,096 bytes).

The FILE_SECTOR_OFFSET specifies what part of the file the extent is associated with. The extent corresponding with the first record of a file would have FILE_SECTOR_OFFSET of 0.

SSM

The SSM (Secondary Storage bitMap) is a data structure on every volume that keeps track of which pages of the disk are free, and which are in use.

The main portion of the SSM for a disk is a large bitmap, with one entry per page, recording whether or not the page is free.

When a volume is mounted, MPE "maps" the SSM into a virtual address range, allowing it (and us) to view the SSM via LOAD/STORE instructions (or via Debug's DV command).

GUFD

A GUFD (Global Unique File Descriptor) is a virtual memory data structure (record) allocated (usually) when a file is first opened.

If a single process has a particular file open, that file has a GUFD. If two thousands processes open the same file at the same time, that file still has a single GUFD.

When the last user of a file closes it, MPE generally does not deallocate the GUFD. Instead, it puts it on a list of recently "closed" GUFDs. It also doesn't deallocate the virtual space address range associated with the file, and it leaves in memory any pages associated with the file (presumably after making sure that any dirty ones are written to disk).

Later, when a process tries to FOPEN the file again, MPE checks to see if it is on the recently closed list ... if it is, then that GUFD is taken off the list and used again for the same file. When this happens, the same virtual address range is assigned to the file, which means that if pages of it were still in memory, the user may benefit (performance wise).

If the list gets too large, the oldest entries on the list will be deallocated and forgotten.

If the file isn't on the recently closed list, a fresh GUFD will be allocated, along with a fresh virtual address range.

Extent B-Tree

An Extent B-Tree is a virtual memory data structure that is used to map from a "file-relative" byte offset back to the equivalent page of disk storage. There is one entry in an extent B-tree for each extent of a file.

Each open file has its own extent B-tree.

Because there is one entry in an extent B-tree for each extent of a file, a 1 MB file with 100 extents will require a much larger extent B-tree than a 1 MB file with a single extent.

When a file is first opened, a GUFD is assigned and the file's extent blocks are read from the Label Table (where the file label is stored), and the extent B-tree is created.

Database

For this tutorial, "database" refers only to IMAGE/SQL databases.

FOS

FOS (Fundamental Operating System) is the set of software that is bundled as part of MPE.

Exploration

Now that we've defined terms, let's explore: volume sets, disks, label tables, file labels, open files and databases.

Exploring: Volume Sets

MPE appears to lack a command to display all defined volume sets, although it does have one that will tell us what volume sets are currently mounted.

The DSTAT command, part of FOS, will quickly show all mounted volume sets (and volumes):

```
:DSTAT ALL
LDEV-TYPE    STATUS     VOLUME (VOLUME SET - GEN)
---------    --------   --------------------------
1-C2474S     MASTER     MEMBER1 .....(MPEXL_SYSTEM_VOLUME_SET-0)
2- 022030    MASTER     MASTER       (USERS-0)
3- 022030    MEMBER     USER2        (USERS-0)
4- 022030    MEMBER     USER3        (USERS-0)
```

This shows us that two volume sets are currently mounted: MPEXL_SYSTEM_VOLUME_SET and USERS. Note that the master volume for USERS is called "MASTER" ... a distinctly uninformative name. I'm at fault here, and if I were to rebuild this volume set again, I'd call the master volume USER1.

NOTE Some users name their volumes after the ldev they were initially created on. This may not be desirable, for a variety of reasons, including the fact that the ldev to volume name mapping may change if the disks are moved around or the ldevs are changed.

De-Frag/X1 also has a DSTAT command, which provides more internal information:

```
:defragx
dstat all

Ldev mvid  Volume Set Name            :Volume Name      Physical Path
---- ----  ---------------------------- ---------------- -------------
1     1  MPEXL_SYSTEM_VOLUME_SET      :MEMBER1          52.6.0
2     2  USERS                        :MASTER           48.0.2
3     3  USERS                        :USER2            48.0.3
4     4  USERS                        :USER3            48.0.4
16   (not "mounted")

     S
     y
     s
Ldev ? MVT entry LM Area   SSMdevinf SSM map    #Pages  #MBs  #SSMupdate
---- - --------- --------- --------- ---------  ------ -----  ----------
   1 Y $c3fd8e00 $c3fd8f84 $c3fd8eac $d2c00000 330,884 1,292  1,591,869
   2   $c3fd9000 $c3fd9184 $c3fd90ac $d5288000 163,736   639     78,701
   3   $c3fd9200 $c3fd9384 $c3fd92ac $d52a8000 163,736   639     71,248
   4   $c3fd9400 $c3fd9584 $c3fd94ac $84bb8000 163,736   639     55,544
  16   $00000000  (not fully mounted)

Ldev  LT GUFD   LT Header LT EOF    Sectrs  Version  SecSize
----  --------- --------- --------- ------  -------- -------
   1  $ca000054 $011.$300 $00900f00  1,760  A.00.00      512
   2  $ca00b43c $125.$300 $00900f00  1,440  A.00.00      256
   3  $ca00b764 $129.$300 $00900f00  1,440  A.00.00      256
   4  $ca00b8f8 $12c.$300 $00900f00  1,440  A.00.00      256
  16  (not "mounted")
```

Note the "#SSMupdate" column. This is the approximate number of disk space allocations and deallocations (combined) that have occurred on the disk drive. MPE may drop this number to zero from time to time, although I haven't noticed it yet.

The DISCFREE command, part of FOS, will also give us an insight into volume sets and volumes:

```
:discfree c

DISCFREE A.50.01 Copyright (C) Hewlett-Packard 1992. All rights
reserved.                          FRI, AUG 11, 1995,  6:46 PM

ALL MEASUREMENTS ARE IN SECTORS.
ALL PERCENTAGES ARE RELATIVE TO THE DEVICE SIZE.

           |    Configured      |      In Use        |     Available      |
-----------+--------------------+--------------------+--------------------+

LDEV :     1 -- (MPEXL_SYSTEM_VOLUME_SET:MEMBER1)
 Device    |    5294144         |    4483344 ( 85%) |     810800 ( 15%) |
 Permanent |    4923552 ( 93%) |    4095248 ( 77%) |     810800 ( 15%) |
 Transient |    5294144 (100%) |     388096 (  7%) |     810800 ( 15%) |

LDEV :     2 -- (USERS:MASTER)
 Device    |    2619776         |    2551488 ( 97%) |      68288 (  3%) |
 Permanent |    2619776 (100%) |    2551488 ( 97%) |      68288 (  3%) |
 Transient |    2619776 (100%) |          0 (  0%) |      68288 (  3%) |
...

TOTALS :
 Device    |   13153472         |   12096208 ( 92%) |    1057264 (  8%) |
 Permanent |   12756688 ( 97%) |   11708112 ( 89%) |    1031072 (  8%) |
 Transient |   13101072 (100%) |     388096 (  3%) |    1057264 (  8%) |
```

DISCFREE has five different formats for its output, if you run it without specifying an output format, it will prompt you for one.

De-Frag/X also has a form of the DISCFREE command:

```
:defragx
discfree
MPEXL_SYSTEM_VOLUME_SET :
          Avail    Avail
Ldev      Pages      MBs  %Avail Volume Name
----  ---------  ------  -----  ----------------
   1     50,776     198     15%  MEMBER1
total MBs available:   198  ( 50,776 pages;    812,416 sectors)
---------------------------------------------------------------
```

```
USERS :
          Avail    Avail
Ldev      Pages      MBs  %Avail Volume Name
----  ---------  ------  -----  ----------------
   2      4,268      16     2%  MASTER
   3      5,417      21     3%+ USER2
   4      4,082      15     2%  USER3
total MBs available:    52  ( 13,767 pages;    220,272 sectors)
---------------------------------------------------------------
```

NOTE Disks flagged with "+" after their %Avail have a VOLUTIL configured "Maximum Permanent Space" of less than 100%.

De-Frag/X's DISCFREE defaults to grouping the ldevs by volume set, but this can be overridden.

Once we know what volume sets are mounted the VOLUTIL utility, a part of FOS, can be used for further exploration:

```
:VOLUTIL

Mirvutil  A.01.00, (C) Hewlett-Packard Co., 1991. All Rights Reserved.
volutil: showset users

Volume-set name: USERS
Creation date: SUN, MAY 28, 1995,  7:55 PM
Generation number: 0
Number of volumes in set: 3
Number of classes in set: 1

volutil: showset users;info=struct

Volumes in set: USERS

MASTER
USER2
USER3

Classes in set: USERS

DISC

Volumes in class: USERS:DISC
MASTER
USER2
USER3
```

VOLUTIL will also show some internals information:

```
:volutil
showset users; labels

Volume name: USERS:MASTER
Initialization date: SUN, MAY 28, 1995,  7:55 PM      Volume type: 2
Member number: 1     Number in set: 3
Label Table Address: $000005A0         MVT Address: $00000000
Free Space Map Address: $00000090      Cold Load ID: $00000000
Logical Volume-id: $055F0001 2513DCD4
Physical Volume-id: $055F0001 2513DCD4
```

Exploring: Disk Layout

The first few sectors of every volume has information about the which volume set the disk belongs to.

We can use Debug, carefully, to explore this information:

```
:debug
dsec 1.$100, 40, s, 8
```

The [previous Debug command] will display the second sector of ldev 1, as ASCII text, 8 32-bit words per line:

```
SEC $1.100      "....MPEXL_SYSTEM_VOLUME_SET     "
SEC $1.120      "    ng..../..ML.........._..%..."
SEC $1.140      "....J..D.q..._..%.......J..D.q.."
SEC $1.160      "........_..%..................."
SEC $1.180      "... ...............7._..%.....w."
SEC $1.1a0      "J..@.y@.........._..%.....h.J..a"
SEC $1.1c0      ".t.i.MEMBER1         ...._..%..."
SEC $1.1e0      "..............................."
```

We see that this disk is MPEXL_SYSTEM_VOLUME_SET:MEMBER1.

NOTE Sector 0 generally contains information in a format called "LIF" (Logical Interchange Format). One benefit of LIF is that it would, in theory, allow HP-UX to recognize an MPE/iX disk.

Here is sector 0 of ldev 1 on one machine:

```
:debug
dsec 1.0,40,s,8

SEC $1.0        "..HPESYS........................"
SEC $1.20       "..............................."
SEC $1.40       "..............................."
```

```
SEC $1.60      "................................"
SEC $1.80      "................................"
SEC $1.a0      "................................"
SEC $1.c0      "................................"
SEC $1.e0      ".................0...‘........."
```

NOTE When exploring disks with Debug's dsec ("Display SECondary storage") command, be sure that the ldev number ("1" in the examples above) you use always corresponds to a mounted, spinning disk drive. If you do a "dsec" on a disk drive that is configured (via SYSGEN), but not available, your Debug session will hang until you reboot.

Exploring: SSM

The SSM (Secondary Storage bitMap) is a data structure on every volume that keeps track of which pages of the disk are free, and which are in use. If you get the virtual address of the SSM for a disk drive, Debug can be use to explore it. Here's one method:

```
:defragx
dstat 1 detail
...
     s
Ldev ? MVT entry LM Area   SSMdevinf SSM map       #Pages
---- - --------- --------- --------- ---------   ---------  ...
   1 Y $c3fd8e00 $c3fd8f84 $c3fd8eac $d2c00000    330,884
...

:debug
dv $d2c00000, 20
```

Note that the first $200 bytes of the SSM are a "header".

You can format the SSM by doing:

```
:debug
symopen symos.osb79.telesup      (or appropriate osXXX group)
fv $d2c00000  "sec_storage_map_type"
```

To see the status of a particular page (e.g., page #12345), do:

```
fv $d2c00000  "sec_storage_map_type.map [#12345]"

CRUNCHED RECORD
PAGE_STATE      : 2
STORAGE_OPTIONS : [ ]
END
```

The PAGE_STATE of 2 means "permanent".

Other page state values are:

```
0  free
1  transient  (e.g., stack, heap, MPE data structures)
2  permanent
```

De-Frag/X has an "SSM" command that allows exploring the state of pages in the SSM:

```
:defragx
ssm 1 12345

Page  12,345 : permanent (info @ $d2c01a1c)
```

Exploring: Label Table

The label table for a volume can be explored with FSCHECK. While not an official part of FOS, FSCHECK seems to generally be bundled with MPE/iX. FSCHECK can be found in FSCHECK.MPEXL.TELESUP.

```
:fcheck.mpexl.telesup

FSCHECK, A.05.00. (C) Hewlett-Packard Co., 1987. All rights reserved.

fscheck: displaylabel 1

File labels allocated on volume: MPEXL_SYSTEM_VOLUME_SET:MEMBER1

MMSAVE          .MPEXL          .SYS               - $0000F300
MPEXLDIR        .PUB            .SYS               - $0000F600
ISL             .MPEXL          .SYS               - $0000F900
START           .MPEXL          .SYS               - $0000FC00
...
```

NOTE Be prepared to "Break/ABORT", since FSCHECK doesn't seem to react to control-Y.

De-Frag/X also has the ability to list all, or part of, a Label Table:

```
:defragx
listlt 1 all
```

A UFID (Unique File IDentifier) acts as a "pointer" into a label table.

Here's the UFID for EDITOR.PUB.SYS on one machine ... note that it will vary from machine to machine:

```
$055f0001 $251006f5 $002385be $4a053886 $047f54c0
 Volume              Index
```

The previous UFID can be decoded as:

```
VOLUME_ID      :
   SPLIT     : 0
   FILL      : 5
   VSCTS0    : 5f
   PHIL      : 0
   VOL_SET_X : 1<--- first volume in volume set
   VSCTS1    :
         DAY    : 94
         HOUR   : 10
         MINUTE : 1
         SECOND : 2f
         TENTHS : 5
LABEL_OFFSET   : 2385<--- index into Label Table
FCTS0          : be
FCTS1          :
   DAY    : 128
   HOUR   : 5
      MINUTE : e
      SECOND : 8
      TENTHS : 6
INTERVAL_TIMER : 47f54c0
```

The extra fields are various forms of timestamps, which help determine which volume set the UFID refers to.

Exploring: File Label

You can see a file label, in hexadecimal, via the LISTF or LISTFILE commands, using the "-1" option (which requires you to have SM capability). Unfortunately, this displays only the first 256 bytes, not the entire 512.

Example:

```
:listf editor.pub.sys, -1   (output shown slightly edited)

F = EDITOR
00000000 44495343 20202020 20202020 20202020 20202020 ....DISC
20202020 20202020 20202020 20310000 45444954 4F522020         1..EDITOR
20202020 20202020 50554220 20202020 20202020 20202020         PUB
00000000 53595320 20202020 20202020 20202020 00000000 ....SYS
20202020 20202020 20202020 20202020 4D414E41 47455220            MANAGER
53595320 20202020 00000000 FC000000 045F0001 251006F5 SYS     ....ü...._
00010401 00000000 00000300 0002D69A 28509A60 0002D923 .............Ö.(P
150C5664 0002DEDE 63736820 0002D69A 28509A60 00014300 ..Vd..ÞÞcsh ..Ö.(P
00000150 00000143 00000143 00000142 00000000 00000000 ...P...C...C...B..
00014300 00000000 00000000 00000000 00000100 00000100 ..C...............
00010000 01440001 01440405 00000000                   .....D...D......
```

You can see the entire file label by using Debug, if you know the virtual address of the file label. Fortunately, LISTF and LISTFILE's -3 option displays this information:

Example:

```
:listf editor.pub.sys, -3

FILE: EDITOR.PUB.SYS
...
MAX LABELS: 0                MODIFIED: WED, APR 26, 1995,  2:45 PM
DISC DEV #: 1                ACCESSED: WED, AUG  9, 1995,  7:31 PM
SEC OFFSET: 0                LABEL ADDR: $00000011.$00238520
VOLCLASS  : MPEXL_SYSTEM_VOLUME_SET:DISC
```

Now that we have the virtual address for the file label, $11.$238520, we can display it in hex via Debug:

```
:debug
dv $11.$238520, $80, b
c
```

A file's file label can be found via LISTF/LISTFILE, and viewed via Debug:

```
:listf editor.pub.sys, -3
...
SEC OFFSET: 0             LABEL ADDR: $00000011.$00238520
...

:debug
dv $11.$00238520, 40, b
```

If you tell Debug to open the SYMOS file corresponding to your release of MPE/iX, you can see the file label in much more detail.

The SYMOS version can be determined by doing a ":SHOWME" command, as follows:

```
:showme

RELEASE: C.50.00   MPE/iX HP31900 B.79.06   USER VERSION: C.50.00
```

The underlined area above shows that the appropriate SYMOS file is in the group OSB79.TELESUP. (It's always OSxxx.TELESUP.)

```
:debug
symopen symos.osb79.telesup
fv $11.$00238520 "flab_t"
```

Here's a partial sample from the above "fv":

```
RECORD
VERSION                : 0
FILE_FLAGS             :
   TEMP_FILE      : FALSE
   RELEASED       : FALSE
   IGNORE_PATH    : FALSE
   NO_BACKUP      : FALSE
   RESTORE        : FALSE
   STORE          : FALSE
   PURGE_PENDING  : FALSE
   PROTECTED      : FALSE
LANG                   : 0
ASCII_EXO_RESTRICTION  : 'DISC                                  1'
PRIV_IN                : 0
PRIV_OUT               : 0
FILE_NAME              : 'EDITOR          '
GROUP1                 :
      NAME : 'PUB             '
      MASK :
...
LOCKWORD               : '                '
CREATOR                :
      NAME : 'MANAGER SYS     '
...

FOPTIONS               :
      FILL_BITS     : 0
      REC_FORMATX   : 0
      FILE_TYPE     : 0
      FILE_EQ       : 1
      LABELED       : 0
      CONTROL       : 0
      REC_FORMAT    : 0
      DEFAULT_DESIG : 0
      ASCII         : 0
      DOMAIN        : 1
FILE_DESC              :
   DEVICE_TYPE     : 0
   DEVICE_SUB_TYPE : 0
   FILE_TYPE       : 0
   RECORD_TYPE     : 0
   ACCESS_METHOD   : 0
FILLER2                : 0
PRIV_LEVEL             : 3
FILLER3                : 0
TIME_STAMPS            :
```

```
   CREATED    : 2d69a28509a60
   ALLOCATED : 2d923150c5664
   ACCESSED   : 2df0516fc1a51
   WRITTEN    : 2d69a28509a60
   EOF_OFFSET                : 14300
   SECTORS_IN_FILE           : 150
   LOGICAL_FILE_REC_LIMIT    : 143
   LOGICAL_END_OF_DATA_REC   : 143
   END_OF_FILE_BLOCK         : 142
   MSG_OPEN_REC_CNT          : 0
   MSG_FIRST_BLK_NUM         : 0
   FILE_LIMIT                : 14300
   USER_LABEL_EOF            : 0
   USER_LABEL_CNT            : 0
   SOF_OFFSET                : 0
   SONR_OFFSET               : 0
   REC_SIZE                  : 100
   BLOCK_SIZE                : 100
   BLOCK_FACTOR              : 1
   FIRST_BLOCK_OFFSET        : 0
   EXTENT_SIZE               : 144
   EXTENTS_IN_FILE           : 1
   LAST_EXTENT_SIZE          : 144
   FILE_CODE                 : 405
...
```

De-Frag/X also shows a list of extents for a file:

```
:defragx
de sl.pub.sys

Ldev  Disk Page#  # Pages  File Page #
----  ----------  -------  -----------
 1      44,273       16           0 ,
 1      44,289       16          16 ,
 1      44,305       32          32 ,
 1      44,337       64          64 ,
 1      44,401      128         128 ,
 1      44,529      128         256 ,
 1      44,657      128         384 ,
 1      44,785      128         512 ,
 1      44,913      128         640 ,
 1      45,041      128         768 ,
 1      45,169      128         896 ,
 1      45,297      128       1,024 ,
...
 1      52,081      128       7,808
```

```
# extents in file:          65
# pages in file:   7,936 pages; 31.0 MBs
Space savable by TRIM:  132 pages;  0.5 MBs
% fragmented:    0.0+
```

Exploring: Open Files

Open files can be explored in several manners. The most straight forward method is to determine the GUFD for a file of interest. This can be done by opening the file, and then calling FFILEINFO to obtain the GUFD (a 32-bit integer).

Alternatively, De-Frag/X can be used to search the list of open files on the system:

```
:defragx
findfile all file gufd

VSOD      GUFD      Virtual Address #DskPgs FileLabelAddr Filename
--------- --------- --------------- ------- ------------- --------
$ca000000 $ca000054 $0011.$00000000   1,536 $0011.$000620 $LABEL_TABLE
$ca000194 $ca0001e8 $0014.$00000000     110 $0011.$001220 $SYSTEM_VSIT
$ca000328 $ca00037c $0015.$00000000   3,258 $0011.$000c20 $xm system log
$ca0004bc $ca000510 $0018.$00000000      12 $0011.$004820 /SYS/PUB
$ca000650 $ca0006a4 $00c8.$00000000      10 $0125.$4d6720 UDC2.MISC.SIELER
$ca0007e4 $ca000838 $08d1.$00000000       2 $0011.$1deb20 /SYS/LIB
$ca000978 $ca0009cc $001d.$00000000       6 $0011.$001820 $ROOT
...
```

Every open file has a GUFD (Global Unique File iDentifier). This controls global access to the file, including for FLOCK/ FUNLOCK.

Once you have the virtual address of a GUFD, you can use Debug to display the GUFD. Here's an example, looking at the GUFD for /SYS/PUB (we got the GUFD from the output above):

```
:debug
symopen symos.osb79.telesup
fv ca000510 "gufd_t"

RECORD
   HASH_LINK                           : 0
   LRU_LINK                            : 0
   PREV_LRU_LINK                       : 0
   UFID                                :
         ALL : ... (binary data shown as ASCII)
   GUFD_COUNT                          : c9
   FILE_OPEN_CNT                       : 47
   FILE_LABEL_PTR                      : 11.1820
   LPTR_FILE_TYPE                      : TRUE
   VERIFY_W_FLAG                       : TRUE
```

```
FLOCK_SPECIFIED                    : FALSE
UNPROTECTED                        : FALSE
...
FLOD_INVALID                       : FALSE
FLOD_PTR                           : 0.0
SEMAPHORE                          :
...
FLOCK_SEMAPHORE                    :
      ...
         SEM_OWNER : 7ffd                <-- $7ffd = unowned
      SEM_SPIN_STATE_REC    :
...
OPEN_SEMAPHORE                     :
      ...
         SEM_OWNER : 7ffd
...
FILE_VIR_ADDR                      : 18.0
GDPD_PTR                           : d4880220
...
TOT_READER                         : 47
TOT_WRITER                         : 0
FILE_SHARING_MODE                  :
   ...
   FLAB_DIRTY   : FALSE
   EXCLUSIVE    : FALSE
   WRITE_OPTION : NORMAL_WRITE
   SEMI_COUNT   : 0
   LOAD_BIT     : FALSE
TM_EOF_OFFSET                      : 0
REFUSE_ACCESS                      : FALSE
SHADOW_LOG                         : FALSE
XM_POST_IN_PROGRESS                : FALSE
LRU_FILE_VIR_ADDR_FROM_LAST_CLOSE  : FALSE
FCLOSE_DISP                        : 0
NEW_FILE                           : FALSE
GDPD_COUNT                         : 1
CCTL                               : FALSE
ASCII_FILE                         : FALSE
...
EOF_OFFSET                         : 15a8
SECTORS_IN_FILE                    : 0
LOGICAL_FILE_REC_LIMIT             : 0
CURR_NUM_REC                       : 0
END_OF_FILE_BLOCK                  : 0
MSG_OPEN_REC_CNT                   : 0
MSG_FIRST_BLK_NUM                  : 0
FILE_SIZE                          : 1000000
USER_LABEL_EOF                     : 0
```

```
      MAP_OUT_MUST_POST                  : TRUE
      PROTECTION_METHOD                  : 0
      PROTECTION_CNT                     : 0
      PID                                : 6
      SOF_OFFSET                         : 0
      XM_PTR                             : 817046dc
      XM_SPAN_LS                         : 0
      ...
      STORE_ACTIVE              : FALSE
      ...
   END
```

As you can see, there are a number of interestingly named fields in the GUFD!

In addition to a GUFD, every open file has another data structure associated with it, the VSOD (Virtual Space Object Descriptor). Every non-file virtual address range allocated by MPE, also has a VSOD associated with it. (Example: your NM stack has a VSOD, but no GUFD because it isn't a file.)

De-Frag/X reports the currently in-use VSODs with the FINDSID command (shown above). However, if you know the GUFD for a file (perhaps from calling the FFILEINFO intrinsic), you can determine the address of the GUFD by subtracting hex $54 from the GUFD address.

Here's a sample VSOD, for the open file /SYS/PUB:

```
   :debug
   fv ca0004bc fum "vs_od_type"

   RECORD
      SEMAPHORE              :
         SEM_INFO_WORD :
            ...
            SEM_OWNER : 7ffd
         ...
      BASE_VA                : VA_TYPE( 18.0   )
      ENDING_VBA             : ffffff
      CURRENT_SIZE           : 1000000
      OPTIONS                : [ 0, 3, 4, 5, 7, 8, 9, 13]
      VIRTUAL_LOCATION       : OBJ_OWN_SPACE
      ACCESS_RIGHTS          :
         PAGE_USAGE : 1
         PL1        : 2
         PL2        : 2
         FILLER     : 0
      PID                    : 0
      NEW_OPTIONS            : [ ]
      INIT_BYTE              : 0
      OBJECT_CLASS           : 7
      SHARED_VS_UNIT_DESC_ID : 0
         CURRENT_SEC_STORAGE_PAGES : 2
         MAX_SEC_STORAGE_PAGES     : 1002
```

```
      B_TREE_ROOT                :
         ROOT_PTR     : cc000e58
         NODE_COUNT   : 1
         ENTRY_COUNT  : 1
      DEV_RESTRICTION            :
         ...
      VS_LL_HEADER_NUM           : 18
         FIRST_SHR_PAGE_DESC_PTR : 0
         LAST_SHR_PAGE_DESC_PTR  : 0
END
```

Note that like most MPE internal data structures, the VSOD has a lock area within it (the "SEMAPHORE" field). The SEM_OWNER value of $7ffd means that no one owns the semaphore at present.

The "B_TREE_ROOT" section has a field called "ENTRY_COUNT". This is the number of extents the file has.

Here's a look at the VSOD for NL.PUB.SYS:

```
De-Frag/X> findsid $a file

VSOD      Virtual Address #DskPgs FileLabelAddr Filename
--------- --------------- ------- ------------- --------
$ca000978 $000a.$00000000   7,040 $0011.$015c20 NL.PUB.SYS

(Examined 1,340 objects)
# disk pages occupied by listed objects:     7,040

De-Frag/X> :debug
HPDEBUG Intrinsic at: a.009ea6a8 hxdebug+$e4
$a ($39) nmdebug > var foo ca000978
$b ($39) nmdebug > fv foo "vs_od_type"

RECORD
   SEMAPHORE               :
      ...
         SEM_OWNER : 7ffd
       ...
   BASE_VA                 : VA_TYPE( a.0   )
   ENDING_VBA              : 3e7fffff
   CURRENT_SIZE            : 3e800000
   OPTIONS                 : [ 0, 4, 5, 6, 7, 8, 9, 1c, 1e]
   VIRTUAL_LOCATION        : OBJ_SR4
   ACCESS_RIGHTS           :
      PAGE_USAGE : 0
      PL1        : 0
      PL2        : 0
      FILLER     : 0
   PID                     : 0
   NEW_OPTIONS             : [ ]
```

```
      INIT_BYTE              : 0
      OBJECT_CLASS           : 8
      SHARED_VS_UNIT_DESC_ID : 0
         CURRENT_SEC_STORAGE_PAGES : 1b80
         MAX_SEC_STORAGE_PAGES     : 100000
         B_TREE_ROOT               :
            ROOT_PTR    : ce000410
            NODE_COUNT  : 1d
            ENTRY_COUNT : dd
         DEV_RESTRICTION           :
               EXO_SPLIT       : 0
               EXO_RESTRICTION : 1
               EXO_TS0         : 0
               EXO_INDEX       : 25
               EXO_TS1         :
                     DAY    : 135
                     HOUR   : 21
                     MINUTE : 1a
                     SECOND : 36
                     TENTHS : 5
         VS_LL_HEADER_NUM          : 1c
            FIRST_SHR_PAGE_DESC_PTR : d1ff8000
            LAST_SHR_PAGE_DESC_PTR  : 0
   END
```

Note that the B_TREE_ROOT.ENTRY_COUNT field has the value $dd (decimal 221). This indicates that NL.PUB.SYS has 231 extents.

Of course, we could have found out how many extents NL.PUB.SYS has by simply doing:

```
listf nl.pub.sys, 2
```

But that wouldn't have been as much fun!

The Extent B-Tree is the data structure that keeps track of those 221 extents for NL.PUB.SYS. If you try to access address $a.$34000, and the page isn't in memory, a page fault occurs. The VSOD is consulted to see if $a.$34000 is a legal address within the file. (It is, since the VSOD (above) says that the ending address is $a.$3e7fffff.)

The B_TREE_ROOT points to the start of the Extent B-Tree. Each of the 221 entries in the Extent B-Tree will describe a single extent of the file NL.PUB.SYS.

Clearly, even an efficient search of a B-Tree with 221 entries must take more time than a search through a B-Tree with only 1 entry...and that's one area where file fragmentation can affect performance. In this case, the amount of extra CPU time required to search the extra B-Tree entries is fairly small ... probably on the order of a hundred microseconds. There is, however, a second penalty to file fragmentation: the memory space occupied by the B-Tree increases with every 20 (or so) extents. Thus, a file with a large number of extents will have a larger B-Tree, using more memory than a defragmented file would. Again, this isn't a terribly large amount of memory .. for an individual file, and the data isn't even marked as "memory resident". However, this means that there is an increasing possibility that the B-Tree for your file won't be in memory when it's needed to process a page fault, which will result in one (or more) subsequent page faults that must be resolved before your page fault is completely resolved.

Analysis

This section discusses analysis of the three types of fragmentation: disk, file, and database.

Analyzing Fragmentation: Disk

A rough picture of disk fragmentation can be seen with the FOS tool DISCFREE. A DISCFREE A displays a summary of how many free areas exist, grouped by size. If a large number of small free areas exist, and the largest free area isn't very large, then you can conclude that your disk is fragmented.

```
:discfree a

DISCFREE A.50.01 Copyright (C) Hewlett-Packard 1992. All rights reserved.
                         WED, AUG 16, 1995,  5:39 PM

-------------------------------------------------------------------------------
LDEV :     1 -- (MPEXL_SYSTEM_VOLUME_SET:MEMBER1)

LARGEST FREE AREA:      1977664  TOTAL FREE SPACE:    3768624

     0 BLOCK(S) OF      1-      9 CONTIG. SECTORS =          0 FREE SECTORS. 0%
   176 BLOCK(S) OF     10-     99 CONTIG. SECTORS =       6048 FREE SECTORS. 0%
   137 BLOCK(S) OF    100-    999 CONTIG. SECTORS =      44528 FREE SECTORS. 1%
    48 BLOCK(S) OF   1000-   9999 CONTIG. SECTORS =     145264 FREE SECTORS. 4%
     7 BLOCK(S) OF  10000- 99999 CONTIG. SECTORS =     133024 FREE SECTORS. 4%
     2 BLOCK(S) OF 100000-AND UP CONTIG. SECTORS =    3439760 FREE SECTORS. 91%
```

Note that about 300 blocks of less than 1000 sectors are free. The above shows that the disk has been "checkerboarded" (fragmented) into about 360 areas. The final line shows, as a number, the approximate size of the largest contiguous free areas. (The "2 blocks" of 100,000+ sectors implies that the single biggest contiguous free area is at least 1,719,880 sectors in size.) (Don't those commas make the numbers easier to read?!) The initial output gave us the exact number, but again as a simple integer value ... with no obvious correlation to the size of the disk drive. Is that 1,977,64 sectors a significant portion of the disk or not?

Additionally, a complete list of in-use permanent, transient, and free disk areas can be obtained from the VOLUTIL SHOWUSAGE command:

```
:volutil

volutil: showusage 1

PERM, FREE, TRANS Space on LDEV 1:

processing ...
```

```
SECTOR ADDRESS        SIZE        SPACE_USED_BY
                   (in sectors)
--------------     ------------    -------------

            48              96     MMSAVE.MPEXL.SYS
        410752           11136     <transient space>
        436608            1328     <transient space>
        437936              32     MPEXLDIR.PUB.SYS
        437968             240     ISL.MPEXL.SYS
        438208              80     IOMAP.MPEXL.SYS
        438288             320     <free space>
        438608             768     SADPATCH.MPEXL.SYS
        ...
       5846640              32     UTAEPROC.TAE.TELAMON
       5846672         1977664     <free space>
volutil: exit
```

De-Frag/X's MAP command shows the disk usage in a graphical manner:

```
   De-Frag/X> map 1

------------------------------------------------------------------------
Ldev    1:   (Each chunk represents 588 pages, or 2.3 MB)
   [XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXxXxxXXxxXXXXXXXXXXX]  0
   [XXXXXXXXXXXXxxxxxxxxxxxxxxxxxxxxXxxxxxxxxxXXXXXXXXxXXXXXXXXXXXXXx]  1
   [xxXXXpxXX*X*xXXXXxxXXPXXPPPXpPXXXXXXXXXXXXXXXXXXXXXXXXXXXXXxXXXX]  2
   [XXPPxPPPPPPPPPPXXXXXXXXXXXxTTtxpx**XXXXXXXPXXXXXXXXXXXXPPPXXXxXX]  3
   [XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXPPXPPPXPPXXXX]  4
   [xXXXPPXXXXXXpPPXXXXXXXXXXXXppPPPppPPPPXXXX***xXpPppppPPPPPPpXPpP]  5
   [ppPpppPPpPppPppPXpxxpppppxppppPPxxPPpPXXppXPPPPPppPPPPPPXPPXXXXP]  6
   [XXXXPPPp**pPpPPp************************************************]  7
   [****************************************************************]  8
   [*******************************************pPp******************]  9
   [****************************************************************] 10
   [****************************************************************] 11
   [****************************************************************] 12
Col[0....+....10...+....20...+....30...+....40...+....50...+....60..] Row
Available Permanent Disk:  808 MB (disk size: 1,910 MB)
MAP_characters:
   Free       *   Permanent  P   Transient  T   unmovable  X
   part Perm  p   part Trans t   same_ldev  x
```

This provides an immediate visual appreciation of the state of disk fragmentation. A map of all disks, one line per disk, can also be obtained:

```
De-Frag/X> map all
                                                                  Disk Size of
                                                        Available Size  chunks
Ldev [ (usage map...)                                   ] PermMBs  MBs     MBs
   1 [XXXXXXXXXXXXXXXXXXXXXXpXXXX*********p************]    808   1,910   38.2
   4 [PPPp*p**p****************************************]  3,830   4,095   81.9
  11 [p*tt**t*p*************************pPPp***********]  1,871   2,033   40.7
  12 [p*********************************p**************]  1,989   2,033   40.7
  13 [pp******************************pp***************]  1,954   2,033   40.7
MAP_characters:
   Free       *   Permanent  P   Transient  T   unmovable  X
   part Perm  p   part Trans t   same_ldev  x
```

Note that the "Size of Chunks" column varies, since the disks show are of differing sizes.

Analyzing Fragmentation: File

Files are fragmented when their extents are not contiguous. (Note that a file of a single extent is not fragmented.) However, if a file consists of two extents, we can still differentiate between different degrees of "badness" for the fragmentation. If the two extents are on the same disk, at the opposite ends, then a serial read of the file will require one full-length motion of the disk head when switching from one extent to the other. If the second extent was in the middle of the disk, the "cost" of fragmentation isn't quite as high. If the second extent was on a different disk, then the cost is harder to estimate, since we don't know where the other disk's head is when we finish reading the first extent (on the first disk). However, it appears that this type of fragmentation is generally not quite as bad as having two non-contiguous extents on the same drive, even when they are close together, as an MPE "prefetch" could be scheduled to simultaneously fetch the two extents if they are on different disks (and could not do so if they are on the same disk).

De-Frag/X "scores" the fragmentation of a file, taking the above into consideration, as well as the size of the file. A file of 2 pages and with 2 extents (on the same disk) is 100% fragmented ... it can't get any worse! A file of 200 pages and with 2 extents isn't as badly fragmented, no matter where the two extents are located.

The FOS command LISTF (and LISTFILE) can be used to quickly determine whether or not a file is fragmented:

```
:listf sl.pub.sys,2
ACCOUNT=  SYS        GROUP=  PUB

FILENAME  CODE  ------------LOGICAL RECORD-----------  ----SPACE----
                  SIZE  TYP        EOF      LIMIT R/B  SECTORS #X MX

SL       * SL     128W  FB      125488     320000   1   126976 65  *
```

The above shows that SL.PUB.SYS has 65 extents, so it is fragmented.

FSCHECK can also be used to look at file fragmentation:

```
:fscheck.mpexl.telesup
displayextents sl.pub.sys

SECT_ADDR    SECTS_IN_EXTENT  VOL_SET_INDEX  FILE_SECT_OFFSET
---------    ---------------  -------------  ----------------

$000C9710        2048               1           $0001C800
$000C9F10        2048               1           $0001D000
$000CA710        2048               1           $0001D800
$000CAF10        2048               1           $0001E000
$000CB710        2048               1           $0001E800
...
```

Note that the extent information displayed is not in any particular order.

Finally, De-Frag/X shows the extents as well, but in logical (relative to the file) order:

```
De-Frag/X> displayextents sl.pub.sys

File: /SYS/PUB/SL  (65 extents)

   Ldev   Disk Page#   # Pages  File Page #
   ----   ----------   -------  -----------
       1      29,408        16            0
       1      29,452        16           16
       1      29,479        32           32
       1      31,494        64           64
       1      44,397       128          128 ,
       1      44,525       128          256 ,
       1      44,653       128          384 ,
       1      44,781       128          512 ,
       1      44,909       128          640 ,
       1      45,037       128          768 ,
       1      45,165       128          896 ,
       ...
       1      51,693       128        7,424 ,
       1      51,821       128        7,552 ,
       1      51,949       128        7,680 ,
       1      52,077       128        7,808

   # extents in file:          65
   # pages in file:   7,936 pages; 31.0 MBs
Space savable by TRIM:   93 pages;  0.3 MBs
% fragmented:    0.0+
```

For SL.PUB.SYS, De-Frag/X noticed that the extents are contiguous to each other, which is very unusual for files with multiple extents. (Note the "," after the "File Page#" value...this indicates that the extent with the "," on the line is immediately followed on disk by the extent shown on the next line.) De-Frag/X takes this into consideration when scoring the fragmentation of a file. In this case, the file was large (31 Mbs), but had only 65 extents (most of them contiguous), so the fragmentation score was non-zero, but barely so ... so small that it would appear as 0.0 if we formatted it with a ##.# format!

NOTE De-Frag/X will show the same data in "sectors" instead of "pages", if desired. I chose pages to be similar to FSCHECK's output. Also, both FSCHECK and De-Frag/X accept "DE" as a synonym for "DISPLAYEXTENTS".

De-Frag/X has a simple command to report just the fragmentation score for a file (or a fileset):

```
analyze @.pub.sys  min 10

Filename                      Frag%  #Pages      #MBs  #Extents
----------------------------  -----  ---------  ------  ---------
DTCCNF02.PUB.SYS               100%          2                  2
DTCSWB02.PUB.SYS                20%          6                  2
DTCSWE04.PUB.SYS                20%          6                  2
DTCSWG04.PUB.SYS                20%          6                  2
DTCSWH04.PUB.SYS                20%          6                  2
DTCSWJ04.PUB.SYS                50%          3                  2
DTCSWK04.PUB.SYS                20%          6                  2
HPOPTMGR.PUB.SYS                11%         10                  2
IODFAULT.PUB.SYS                10%         11                  2
IODFDATA.PUB.SYS                12%          9                  2
JSMAIN.PUB.SYS                  10%         11                  2
VERSION.PUB.SYS                 11%         10                  2

Fragmentation by fileset:     Frag%   #Pages      #MBs  #Extents   #Files
             files shown:       17%       86                  24       12
             files examined:     0%   66,647       260     1,911      831
   Space savable by TRIM: 1,755 pages; 6.8 MBs
```

The final line, "Space savable by TRIM" indicates that some of the files examined have disk space allocated beyond their EOFs. A "TRIM" command will tell MPE to discard this wasted space, without affecting the EOF or the LIMIT of the file.

Analyzing Fragmentation: Database

Database fragmentation is when the physical ordering (within a file) of entries in a dataset does not correspond to the most-desired logical ordering. "Most-desired" is mentioned because a dataset can generally be viewed in many ways. For this tutorial, we assume that the "primary path" is the most-desired logical ordering. Thus, if records with the key value of RED and the key value of BLUE (where COLOR is the primary path) occur in the order: RED, BLUE, RED, BLUE in the dataset, the dataset is fragmented. If they occur in the order: RED, RED, BLUE, BLUE (and RED comes before BLUE in the master dataset), then it is not fragmented.

Database fragmentation can be checked by running HOWMESSY (or the newer version, DBLOADNG). (Both programs are generally free of charge.)

Here is a HOWMESSY report on a sample database before any repacking:

```
HowMessy/XL (Version 2.0)                         Data Base: LOCAL.PUB.DEFRAG              Run on: FRI, AUG 11, 1995,  8:29 PM
 for IMAGE/3000 databases                         By Robelle Consulting Ltd.                                       Page:    2
                                            Secon-  Max
                  Type                 Load daries Blks  Blk                      Max    Ave       Std    Expd    Avg  Ineff  Elong-
  Data Set            Capacity Entries Factor (Highwater) Fact   Search Field   Chain  Chain      Dev  Blocks  Blocks Ptrs    ation
LOCAL-CONTROL      Det        1       1 100.0%  (      1)   1
LOCAL-USER         Man       55      53  96.4%  17.0%    8   5  USER-NUMBER         2   1.20     0.41    1.00    1.67  66.7%   1.67
DESIGNATE          Det       50       4   8.0%  (      5)  50 !PRINCIPAL-NUMBER    1   1.00     0.00    1.00    1.00   0.0%   1.00
                                                                DESIGNATE-NUMBER    1   1.00     0.00    1.00    1.00   0.0%   1.00
NAME-INDEX         Ato       55      31  56.4%  19.4%    0  29  NAME-PROBE          2   1.24     0.44    1.00    1.00   0.0%   1.00
NODE               Man       40      35  87.5%  37.1%    0  21  MAIL-NODE           4   1.59     0.96    1.00    1.38  23.1%   1.38
USER-XREF          Det       60      53  88.3%  (     55)  20  MAIL-NODE          18   4.08     5.19    1.00    1.62  15.1%   1.62
                                                                USER-NUMBER         1   1.00     0.00    1.00    1.00   0.0%   1.00
                                                              S!NAME-PROBE         13   1.71     2.19    1.00    1.23  13.2%   1.23
ITEM-HEADER        Man     7902    3951  50.0%   0.0%    1   9  ITEM-NUMBER         1   1.00     0.00    1.00    1.00   0.0%   1.00
ITEM-STRUCTURE     Det     8925    4451  49.9%  (   4452)  51 !FOLDER-ITEM-NO    117   3.75     6.48    1.01    1.82  21.8%   1.81 ***
                                                                CONTENT-ITEM-NO    41   1.13     1.08    1.00    1.06   4.9%   1.06
ITEM-CONTENT       Det    18628    9314  50.0%  (   9350)   2 !ITEM-NUMBER      1027   4.80    35.74    2.77    4.38  70.5%   1.58 ***
COMPUTER           Man       10       9  90.0%  44.4%    2   4  COMPUTER            3   1.80     1.10    1.00    2.00  50.0%   2.00
ROUTE              Det       28      14  50.0%  (     14)  14 !MAIL-NODE           1   1.00     0.00    1.00    1.00   0.0%   1.00
                                                                COMPUTER            4   1.75     1.04    1.00    1.00   0.0%   1.00
ENTRY-NAME-INDEX   Man      550       0   0.0%   0.0%    0  58  USER-KEYWORD        0   0.00     0.00    0.00    0.00   0.0%   0.00
ENTRY-INDEX        Man     2750     127   4.6%   0.0%    0  25  ENTRY-NUMBER        1   1.00     0.00    1.00    1.00   0.0%   1.00
USER-DATE          Man     5500     188   3.4%   1.6%    0  60  USER-NO-DATE        2   1.02     0.13    1.00    1.00   0.0%   1.00
OWNS               Det     2756     127   4.6%  (    135) 106 !ENTRY-NUMBER        1   1.00     0.00    1.00    1.00   0.0%   1.00
                                                                USER-NUMBER        71  15.88    22.79    1.00    1.25   1.6%   1.25
KEYWORD            Det     1100      28   2.5%  (     31)  44 !ENTRY-NUMBER        1   1.00     0.00    1.00    1.00   0.0%   1.00
                                                              S USER-NUMBER        28  28.00     0.00    1.00    1.00   0.0%   1.00
MAIL-ITEM-ENTRY    Det      636      34   5.3%  (     34) 106 !ENTRY-NUMBER        1   1.00     0.00    1.00    1.00   0.0%   1.00
                                                                ITEM-NUMBER         2   1.31     0.47    1.00    1.00   0.0%   1.00
MAIL-ITEM-SUBJ     Det     2765     126   4.6%  (    134)  35 !ENTRY-NUMBER        1   1.00     0.00    1.00    1.00   0.0%   1.00
SIMPLE-ENTRY       Det       18       9  50.0%  (      9)   9 !ENTRY-NUMBER        7   3.00     3.46    1.00    1.00   0.0%   1.00
INSERTION          Det     4134     335   8.1%  (    346)  53 !ENTRY-NUMBER       31   2.66     5.36    1.00    1.04   1.5%   1.04
                                                              S USER-NO-DATE        8   1.78     1.09    1.00    1.46  25.7%   1.46
FSC-PROGRAM        Man       16       0   0.0%   0.0%    0   9  FILE-FORMAT         0   0.00     0.00    0.00    0.00   0.0%   0.00
NODE-NODE          Det       24      12  50.0%  (     15)  12 !FROM-NODE           1   1.00     0.00    1.00    1.00   0.0%   1.00
                                                                TO-NODE            12  12.00     0.00    1.00    2.00   8.3%   2.00
```

The "Ineff Ptrs" (Inefficient Pointers) column is the one that indicates, roughly, the logical fragmentation of the dataset. For this tutorial, we will be looking at the datasets ITEM-HEADER and ITEM-CONTENT. Note that the "Ineff Ptrs" for those two are 21% and 70%, respectively. A value of 100% would mean that for a given chain, no two logically contiguous records are ever physically contiguous.

Correcting Fragmentation

This section discusses correction of the three types of fragmentation: disk, file, and database.

Correcting Fragmentation: Disk

FOS provides two mechanisms to defragment a disk drive: STORE/RESTORE and VOLUTIL: CONTIGVOL.

The STORE/RESTORE solution is the slowest, but has been available (free) since MPE was released. In this, a STORE of all files is done, followed by a "PURGE" of most files (I generally avoid purging files in @.PUB.SYS), followed by a RESTORE of all files. This results in disk drives with reasonably compact in-use areas, with most of the free space at the end of the drives. Of course, this is a tedious process, at best.

With the release of MPE/iX 5.0, FOS now includes the CONTIGVOL command in the VOLUTIL utility. According to HP, CONTIGVOL is not a disk "condense" utility ... its purpose in life is to create a large contiguous free space on ldev 1.

Here's an example of using VOLUTIL's CONTIGVOL to make a larger contiguous free area on ldev 1:

```
volutil: contigvol 1

*WARNING: *** Running CONTIGVOL on a busy system may cause *** WARNING*
*WARNING: *** "OUT OF DISK SPACE" errors temporarily on    *** WARNING*
*WARNING: *** specified LDEV/Volume.                       *** WARNING*

*Verify: 1977664 contiguous sectors available on ldev 1. Continue [Y/N] ?yt
Processing Labels on Ldev 13 ......
Processing Labels on Ldev 12 ......
Processing Labels on Ldev 11 ......
Processing Labels on Ldev 1 ......
Percent Complete 10
Percent Complete 20
Percent Complete 30
Percent Complete 40
Percent Complete 50
Percent Complete 60
Percent Complete 70
Percent Complete 80
Percent Complete 90
Number of Extents Moved 1036
Maximum Contiguous Sectors Free 3584816
```

De-Frag/X has a "CONDENSE" command, which acts a little differently. Instead of simply trying to create a single large hole, the condense command attempts to squeeze out all of the "holes" (free areas) between chunks of allocated disk space (similar to the MS-DOS DEFRAG command). The desired goal is to have the first X% of the disk fully occupied with permanent files (with no "holes), and the rest of the disk free.

Both De-Frag/X and CONTIGVOL will refuse to move in-use files, since that could result in major problems. Also, both programs refuse to move files off an ldev if they were specifically placed on that ldev. For example, the command:

```
build foo; dev=1 ...
```

means that you want the file's disk to be on ldev 1, no matter what!

In addition to CONDENSE, De-Frag/X has a MAKEROOM command, which allows you to move a specified number of megabytes of files off a particular ldev (and onto a specified list, or simply onto any other disks in the volume set). This command, like the CONTIGVOL command, can be used to make space available on ldev 1 for an UPDATE, if necessary.

Finally, De-Frag/X has a BALANCE command, which is useful when a new disk is added to a volume set. BALANCE will move some files from heavily used disks onto less used disks, aiming at balancing the load (and therefore, hopefully, the I/O traffic) across all of the disks in a volume set.

Correcting Fragmentation: File

A file can be defragmented via FOS tools by simply "copying" it:

```
:rename myfile, foo
:copy foo, myfile
:purge foo
```

The COPY command will generally copy the file into a single extent. File equates can be used to control which ldev (or volume class) the file is placed onto.

NOTE To make a copy of in-use CM program files or CM SL files, you will need to issue a file equate and do the COPY slightly differently.

Here’s an example for copying an in-use SL:

```
:file myfile; lock
:copy *myfile, foo
:reset myfile
```

then, at a later time:

```
:purge myfile
:rename foo, myfile
```

If the POSIX shell is used (SH.HPBIN.SYS), it is possible to rename in-use files (via the "mv" command).

De-Frag/X can also defragment a single file:

```
De-Frag/X>  defragment myfile
```

Or a fileset:

```
De-Frag/X>  defragment myfi@.@
```

Correcting Fragmentation: Database

Database fragmentation can be corrected with the FOS tools DBUNLOAD/DBLOAD, by physically copying all of the database data to tape, erasing the database, and reloading it from tape. This is a time consuming, error prone task.

Several products exist to repack a database. The original and most powerful is Adager, which has a "repack" command.

The following report was run after repacking the sets "item-structure" and "item-content" (of the database shown before) along their primary paths.

```
HowMessy/XL (Version 2.0)                        Data Base: LOCAL.PUB.DEFRAG            Run on: FRI, AUG 11, 1995, 10:39 PM
 for IMAGE/3000 databases                        By Robelle Consulting Ltd.                                     Page:    1
                                           Secon-  Max
                 Type                Load daries Blks  Blk                    Max    Ave      Std    Expd     Avg  Ineff  Elong-
   Data Set         Capacity Entries Factor (Highwater) Fact   Search Field  Chain  Chain     Dev   Blocks  Blocks Ptrs    ation
LOCAL-CONTROL     Det       1       1 100.0%  (       1)   1
LOCAL-USER        Man      55      53  96.4%  17.0%     8   5  USER-NUMBER        2   1.20    0.41    1.00    1.67  66.7%   1.67
DESIGNATE         Det      50       4   8.0%  (       5)  50 !PRINCIPAL-NUMBER   1   1.00    0.00    1.00    1.00   0.0%   1.00
                                                               DESIGNATE-NUMBER   1   1.00    0.00    1.00    1.00   0.0%   1.00
NAME-INDEX        Ato      55      31  56.4%  19.4%     0  29  NAME-PROBE         2   1.24    0.44    1.00    1.00   0.0%   1.00
NODE              Man      40      35  87.5%  37.1%     0  21  MAIL-NODE          4   1.59    0.96    1.00    1.38  23.1%   1.38
USER-XREF         Det      60      53  88.3%  (      55)  20  MAIL-NODE         18   4.08    5.19    1.00    1.62  15.1%   1.62
                                                               USER-NUMBER        1   1.00    0.00    1.00    1.00   0.0%   1.00
                                                             S!NAME-PROBE        13   1.71    2.19    1.00    1.23  13.2%   1.23
ITEM-HEADER       Man    7902    3951  50.0%   0.0%     1   9  ITEM-NUMBER        1   1.00    0.00    1.00    1.00   0.0%   1.00
ITEM-STRUCTURE    Det    8928    4451  49.9%  (    4451)  72 !FOLDER-ITEM-NO   117   3.75    6.48    1.00    1.04   1.0%   1.04 ***
                                                               CONTENT-ITEM-NO   41   1.13    1.08    1.00    1.10   8.8%   1.10
ITEM-CONTENT      Det   18628    9314  50.0%  (    9314)   2 !ITEM-NUMBER     1027   4.80   35.74    2.77    2.90  39.5%   1.04 ***
COMPUTER          Man      10       9  90.0%  44.4%     2   4  COMPUTER           3   1.80    1.10    1.00    2.00  50.0%   2.00
ROUTE             Det      28      14  50.0%  (      14)  14 !MAIL-NODE          1   1.00    0.00    1.00    1.00   0.0%   1.00
                                                               COMPUTER           4   1.75    1.04    1.00    1.00   0.0%   1.00
ENTRY-NAME-INDEX  Man     550       0   0.0%   0.0%     0  58  USER-KEYWORD       0   0.00    0.00    0.00    0.00   0.0%   0.00
ENTRY-INDEX       Man    2750     127   4.6%   0.0%     0  25  ENTRY-NUMBER       1   1.00    0.00    1.00    1.00   0.0%   1.00
USER-DATE         Man    5500     188   3.4%   1.6%     0  60  USER-NO-DATE       2   1.02    0.13    1.00    1.00   0.0%   1.00
OWNS              Det    2756     127   4.6%  (     135) 106 !ENTRY-NUMBER       1   1.00    0.00    1.00    1.00   0.0%   1.00
                                                               USER-NUMBER       71  15.88   22.79    1.00    1.25   1.6%   1.25
KEYWORD           Det    1100      28   2.5%  (      31)  44 !ENTRY-NUMBER       1   1.00    0.00    1.00    1.00   0.0%   1.00
                                                             S USER-NUMBER       28  28.00    0.00    1.00    1.00   0.0%   1.00
MAIL-ITEM-ENTRY   Det     636      34   5.3%  (      34) 106 !ENTRY-NUMBER       1   1.00    0.00    1.00    1.00   0.0%   1.00
                                                               ITEM-NUMBER        2   1.31    0.47    1.00    1.00   0.0%   1.00
MAIL-ITEM-SUBJ    Det    2765     126   4.6%  (     134)  35 !ENTRY-NUMBER       1   1.00    0.00    1.00    1.00   0.0%   1.00
SIMPLE-ENTRY      Det      18       9  50.0%  (       9)   9 !ENTRY-NUMBER       7   3.00    3.46    1.00    1.00   0.0%   1.00
INSERTION         Det    4134     335   8.1%  (     346)  53 !ENTRY-NUMBER      31   2.66    5.36    1.00    1.04   1.5%   1.04
                                                             S USER-NO-DATE       8   1.78    1.09    1.00    1.46  25.7%   1.46
FSC-PROGRAM       Man      16       0   0.0%   0.0%     0   9  FILE-FORMAT        0   0.00    0.00    0.00    0.00   0.0%   0.00
NODE-NODE         Det      24      12  50.0%  (      15)  12 !FROM-NODE          1   1.00    0.00    1.00    1.00   0.0%   1.00
                                                               TO-NODE           12  12.00    0.00    1.00    2.00   8.3%   2.00
```

Note that the "Inefficient Pointers" for the primary paths for the two sets went from 21.8 and 70.5% to 1.0 and 39.5%, respectively. Note: HOWMESSY and DBLOADNG don't take into account some realities of life under MPE/iX, partially betraying their origin as tools written for MPE V (and earlier!). In particular, on MPE/iX, it isn't bad to have a chain cross a "block" (a file system record). Instead, the real penalty is if a chain crosses a page boundary , because then it may cause a page fault. HOWMESSY/DBLOADNG currently don't take this into consideration. Nevertheless, the result shows that the Adager repack of the two sets put the detail's records into a physical order (within the file) that more closely matches the logical order. I.e., if an application serially reads the associated master datasets, and for each entry reads the entire chain from the detail, the detail entries are in 100% optimum order (as a result of repacking) ... or, to say it somewhat differently, the dataset has been logically defragmented.

Conclusion

The three types of fragmentation: disk, file, and database, can be examined, and analyzed with various tools. Fragmentation can be controlled, as desired. The primary measurable performance impact fragmentation causes is hard to measure...database fragmentation's effect on performance is relatively straightforward to measure, file fragmentation's effect harder, and disk fragmentation's effect (except for the obvious impact on UPDATEs) is the hardest to measure.

I'll be reporting on ongoing efforts to characterize the full performance effect of the various kinds of fragmentation ... stay tuned to HP3000-L2!

Footnotes

1 De-Frag/X is a product of Lund Performance Solutions. The primary author is Stan Sieler (me), of Allegro Consultants, Inc. I use De-Frag/X in many examples because of my familiarity with it.

One other "defrag"-type product is available, 9.1 from Bradmark. This is a program that can be purchased independently of any other products, if desired. I am, however, relatively unfamiliar with 9.1, and as such will not be giving examples of it.

Users interested in purchasing such products should always consider all alternatives. Additionally, regardless of the product (De-Frag/X, 9.1, or VOLUTIL), I give the same advice Microsoft and Symantec give before you use their "defrag" products on a PC: check the file system integrity first (CHKDSK or SCANDISK on DOS, FSCHECK.MPEXL.TELESUP on MPE/iX), backup the system, and then defragment!

2 HP3000-L is the mailing list devoted to the HP3000. It is a reflection of the Usenet news group "comp.sys.hp.mpe". You can subscribe to HP3000-L for free, by sending an email message as follows:

To: listserv@raven.utc.edu

Subject: hi (anything at all, actually, since this is ignored)

Body: subscribe HP3000-L firstname lastname